October 1981

Report. No. STAN-CS-81-898

Also numbered:
CSL TR-222

Separability As A Physical Database Design Methodology

by

K. Whang

G. Wiedcrhold

D. Sagalowicz

Department of Computer Science

Stanford University
Stanford, CA 94305

Separability As A Physical Database Design Methodology

17 January 1982

by

Kyu-Young Whang
Gio Wiederhold

Computer Systems Laboratory
Electrical Engineering Department,
Computer Science Department,
Stanford University,
Stanford, California 94305

and

Daniel Sagalowicz

Artificial Intelligence Center,
SRI international,
Menlo Park, California 94025

Table of Contents

1. Separability – An Approach to Physical Database Design

1.1 introduction

Problems of access path selection in large integrated databases can be approached from two standpoints. Query optimization seeks the optimal selection of access paths for a specific query being processed -given a certain structure of the underlying physical database [SMI 75] [PEC 75] [GOT 75] [BLA 76] [YAO 79] [SEL 79]. On the other hand, design of a physical database is concerned with the optimal configuration of physical file and access structures-given the logical access paths that represent the interconnections among objects in the data model; the usage patterns of those paths; the organizational *characteristics* ***of the*** *data* stored in the files; the various features of the particular DBMS such as available access structures (indexes, links, hashed organization, clustering of records, etc.) [HSI 70] [CAR 75] [SCH 75] [SEV 75] [HAM 76] [YAO 77] [BAT 80]. Throughout this paper ***we*** use the term ***access*** *configuration* to mean the aggregate of access structures assigned to a relation or to the whole database.

Most past research directed toward optimal design of physical databases has concentrated on single-file cases. This research must be extended to the design of the access configuration of multifile databases. Although some efforts have been devoted to multifile cases [GAM 77] [BAT 80] [KAT 80], the approaches employed fall far short of accomplishing automatic design of optimal physical databases.

In this paper we discuss the issues involved in designing the access configuration of a physical database so as to minimize the number of disk accesses for queries and updates. Our approach is somewhat formal and mathematical, deliberately avoiding excessive reliance on heuristics. Our purpose is to render the whole design phase manageable and to facilitate understanding of the underlying mechanisms.

By analyzing an important set of join methods possessing the property we call *separability*, we shall prove that optimal design of the access configuration of a multifile database can be reduced to the collective optimal designs of individual relations. In this paper we restrict the available join methods to this set to make the

whole approach formally manageable. Extensions to other join methods will be mentioned briefly. The main idea is to set up a ***basic design*** *methodology* in accordance with a formal method that includes a large subset of practically important join methods, and then, using some straightforward heuristics, extend this ***basic design methodology*** to include other join methods as well.

Section 1.2 introduces several key assumptions, while Section 1.3 describes applicable join methods of interest. In Section 1.5, the design theory will be developed by using the simple cost model introduced for the examples in Section 1.4. A design algorithm based on the theory will be introduced in Section 1.6. Extensions of our approach are mentioned, briefly in Section 1.7.

1.2 Approaches and Assumptions

The design of an optimal physical database is complex for a number of reasons- two of which we shall discuss here. First, we may have several types of access structures available as options. Although some generalized formulas for determining access cost have been devised for certain kinds of file structures [HSI 70] [SEV 75] [YAO 77] [BAT 80], it is generally difficult to use them for the selection of optimal file structures without an exhaustive search among all possible alternatives. It therefore becomes necessary to accomplish a judicious separation of design steps and to develop interfaces that will minimize interactions among those steps.

The second source of complexity addressed is the interaction among the access structures assigned to different relations. There are various techniques available, especially join methods, for processing a query – and the choice frequently depends on the access structures available on more than one relation. Therefore, the processing cost of a query associated with one relation depends upon other interacting relations. It is the purpose of this paper to provide a mechanism for coping with these interactions during the design phase.

We choose a relational DBMS and start with the indexes and the clustering property of a single relation as the initially available access structures. The link structure [BLA 76] will be included as an extension of the basic result by using heuristics. Clustering of two or more relations, as in many hierarchical organizations, is

not considered. We also assume that all TID (tuple identifier) manipulations can be performed in the main memory without any need to perform I/O accesses.

The database is assumed to reside on disklike devices. Physical storage space for the database is divided into units of fixed size called blocks [WIE 77]. The block is not only the unit of disk allocation, but is also the unit of transfer between main memory and disk. We assume that a block that contains tuples of a relation contains only the tuples of that relation. Furthermore, we assume that the blocks containing tuples of a relation, which comprises a file, can be accessed serially. However, the blocks do not have to be contiguous on the disk.[2]

In principle, we assume that a relation is mapped into a single file. Accordingly, from now on, we shall use the terms *file* and *relation* interchangably. This does not mean, however, that we exclude the possibility of storing prejoined forms of relations directly in the physical database. We believe this can be considered in a separate refining phase after the basic design has been obtained.

We shall develop a simple cost model of the storage structure in Section 1.4, and shall use various cost formulas based on this model. For convenience, we assume that the size of the available buffer is one block. However, the theory we develop is not dependent on the buffer size, if we ignore the contention among many transactions in the buffer pool at query-processing time. Not encorporated in our theory are either the effect of the contention in the buffer pool and the scheduling algorithm.

We consider only one-to-many (including one-to-one) relationships between relations. It is argued in Appendix A that many-to-many relationships between relations are less important for the optimization. Note that here we arc dealing with relationships in relational rcprescntations, so that a relationship among distinct entity sets at the conceptual level is often structured with an additional intermediate relation.

Finally, we are considering only one-variable or two-variable queries in this paper. For a query of more

[2] For example, blocks of a file can be spread all over the disk while they are connected as a linked list or linked implicitly by a file map.

than two variables, a heuristic approach can be employed to decompose it into a sequence of two-variable queries (These correspond to one-overlapping queries in [WON 76]).

1.3 Query Evaluation

The class of queries we shall be considering is shown in Figure l-l. The conceptual meaning of this class of queries is as follows. Tuples in relation R_1 are restricted by restriction predicate P,. Likewise tuples in relation R_2 are restricted by predicate P_2 The resulting tuples from each relation are joined according to the join predicate $R_1.A = R_2.B$, and the result projected over the columns $a_1...a_n$. We shall call the columns that are involved in the restriction predicates ***restriction columns,*** and those in the join predicate ***join columns. The*** actual implementation of this class of queries does not have to follow the order specified above as long as it produces the same result.

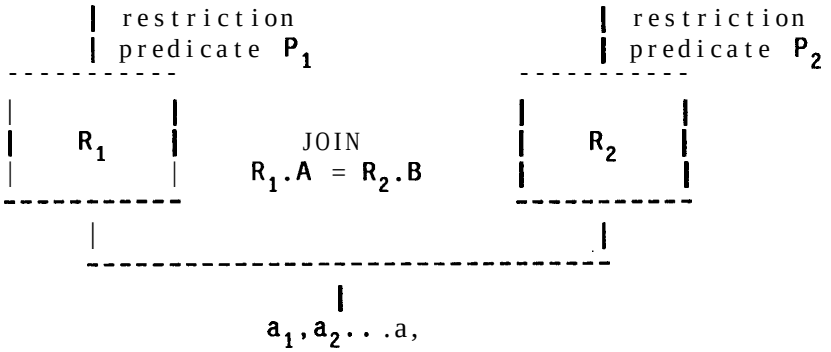

Figure l-l: General Class of Queries to be Considered.

Query evaluation algorithms, especially for two-variable queries, have been studied in [BLA 76] and [YAO 79]. The algorithms for evaluating queries differ significantly in the way they use join methods. Before discussing the various join methods, let us define some terminology.

Given a query, an index is called a ***join index*** if it is defined for the join column of a relation. Likewise, an index is called a ***restriction index*** if it is defined for a restriction column. We shall use the terrn *subtuple* for a tuple that has been projected over some columns. The restriction predicate in a query for each relation is decomposed into the form Q1 A Q2, where Q1 is a predicate that can be processed by using indexes while Q2

cannot. Q2 must then be resolved by accessing individual records. We shall call Q1 ***the index-processible predicate*** and Q2 the ***residual predicate.***

Some algorithms of interest for processing joins are summarized briefly hereunder (see also [BLA 76] [SEL 79]):

- ***Join Index*** *Method*: This method presupposes the existence of join indexes. For each relation, the TIDs of tuples that satisfy the ***index processible predicates*** are obtained by manipulating the TIDs from each index involved; the resultant TIDs are stored in temporary relations R,' and R,'. TID pairs with the same join column values are found by scanning the join column indexes according to the order of the join column values. As they are found, each TID pair (TID,, TID_2) is checked to determine whether TID, is present in R,' and TID_2 in R_2'. If they arc, the corresponding tuple in one relation, say R,, is retrieved: When this tuple satisfies the *residual* ***predicate*** **for R,, the** corresponding tuple in the other relation R_2 is retrieved and the residual predicate for R_2 is checked. If qualified, the tuples are concatenated and the subtuple of interest is constructed.

- *Sort-Merge Method*: The relations R_1 and R_2 are scanned- either by using restriction indexes, if there is an index-processible predicate in the query, or by scanning the relation directly- and temporary relations T_1 and T_2 are created. Restrictions, partial projections, and the initial step of sorting are performed while the relations are being initially scanned and stored in T_1 and T_2. T_1 and T_2 are sorted by the join column values. The resulting relations are scanned in parallel **and** the join is completed by merging matching tuples.

- ***Combination of the Join Index Method and the Sort-Merge Method:*** One relation, say R1, is sorted as in the sort-merge method and stored in T,. Relation R_2 is processed as in the join index method, storing the TIDs of the tuples that satisfy the index processible predicates in R_2'. T_1 and the join column index of R_2 are scanned according to the join column values. As matching join column values arc found, each TID from the join index of R_2 is checked against R_2'. If it is in R_2', the corresponding tuple in R_2 is retrieved and the residual predicate for R_2 is checked. If qualified, the tuples are concatenated and the subtuple is constructed.[3]

- *Inner/Outer-Loop* ***Join*** *Method*: In the two join methods described above, the join is performed by scanning relations in the order of the join column values. In the inner/outer-loop join, one of the relations, say R,, is scanned without regard to order, either by using restriction indexes or by scanning the relation directly, and, for each tuple of R_1 that satisfies predicate P_1, the tuples of relation R_2 that satisfy predicate P_2 and the join predicate are retrieved and concatenated with the tuple of R,. The subtuples of interest are then projected upon the result.[4]

---

[3] In actual implementation, the combinations of join methods can be either coded separately or programmed to be dynamically synthesized at query-processing time. A specific combination of join methods will be selected or synthesized according to the result of the query optimization which, given a fixed structure of the physical database, will find the best evaluation method for a query.

[4] One of the advantages of this join is that it does not require scanning a relation in a sorted order. Furthermore, this method is often better than the join index method if the number of qualified tuples retrieved from R_2 is small, making it unnecessary to scan the entire join index for R_2. On the other hand, if a large portion of relation R_2 satisfies the predicates, this method will cause repeated accesses of the index tree-which will be more costly than a single scan of the index.

- *Multiple-Pass **Method:*** One of the relations participating in the join, say R_1, is scanned, the tuples are obtained, restricted, projected, and inserted into a data structure T_1, whose size is constrained to fit in the available main store. If space in main store is available to insert the resulting subtuple, r, this is done. If space is not available, but the join column value in r is less than the current highest join column value in T_1, the subtuples with the highest join column value in T_1 are then deleted and r is inserted. Otherwise r is not inserted at all. After T_1 has been formed, R_2 is scanned by using an appropriate access path, and every tuple of R_2 that satisfies the predicate is concatenated (if possible) with the appropriate subtuples in T_1 and the result projected. If there are more qualified tuples in R_1 than can fit in the main store for T_1, another scan of R_1 is done to form a new T_1 consisting of subtuples with join column values greater than the current highest. R_2 is also scanned again and the whole process repeated. This method is very fast if only one pass is needed. But processing time increases rapidly when more passes are performed.

- ***Link- Based Join Method:*** This is conceptually similar to the inner/outer-loop join method, but it takes advantage of existing links [BLA 76] between the two relations. The use of links will be mentioned briefly as an extension of our basic methodology.

Let us note that, in the combination of the join index method and the sort-merge method, the operation performed on either relation is identical to that performed on one relation-whether in the join index method or the sort-merge method. We call the operations performed on each relation ***join index method (partial)*** or ***sort-merge method (partial),*** respectively; whenever no confusion arises, we call these operations simply ***join index method*** or ***sort merge method.*** According to these definitions, the join index method actually consists of two join index methods (partial) and similarly the sort-merge method consists of two sort-merge methods (partial).

1.4 Cost Model of the Storage Structure

To calculate the cost of evaluating a query, we need a proper model of the underlying storage structure and its corresponding cost formula. Although the theory does not depend on the specifics of cost models, it is helpful to have a simple cost model for illustrative purposes.

We assume that a B-tree index [BAY 72] can be defined for a column or for a set of columns of a relation. The leaf-level of the index consists of pairs (key and TID) for every tuple in that relation. The leaf-level blocks are chained according to the order of indexed column values, so that the index can be scanned without traversing the index tree. Entries having the same key value are ordered by TID.

An index is called a *clustering index* if the relation for which this index is defined is physically clustered according to the index column values. With a clustering index, we assume that no block is fetched more than once when tuples with consecutive values of the indexed column are retrieved.. Except for this ordering property, no other difference in the structure is assumed between a clustering and a nonclustering index. The clustering property can greatly reduce the access cost, especially when a join column has a clustering index. Unfortunately, only one column of a relation can have the clustering property, since clustering requires a specific order of records in the physical file. One of the objectives of designing optimal physical databases is to determine which column will be assigned the clustering property.

The access cost will be measured in terms of the number of I/O accesses. The following notation will be used throughout this paper:

n_R : Number of tuples in relation R (cardinality)
p_R : Blocking factor of a block containing tuples of relation R.
L_I : Blocking factor of an index block containing index I.
F_C : Selectivity of the column used or the index thereof.
m_R : Number of blocks in relation R, which is equal to n_R/p_R.

By using the simplified model above, the cost of various operations can be obtained as follows:

- Relation Scan Cost - Cost for serially accessing all the blocks containing the tuples of a relation:

  $RS(R) = n_R/p_R = m_R$

- Index Scan Cost - Cost for serially accessing the leaf- level blocks of an entire index:

  $IS(I,R) = n_R/L_I$

- Index Access Cost - Cost for one access of the index tree from the root:

  $IA(I,R) = \log_{L_I} (n_R/L_I) + F_I \times n_R/L_I$

- Sorting Cost - Cost for sorting a relation, or a part thereof, according to the values of the columns of interest:

  $SORT(NB) = 2 \times NB \times \log_z NB$

  Here we assume that a z-way sort-merge is used for the external sort [KNU-b 73]. NB is the number of blocks in the temporary relation containing the subtuples to be sorted after restriction and projection have been resolved. It will be noted that SORT(NB) does not include the initial scanning time to bring in the original relation, while it does include the time to scan the temporary relation for the actual join after sorting (see [BLA 76]).

1.5 Design Theory

In this section we develop a theory for the design of optimal physical da&bases. We shall seek to facilitate comprehension through a series of examples and by case analysis, using the cost model developed in Section 1.4. Observations resulting from this procedure are formalized and proved in Section 1.5.4.

Our approach to physical database design is based on the premise that at execution time the query processor will choose the best processing method for a given query. We call this processor an ***optimizer.*** Since the behavior of the optimizer at execution time affects the physical database design critically, we investigate this issue and discuss how it is related to the design.

Since the set of join methods consisting of the join index method, the sort-merge method, and the combination of the two possesses the special property, called *separability* which we shall define later, we regard only those methods as being available for the design theory (the inner/outer-loop join method, the multiple-pass method, and the link-based method are nonseparablc join methods with respect to this separable set).

We define the influence of the restriction on one relation to the number of tuplcs to be retrieved in the other relation the *coupling effect* (which is similar in concept to the ***feedback*** mentioned in [YAO 79]). Starting with a case in which coupling effects between relations are not considered, we then proceed to those cases in which they are included.

1.5.1 Cases without coupling effects

Example 1: Figure l-2 describes two relations R_1 and R_2 with their access configurations. Dashed lines (/) reprcsont clustering indcxcs, the dotted lines (:) nonclustcring indcxcs. Columns without either type of line have no indexes defined for them. We would like to find the best method of evaluation-which the optimizer would choose at query-processing time, for the following query:

SELECT A_1, A_2, B_2
FROM R_1, R_2
WHERE $R_1.A_2 = 'a_2'$ **AND**
$R_2.B_2 = 'b'$ **AND**
$R_1.A_1 = R_2.B_1$

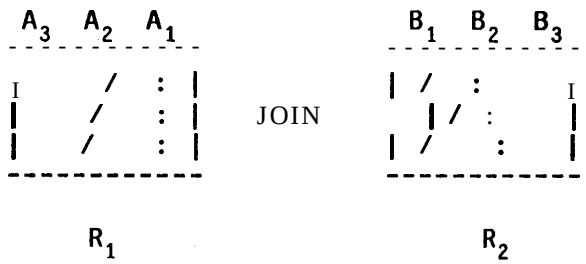

Figure 1-2: Relations R_1 **and** R_2

For this example only, it is also assumed that all the tuples in each relation participate in the join.

Given these assumptions, the optimizer could try all the possible combinations of the join methods, evaluate the cost of each, and then select the one that costs the least. We have here the following combinations:

R_1	R_2
1. Join index method (partial)	Join index method (partial)
2. Sort-merge method (partial)	Sort-merge method (partial)
3. Join index method (partial)	Sort-merge method (partial)
4. Sort-merge method (partial)	Join index method (partial)

Using the cost model developed in Section 1.4, the following formulas give the cost (number of block accesses) for each of the four cases above. In each formula the first and second bracketed expressions represent the cost of accessing relation R_1 and R_2 respectively. Bracketed expressions in the formulas are given arbitrary values for illustrative purposes. Those expressions whose form is identical are given the same value.

$$\text{Cost} = [IA(I_{A2}, R_1) + IS(I_{A1}, R_1) + F_{A2} \times n_{R1}] + [IA(I_{B2}, R_2) + IS(I_{B1}, R_2) + b(m_{R2}, p_{R2}, F_{B2} \times n_{R2})] \quad : 100 + \quad : 20 \qquad (1.1)$$

$$\text{cost} = [IA(I_{A2}, R_1) + F_{A2} \times m_{R1} + SORT(F_{A2} \times H_{R1} \times m_{R1})] + [IA(I_{B2}, R_2) + b(m_{R2}, p_{R2}, F_{B2} \times n_{R2}) + SORT(F_{B2} \times H_{R2} \times m_{R2})] \quad : 60 + : 50 \qquad (1.2)$$

$$\text{Cost} = [IA(I_{A2}, R_1) + IS(I_{A1}, R_1) + F_{A2} \times n_{R1}] + [IA(I_{B2}, R_2) + b(m_{R2}, p_{R2}, F_{B2} \times n_{R2}) + SORT(F_{B2} \times H_{R2} \times m_{R2})] \quad : 100 + : 50 \qquad (1.3)$$

$$\text{Cost} = [IA(I_{A2}, R_1) + F_{A2} \times m_{R1} + SORT(F_{A2} \times H_{R1} \times m_{R1})] + [IA(I_{B2}, R_2) + IS(I_{B1}, R_2) + b(m_{R2}, p_{R2}, F_{B2} \times n_{R2})] \quad : 60 + : 20 \qquad (1.4)$$

Here b(m,p,k) is a function that provides the number of block accesses, where k is the number of tuples to be retrieved in TID order. An exact form of this function and various approximation formulas are summarized in Chapter 2. The function is approximately linear in k when $k \ll n$, and approaches m as k becomes large. A familiar approximation suggested by Cardenas [CAR 75] is $b(m,p,k) = m\,[1 - (1 - 1/p)^k]$. F_{A2} and F_{B2} are the selectivities of the columns $R_1.A_2$ and $R_2.B_2$, respectively. In Equation (1.1), $F_{A2} \times n_{R1}$ and $b(m_{R2},p_{R2},F_{B2} \times n_{R2})$ represent the numbers of blocks accessed that contain data tuples of relation R_1 and R_2, respectively. Since retrieving tuples by scanning a nonclustering join index will access the tuples randomly, the same block will be accessed repeatedly if it contains more than one tuple. Therefore it is very likely that one block access is needed to retrieve each tuple. Hence we get $F_{A2} \times n_{R1}$ for the number of data blocks fetched from relation R,. Note that in this case the tuples cannot be accessed in TID order. For relation R_2, however, the join index is clustering and thus the tuples will be retrieved in TID order, even though they are selected randomly by the restriction. Therefore, even though a block contains more than one tuplc, in all likelihood each block will be fetched only once. We thus get $b(m_{R2},p_{R2},F_{B2} \times n_{R2})$ for the number of data blocks fetched from R_2, where $F_{B2} \times n_{R2}$ is the number of tuples selected by the restriction.

In Equation (1.2), $F_{A2} \times m_{R1}$ and $b(m_{R2},p_{R2},F_{B2} \times n_{R2})$ represent the numbers of blocks accessed during thc initial scan of thc relation prior to sorting. Since the restriction index is clustering in relation R,, the initial scan through this restriction index will access $F_{A2} \times m_{R1}$ blocks. In relation R_2, a nonclustering restriction index is used to access the relation initially. This restriction results in random distribution of TIDs of the qualified tuples over the blocks. Since these tuples are then accessed in TID order, the access cost is $b(m_{R2},p_{R2},F_{B2} \times n_{R2})$.

The factor H_{R2} used in the Equation (1.3) represents the projection effect upon relation R_2. Since the projection selects only part of the attributes from the relations, the tuple is usually smaller after projection. The time required to write the final result is not included, since it is the same regardless of the join method used.

With the specific values of the access cost given, Equation (1.4) gives the minimum access cost. We note that the access costs for each relation do not depend on any parameter of any other relation, and that each part of the cost of Equation (1.4) becomes the local minimum. That is, the first part of the cost incurred by accessing relation R_1 is the minimum of the costs of the join methods used for R,, while the second part is the minimum of those for R_2. This implies that the optimizer can determine the optimal join method on one relation without regard to any properties of other relations. □

The foregoing observation is extremely important because, if we can determine the optimal join method for one relation without regard to other relations, we can also use the following method to determine the optimal access configuration for the relation without regard to other relations:

1. try every possible access configuration for a relation in turn.
2. for a given access configuration, find the best evaluation method- which the optimizer would choose at query-processing time – for each given query (this corresponds to the query optimization problem).
3. then calculate the total cost for processing the queries, using their expected frequency of occurrence.
4. repeat this procedure for all other possible access configurations, finally selecting the one that yields the minimal total cost.

The result of this will be to reduce designing an optimal access configuration of a database to that of a single relation. Local optimal solutions for individual relations constitute an optimal solution for the entire database. However, the foregoing procedure of making an exhaustive search of all the possible access configurations could yet prove too costly. Therefore, in Section 1.6 we divide the design procedure into two parts: choice of the clustering column and index selection. We shall provide a clean interface between the two steps and discuss deviations from the true optimum.

It should be pointed out here that, despite our assumption that there is no coupling effect between the two relations and despite the fact that the above argument appears to follow directly from that assumption, .it will be shown, in the following discussion, that the problem is similarly reduced even when coupling effects are actually present.. Before furtherdiscussion, we need the following definition and example.

Definition 1: **The *join selectivity*** J(R,JP) of a relation R with respect to a join path JP is the' ratio of the number of distinct join column values of the tuples participating in the unconditional join to the total number of the distinct join column values of R. A ***join path*** is a set (R_1,R_1.A,R_2,R_2.B), where R_1 and R_2 are relations participating in the join and R_1.A and R_2.B are the join columns of R_1 and R_2, respectively. An ***unconditional join*** is a join in which the restrictions on either relation are not considered. □

Join selectivity is the same as the ratio of the number of tuples participating in the unconditional join to the total number of tuples in the relation (cardinality of the relation). Join selectivity is generally different in R_1 and R_2 with respect to a join path, as shown in the following example:

Example 2: Let us assume that the two relations in Figure l-3 have a l-to-N partial-dependency relationship. Partial dependency means that every tuple in the relation R_2 that is on the N-side of the relationship has a corresponding tuple in R,, but not vice versa [ELM 80]. Let us assume that 50% of the employees have at least one child each so that the tuples representing those. employees participate in the unconditional join. Every tuple in the children relation R_2 is assumed to have only one corresponding tuple in R_1 and all of them participate in the unconditional join according to the partial dependency. The join selectivity of the employees relation is then 0.5, while that of the children relation is 1.0 □

R_1: Employees(E#, Job, Age, Salary)
R_2: Children(E# , Name, Hair-color, Sex)

Figure 1-3: Employees and Children relations.

1.52 Cases with coupling effects

Let us investigate the four cases shown in Example l-using the same query, join methods, and access configuration defined as in Figure l-2, but now with coupling effects. In fact, we shall consider coupling effects throughout our subsequent discussions. We shall also assume that R_1 and R_2 have a l-to-N relationship (1 for R_1 and N for R_2).

Case 1: The join index method is applied to both relations R_1 and R_2. With coupling effect, the join will be performed as follows: If a tuple of relation R_1 does not satisfy the restriction predicate for R,, the corresponding tuples of R_2 that have the same join column values are not accessed. Hence, we have the coupling effect from R_1 to R_2. If there are only index-proccssible predicates in the query to be evaluated, the situation is then symmetric-in the sense that, for thc tuples in relation R_2 that do not satisfy the restriction predicate for R_2, the corresponding tuplcs of R_1 are not accessed either. We have this symmetry because we can resolve all index-processible predicates by using TIDs only, without any need to access the data tuples themselves.

Since both $R_1.A_2$ and $R_2.B_2$ have indexes defined for them, the restriction predicates in the WHERE clause are index-processible. Therefore, the cost of evaluating this query, including the coupling effect, will be as follows:

$$\begin{aligned}\text{Cost} = {} & [IA(I_{A2},R_1) + IS(I_{A1},R_1) + \{\langle J_1 \text{ X } b(1/F_{B1}, F_{B1} \text{ X } n_{R2}, \\ & F_{B2} \times n_{R2})/(1/F_{B1})\rangle \times F_{A2} \times n_{R1}\}] + \\ & [IA(I_{B2},R_2) + IS(I_{B1},R_2) + b(m_{R2},p_{R2},\{\langle J_2 \times F_{A2}\rangle \times F_{B2} \times n_{R2}\})]\end{aligned}$$

Here J_1 and J_2 represent the join selectivity of relations R_1 and R_2, respectively, for the join path considered. Expressions in the braces represent the numbers of data tuples accessed in relations R_1 and R_2, respectively. In the first part of the formula, the expression in thc braces simultaneously represents the number of blocks accessed in relation R,. This follows the argument shown in Example 1.

F_{B1} is the selectivity of column $R_2.B_1$ and $1/F_{B1}$ represents the number of groups' of tuples that have the same join column values in relation R_2-which is essentially the same as thc number of distinct join column values.

[5] Group here is very close in concept to *set occurrence* in CODASYL-type databases.

The expression $b(1/F_{B1}, F_{B1} \times n_{R2}, F_{B2} \times n_{R2})$ represents the number of groups selected by restriction F_{B2}. Although the b function estimates the number of block accesses in which a certain number of tuples are randomly selected, the same function is used for estimating the number of logical groups selected- if the latter are assumed to be of uniform size. Note that the clustering or nonclustering of tuples in a group is irrelevant. $F_{B1} \times n_{R2}$, the number of tuples in one logical group, plays a role similar to that of the blocking factor.

The expression $b(1/F_{B1}, F_{B1} \times n_{R2}, F_{B2} \times n_{R2})/(1/F_{B1})$ represents the ratio of the number of groups selected by restriction F_{B2} to the total number of groups in relation R_2. Since every tuple participating in the unconditional join in R_1 has a unique join column value and, accordingly, exactly one corresponding group in R_2 (let us recall that R_1 is on the l-side of the l-to-N relationship), this ratio correctly represents a special restriction upon R_1 caused by the coupling effect originating in R_2.[6]

In the second part of the cost formula, we simply use F_{A2} to represent the coupling effect directed from R_1 to R_2. Since in R_1 every tuple has a unique join column value, if a tuple is selected according to the restriction, the corresponding group in R_2 that has the same join column value (if it exists) will be selected on the basis of this special restriction resulting from the coupling effect. Hence, F_{A2} represents the ratio of the number of groups selected as a consequence of the coupling effect to the total number of groups in R_2 participating in the unconditional join. That ratio, in turn, has the same value as the ratio of tuples, selected according to the coupling effect, to the total number of tuples participating in the unconditional join in R_2. □

The coupling effect is formally defined as follows:

Definition 2: The ***coupling effect*** from relation R_1 to relation R_2, with respect to a type of query, is the ratio of the number of distinct join column values of the records of R,, selected according to the restriction predicate for R,, to the total number of distinct join column values in R,. ❑

[6] Note that this ratio could be very different from and is always larger than F_{B2}, especially when a group is large. The reason is that, if at least one tuple in a group is selected, the corresponding join column value and the corresponding tuple in R_1 are selected according to this special restriction resulting from the coupling effect.

If we assume that the join column values are randomly selected, the coupling effect from R_1 to R_2 is the same as the ratio of the number of distinct join column values of R_2 selected by the effect of the restriction predicate for R_1 to the number of distinct join column values in R_2 participating in the unconditional join.

Definition 3: A ***coupling factor*** Cf_{12} from relation R_1 to relation R,, with respect to a type of query, is the ratio of the number of distinct join column values of R,, selected by both the coupling effect from R_1 (through the restriction predicate for R_1) and the join selectivity of R,, to the total number of distinct **join** column values in R,. □

According to the definition, a coupling factor can be obtained by multiplying the coupling effect from R_1 to R_2 by the join selectivity of R,. This coupling factor contains all the consequences of the interactions of relations in the join operation, since it includes both coupling and joining filtering effects. Let us note that, although the coupling factor can be obtained in any case, it does not always contribute to the reduction of the tuples to be retrieved. We will see an example of this in Case 2 below. A coupling factor is said to be ***effective*** if the coupling effect actually contributes to the reduction of the tuples to be retrieved. In Case 1, the expressions in angle brackets represent the coupling factors from R_2 to R_1 and from R_1 to R,, respectively, for the type of query considered. By definition, different queries are of the same ***type*** if they are identical except for their literal values. The same applies to update transactions. For example, INSERT INTO R_1 <a,b> is of the same type as INSERT INTO R_1 <c,d>. Hence,

$$Cf_{12} = J_2 \times F_{A2},$$
$$Cf_{21} = J_1 \times b(1/F_{B1}, F_{B1} \times n_{R2}, F_{B2} \times n_{R2})/(1/F_{B1}).$$

One important observation here is that the coupling factors do not depend on the specific access structures present in either relation, nor on the specific join method selected, but rather (and solely) depend on the restriction and the data characteristics. Such characteristics include the side the relation is on in the l-to-N relationship, the average number of tuples in one group, and the join selectivity – which will be known **before** we start the design phase.

Note that the coupling factors differ according to the specific type of query being considered. Different

types of queries have different join paths and different combinations of columns in the restriction predicate – with consequently different selectivities for the calculation of coupling factors.

Now let us investigate the remaining cases in which coupling effects are present between relations.

Case 2: The sort-merge join method is applied to both relations, in the same situation as in Figure 1-2. The cost formula is then as follows:

$$\text{Cost} = [F_{A2} \times m_{R1} + \text{SORT}(F_{A2} \times H_{R1} \times m_{R1})] + [IA(I_{B2},R_2) + b(m_{R2},p_{R2},F_{B2} \times n_{R2}) + \text{SORT}(F_{B2} \times H_{R2} \times m_{R2})]$$

It will be noted that the coupling factors do not appear in the cost formula. This is because, when the sort-merge join method is used, an initial scan and the sort are performed before the join is resolved; indexes are not used any more while the join is being actually resolved, since the relation scan is performed upon the sorted temporary relations. The coupling effect can arise only when the join is being actually resolved and only when the join index is used. Thus, the coupling factor is not effective in this case.

Case 3: The sort-merge join method is used for R,, the join index method for R_2 – in the same situation as in Figure l-2. The join will be performed as described in Section 1.3, under the heading: Combination of the Join Index Method and Sort-Merge Method. Note that the coupling factor is effective from R_1 to R,. Thus, we obtain the following cost formula:

$$\text{Cost} = [F_{A2} \times m_{R1} + \text{SORT}(F_{A2} \times H_{R1} \times m_{R1})] + [IA(I_{B2},R_2) + IS(I_{B1},R_2) + b(m_{R2},p_{R2},Cf_{12} \times F_{B2} \times n_{R2})]$$

Case 4: The join index method is used on R,, the sort-merge method on R_2 – in the same situation as in Figure l-2. We obtain the following cost formula:

$$\text{Cost} = [IA(I_{A2},R_1) + IS(I_{A1},R_1) + Cf_{21} \times F_{A1} \times n_{R1}] + [IA(I_{B2},R_2) + b(m_{R2},p_{R2},F_{B2} \times n_{R2}) + \text{SORT}(F_{B2} \times H_{R2} \times m_{R2})]$$

1.5.3 Cases when restriction indexes are absent on one relation

All four cases that have been discussed so far assume the same situation as in Example l-except for inclusion of the coupling effect. We still have to consider more general cases in which restriction indexes are absent for the columns specified in the predicate of the query for one relation. The case in which the restriction indexes are absent in both relations will be treated in Section 1.5.4. For clarity of presentation, let, us define a shorthand notation for the cost formula.

Definition 4: Cost(R_k, Cf_{jk}, type-of-join) is the cost of a join operation associated with relation R_k when R_k has a coupling factor Cf_{jk} from R_j to R,, with respect to the query of interest, and the type-of- join is the join method used between R_k and R_j. □

Although costs differ for different access configurations, this shorthand notation for the cost function does not show that difference explicitly, because it is irrelevant to our subsequent discussions. Using this definition, cost formulas for the previous cases can be restated as

Case 1: $Cost(R_1,Cf_{21},Join\text{-}index) + Cost(R_2,Cf_{12},Join\text{-}index)$ (1.5)

Case 2: $Cost(R_1,Cf_{21},Sort\text{-}merge) + Cost(R_2,Cf_{12},Sort\text{-}merge)$ (1.6)

Case 3: $Cost(R_1,Cf_{21},Sort\text{-}merge) + Cost(R_2,Cf_{12},Join\text{-}index)$ (1.7)

Case 4: $Cost(R_1,Cf_{21},Join\text{-}index) + Cost(R_2,Cf_{12},Sort\text{-}merge)$ (1.8)

If there is no coupling effect between the two relations, as in the case of a query that does not impose a restriction on a relation, say R_2, then the coupling factor Cf_{21} simply becomes the join selectivity, J_1 – if the join index method is used for R,. The cost, in this case, will be $Cost(R_1,J_1,type\text{-}of\text{-}join)$. When the sort- merge join method is used for relation R_k the cost becomes $Cost(R_k,1,sort\text{-}merge)$. But it is identical to $Cost(R_k,Cf_{jk},sort\text{-}merge)$, because, as we observed in Case 2, the coupling factor is not used in the cost formula. According to the same argument, we conclude that the cost of the sort-merge join method can always be written as $Cost(R_k,Cf_{jk},sort\text{-}merge)$.

Case **1-A:** Let us assume that the join index method is used for both R_1 and R_2, in the same situation as in Figure l-2, except that the restriction index for column $R_1.A_2$ is missing. The join will be performed as follows. First the TID set R,' of the tuplcs that satisfy the restriction on R_2 is obtained by using the restriction

on column $R_2.B_2$. TID pairs that have the same join column values are found by scanning **the join column** indexes according to the order of join column values. **As** it is found, each TID pair (TID_1,TID_2) is checked to see if TID, is present in R,'. If it is, the corresponding tuple in relation R_1 is retrieved. If this tuple satisfies the restriction upon R,, the corresponding tuple in R_2 is also retrieved and concatenated, and the result projected. Note that the coupling factors are effective in both directions. Thus, the cost of evaluating the query will be

$$\begin{aligned}\text{Cost} &= [IS(I_{A1},R_1) + Cf_{21} \times n_{R1}] \\ &\quad + [IA(I_{B2},R_2) + IS(I_{B1},R_2) + b(m_{R2},p_{R2},Cf_{12} \times F_{B2} \times n_{R2})] \\ &= \text{Cost}(R_1,Cf_{21},\text{join-index}) + \text{Cost}(R_2,Cf_{12},\text{join-index}).\end{aligned}$$

Note that, since the restriction index on column $R_1.A_2$ is missing, the first part of the cost formula is different from that of Case 1, but the coupling factors remain the same. The case in which $R_2.B_2$ is absent instead of $R_1.A_2$ is treated similarly and will result in the same formula in the shorthand notation.

Case 2-A: The sort-merge method is used for both R_1 and R_2 in the same situation as in Figure 1-2, except that the restriction index on the column $R_1.A_2$ is missing. The cost formula becomes

$$\begin{aligned}\text{Cost} &= [m_{R1} + \text{SORT}(F_{A2} \times H_{R1} \times m_{R1})] \\ &\quad + [IA(I_{B2},R_2) + b(m_{R2}, p_{R2}F_{B2} \times n_{R2}) + \text{SORT}(F_{B2} \times H_{R2} \times m_{R2})] \\ &= \text{Cost}(R_1,Cf_{21},\text{sort-merge}) + \text{Cost}(R_2,Cf_{12},\text{sort-merge})\end{aligned}$$

The case in which the index on $R_2.B_2$ is missing (rather than $R_1.A_2$) is treated similarly and will result in the same formula in the shorthand notation.

Cases 3-A and 4-A: The sort-merge method is used for R_1 and the join index method for R,, in the same situation as in Figure 1-2, except that the restriction index for the column $R_2.B_2$ is missing. In this case, the join is performed as in Case 3. The only difference is that, since indexes are now absent for the restriction columns of R,, the restriction predicate for R_2 can be resolved only after the tuples are retrieved. The cost of evaluating the query becomes

$$\begin{aligned}\text{Cost} &= [IA(I_{A2},R_1) + F_{A2} \times m_{R1} + \text{Sort}(F_{A2} \times H_{R1} \times m_{R1})] \\ &\quad + [IS(I_{A1},R_1) + b(m_{R2},p_{R2},Cf_{21} \times n_{R2})] \\ &= \text{Cost}(R_1,Cf_{21},\text{sort-merge}) + \text{Cost}(R_2,Cf_{12},\text{join-index})\end{aligned}$$

In the case in which $R_1.A_2$ is missing (rather than $R_2.B_2$), it will change the first part of the cost formula we obtained in Case 3, but will result in the same shorthand form. The case in which the join index method is used on R_1 and the sort-merge method on R,, as in Case 4, is treated similarly.

1.5.4 Formalization

In all the cost formulas so far, the coupling factors have been used in both directions, – i.e., both bracketed expressions in a formula were of the form $Cost(R_k, Cf_{jk}, type\text{-}of\text{-}join)$. We shall call the form of these formulas *symmetric.*

Join costs can be written in this form only when the coupling factors are known to be effective for the join method used (as when the join index method was used in the previous cases), or when the cost can be determined regardless of the coupling factors (as when the sort-merge method is used). The reason is that the only ambiguity in determining the cost of a join is whether or not the coupling factor will be included in the calculation – since all other information needed is local and is not affected by interaction with other relations. If we know at design phase that coupling factors are effective or that the cost is independent of the coupling factor, we can determine at design phase the costs of various possible joins on each relation and, using only local information and the coupling factors without ambiguity, accordingly determine the best join method and its cost. There are, however, some cases in which we cannot determine whether the coupling factors are effective at design phase. These will be introduced in Example 3.

If the best join method can be determined with only the local information (the access configuration of the relation and the type of join method used) and coupling factors, without any regard to other relations, the clear implication is that we can design an optimal access configuration of a relation by using only local information and the coupling factors, indcpcndcntly of the other relations. The design could be performed by the following procedure:

1. Consider each possible access configuration of a relation in turn
2. Find the best join method and its cost for the particular configuration
3. Repeat this procedure for other access configurations

4. Find the one that gives the minimum join **cost**

The only nonlocal information used here is furnished by coupling factors. Lumped within them are all the interactions from other relations. B havet we already observed that the coupling factors do not depend on access configurations of the other relations, nor do they depend on the join methods chosen; they depend exclusively on the properties of given queries and the data characteristics of the relation. Furthermore, these properties can be determined before we start designing any access configuration in the database.

We conclude here that we can design the access configuration of the entire database optimally by designing the optimal access configurations of individual relations one by one, regardless of the remaining relations- when all the information needed is known at design time. The local optimum configurations will collectively comprise the global optimum configuration.

To formalize the foregoing observation, we need the following definitions and theorems.

Definition 5: A ***partial-join cost*** is that part of the join cost that represents the accessing of only one relation, as well as the auxiliary access structures defined for that relation. □

In the examples above, each expression in square brackets represents a partial-join cost.

Definition *6:* A ***partial-join algorithm*** is a conceptual division of the algorithm of a join method whose processing cost is a partial- join cost. □

Definition 7: A join method is ***symmetric*** under certain constraints if, under these, both partial-join costs can be determined with only local information of the pertinent relation and the coupling factor, regardless of the partial-join algorithm used and the access configuration defined for the relation on the other side of the join. □

Definition 8: A set of join methods is ***separable*** under certain constraints, if under these constraints

- Any partial-join algorithm of a join in the set can be combined with any partial-join algorithm of any join method in the set, and

- Any combination of partial-join algorithms of the join methods in the set produces a symmetric join method. Cl

From the discussion at the beginning of Section 1.5.4, we have the following lemma.

Lemma 1: A join method is symmetric if and only if its cost has a symmetric form. Cl

Theorem 2: The problem of designing the optimal access configuration of a database can be decomposed into the tasks of designing the optimal access configurations of individual relations independently of one another, if the set of join methods used by the optimizer is ***separable*** with respect to the constraints imposed upon the database system.

Proof: Since the set of join algorithms used is separable, we can choose an arbitrary combination of partial-join algorithms within the set. Thus, we can choose any partial-join algorithm to be used for one relation without regard to the partial-join algorithm used for the other relation. Furthermore, since a join method consisting of any combination of partial-join algorithms is symmetric, the partial-join cost of a partial-join algorithm can be evaluated independently of the partial-join algorithm used and the access configuration defined on the other side of the join. As a result, the specific access methods assigned to and the partial-join algorithm used for one relation cannot affect any design parameters for the other relations. It is therefore guaranteed that there will be no interference among the designs of individual relations. Q.E.D.

Theorem 2 is a generalization of the observation made from Example 1, except that it now includes the coupling effects between relations.

Theorem 3: The set of join algorithms consisting of the join index method and the sort-merge method is ***separable*** under the constraint that every column in every relation in the database must have an index defined for it.

Proof: Part 1 of Definition 8 is obvious from previous examples and cases. When the join index method is used for both relations, all predicates are index-processible since every column has an index. Hence, all predicates are resolved with TIDs before the relations themselves are accessed; coupling factors are effective in both directions; and the cost formula has symmetric forms. When the sort-merge method is used for one

relation and the join index method for the other, then, by the same reasoning as in Case 3, the cost formula has symmetric forms. If only the sort-merge method is used, the cost formula is always symmetric. Therefore, from Lemma 1, the theorem holds. Q.E.D.

Only symmetric joins have been used in the example and cases presented so far. There are, however, instances of nonsymmetric joins.

Example 3: Let us assume that the join index method is used for both R_1 and R,, in the same situation as in Figure l-2, but that now restriction indexes for both R_1 and R_2 are missing. In this situation, since there are no restriction indexes, there is no way of resolving the restriction predicate without accessing the tuples themselves. Therefore, if we access relation R_1 first, the access cost would be

$$\begin{aligned} \text{Cost1} &= [IS(I_{A1},R_1) + J_1 \times n_{R1}] + [IS(I_{B1},R_2) + b(m_{R2},p_{R2},Cf_{12} \times n_{R2})] \\ &= \text{Cost}(R_1,J_1,\text{join-index}) + \text{Cost}(R_2,Cf_{12},\text{join-index}) \end{aligned}$$

On the other hand, if we access relation R_2 first, the access cost would then be

$$\begin{aligned} \text{cost2} &= [IS(I_{A1},R_1) + Cf_{21} \times n_{R1}] + [IS(I_{B1},R_2) + b(m_{R2},p_{R2},J_2 \times n_{R2})] \\ &= \text{Cost}(R_1,Cf_{21},\text{join-index}) + \text{Cost}(R_2,J_2,\text{join-index}) \end{aligned}$$

Therefore, we have two expressions each for the partial-join cost of each relation and we cannot determine at the design stage which of them is cheaper. Hence, this join method is not symmetric. The coupling factor is ineffective in one direction in each formula, since the join selectivity is used in its place. The cost formula is now also asymmetric relative to the coupling factors. □

We can still determine which of the two expressions is cheaper at query-processing time, but we do not have this knowledge when the physical database is being designed. If we want to ascertain the cheaper expression at design time, we have to analyze simultaneously the relation on the other side of the join- but this violates the definition of symmetry. The design of access configuration for one relation is no longer independent of the other relations. The theory presented in this paper depends entirely on the property of separability, which in turn depends on that of symmetry. The situation depicted in Example 3 is an apparent exception to our theory. However, in our discussion of the index selection problem in Section 1.6, the justification on the validity of our approach will be amply reinforced.

Theorem 4: The set of join methods consisting of the join index method and the sort-merge method is ***separable*** under the constraint that, whenever the join index method is used for both relations, at least one relation must have indexes for all restriction columns.

Proof: When both relations have indexes on all restriction columns, this theorem reduces to Theorem 3. As before when the sort-merge method is used for both relations, the cost formulas are always symmetric. When the join index method is used for one relation and the sort-merge method for the other, then, by a reasoning similar to Case 3-A, we obtain symmetric cost formulas. If only the join index method is used and one of the relations, say R,, has incomplete restriction indexes, the join is performed as in Case 1-A except that the restriction on R_1 is now partially resolved by using TIDs before accessing the tuples in R,. We thus get symmetric cost formulas. By Lemma 1, we prove this theorem. Q.E.D.

1.5.5 Update Cost

We assume here that the updates are performed only on individual relations, although the qualification part (WHERE clause) may involve more than one relation. Thus, updates are not performed on the join of two or more relations. Hence, if we segregate the qualification part (this will be treated as a query), the remaining part of an update transaction becomes separable (the update operation on one relation does not depend on the access configuration of the other relations). Note that we have assumed throughout that a block containing tuples of a relation contains only the tuples of that relation.

1.6 Design Algorithm

In this section, an algorithm for the design of optimal access configuration of the database will be presented.

1.6.1 Design Step 1

Based mainly on the result of Theorems 2 and 3, the first step of our algorithm is as follows:

Inputs:

- Usage information: A set of various types of queries and update transactions with their respective frequencies.

- Data characteristics (for every relation in the database): Size, blocking factor, selectivities of all columns, relationships with other relations with respect to join paths, join selectivity with respect to join paths.

outputs:

- Optimal position of the clustering column for each relation.
- Optimal combination of partial-joins for each type of two-variable query.

Condition Assumed:

- Every column of each relation in the database has an index defined for it. Some of these indexes will be dropped in the subsequent index selection step.

Algorithm 1:

1. Segregate the usage information in such a way that, if there is a subquery involving more than one relation in the qualification part of an update transaction, it is separated and its frequency is included with that of the same type'of query. Thereupon, all the remaining parts of the update transactions will refer to only one relation.

2. Calculate the coupling factors with respect to individual two-variable queries for every relation in the database using the given data characteristics.

3. Pick one relation and determine the optimal position of the clustering column as follows:

   a. Assign the clustering property to one column of the relation.

   b. Given that position of the clustering column, identify the best partial-join algorithm and calculate its partial-join cost for every two-variable query that refers to this relation, using the given data characteristics and the coupling factors.

   c. Utilizing the usage information and the result of Step b, calculate the total cost associated with this relation. This is done by summing up all the partial-join costs identified in Step b-multiplied by their respective frequencies- and all costs incurred by one-variable queries and update transactions acting upon this relation.

   d. Shift the clustering property to another column of the relation and repeat Steps b and c.

   e. Repeat Step d until all the columns of the relation have been considered. (The case in which there is no clustering column is also considered. Then determine the one that gives the minimal cost as the clustering column (or none).

4. Step 3 is repeated for every relation in the database. The aggregate of results for all relations comprises the global optimum.

A join path can often have a multiple column as the join column on either relation. In such cases, we consider the multiple join column as a single effective column, independent of its component columns. Therefore, according to the condition in the above algorithm, this effective column is considered to have a multiple-column index defined for it (we do not consider here additional problems involved in the multiple-column indexes).

Although in some cases improvement can be obtained by an adjustment in ordering among the effective column's component columns and by the deletion of overlapping indexes, this is not being considered here. It will be noted that, under the assumptions given, the Design Step 1 algorithm yields a mathematically true optimum.

1.6.2 Design Step 2: Index Selection

In the algorithm for Design Step 1, we imposed the restriction that every column of the relations in the database must have an index defined for it. However, not every index is beneficial. Some indexes can increase the total access cost because of their own access and update costs.

The index selection problem has been extensively studied by [KIN 743 [SCH 75] [HAM 76]. It concerns the method of selecting a set of indexes that will minimize the processing cost in a single-relation environment. Here we are using a slightly modified version of the approach introduced in [HAM 76]. The main modification involves translating the frequency of a partial-join into an equivalent frequency of a one-variable query (single-relation restriction). The following example should clarify the procedure:

Example 4: Let us consider the same query and situation as in Figure l-2, except that now both indexes for $R_2.B_1$ and $R_2.B_2$ are nonclustering. When we use the join index method for both relations, the partial- join cost of the partial-join for relation R_2 becomes

$$\text{Cost} = [IA(I_{B2},R_2) + IS(I_{B1},R_2) + Cf_{12} \times F_{B2} \times n_{R2}] \quad (1.9)$$

However, if we refine our assumption so that tuplcs having the same join column value are now accessed in TID order (we have ignored this fact so far for the sake of simplicity), the cost formula becomes

$$\text{Cost} = [IA(I_{B2},R_2) + IS(I_{B1},R_2) + (Cf_{12}/F_{B1}) \times b(m_{R2}, p_{R2}, F_{B1} \times F_{B2} \times n_{R2})] \quad (1.10)$$

Here (Cf_{12}/F_{B1}) is the number of distinct join column values selected by the coupling factor and the function b represents the cost of accessing the data tuples that have specific join column values and satisfy the restriction predicate. We used the b function because those tuples that have the same join column values are accessed in TID order. If tuples with the same join column values were accessed randomly, we would obtain Equation (1.9). Although Equation (1.10) is only a small refinement over Equation (1.9), it makes it easy to observe that the data tuple access cost of a partial-join can be regarded as the joint restriction cost of the join index and the restriction index, multiplied by the factor (Cf_{12}/F_{B1}). We call this factor ***the equivalent restriction frequency*** of a partial-join. Let us note that the function b in the above formula yields exactly the same cost as would the joint restriction of two indexes.

More importantly, it can be shown that the gain in access cost by having the restriction index in a partial-join – assuming the join index-is always present- is equal to the gain in access cost that the same restriction index would yield in the joint restriction, multiplied by the equivalent restriction frequency. The same observation holds with certain limitations when one of the indexes is clustering. A more detailed treatment can be found in Appendix B. □

Definition 9: The ***equivalent restriction frequency*** of a partial- join is defined as the ratio of the gain in access cost by having the restriction indexes in a partial-join to the gain in access cost that the same restriction indexes would yield in the joint restriction with the join index if the join index is used in the partial-join (i.e, if the join index method is used), or in the restriction of the restriction indexes alone if the join index is not used (i.e., if the sort-merge method is used). □

According to this definition, the equivalent restriction frequency of a partial-join using the sort-merge method is 1, if the restriction indexes are used to access the relation initially before sorting, and 0 otherwise.

Since the preceding discussion is not concerned with the index-accessing cost, we use the equivalent restriction frequency only to estimate the ranking of indexes in importance, as will be explained later. We shall utilize partial-join cost formulas in our actual cost calculation.

Following is the algorithm for Design Step 2. This algorithm is mainly based on the above discussion, and on Theorems 2 and 4.

Inputs:

- Outputs from Design Step 1: optimal position of the clustering column for each relation and optimal combination of partial-joins for each type of two- variable query.
- Set of types of one-variable queries and update transactions of interest with their respective frequencies. Here each type of one-variable query represents any Boolean combination of simple predicates. A simple predicate is one that refers to only one column of the relation.
- Data characteristics similar to the ones used in Design Step 1, but only those parameters that pertain to single relations are relevant.

outputs:

- Set of indexes of each relation that gives the minimum processing time.

Algorithm 2:

1. Select one relation
2. From the information outputted in Design Step 1, calculate the equivalent restriction frequency of each partial-join involving this relation.
3. From the usage information for one-variable queries and the equivalent restriction frequencies calculated above, compute the total frequency f of references to each column.
4. Rank the importance of the columns, using f X m X (1 − F), where m is the total number of blocks of the relation and F is the selectivity of each column. The above formula represents an upper bound on the number of block accesses saved by the restriction index, in the sense that it represents the number of block accesses saved if there is no other index and all the selected tuples are clustered [HAM 76].
5. If a join index has ever been used in Design Step 1- that is, if at least one partial-join uses the join index method- then assign an index to that column by default. This is a heuristic WC use to avoid strong intcrfercncc between Design Step 1 and Design Step 2 (We assume that column domains are rigorously defined, and that joins are limited to semantically appropriate columns [WIE 79]). If the join index used in Design Step 1 were dropped in Design Step 2, we would have to switch all the partial-joins that used this now nonexistent join index to the sort-merge method. The result would be to distort the entire cost calculation that was performed in Design Step 1.
6. Select indexes incrementlly one by one, ordered by rank. Include only those indexes that reduce the total cost. If, during the cost calculation, a query type represents a partial-join rather than a one-variable query, the partial-join cost is used instead of the joint restriction cost.

1.6.3 Separability in Design Step 2

The implicit meaning of the index selection is that those indexes that do not compensate for their own maintenance and access cost should be dropped. In Design Step 2 we again considered relations singly and independently of one another. This was based on the separability theory of Theorem 2, i.e., that the access structures assigned to one relation do not affect cost calculations for other relations. However, since, in contrast to Design Step 1, we are eliminating some indexes, we can encounter situations that were excluded as exceptions in Example 3 and Theorem 4. In these situations, calculation of cost is no longer separable. Nevertheless, it turns out the calculative error caused by the assumption of separability, even in these exceptional situations, is not significant.

If we look at Example 3 again, the actual cost at query-processing time will be

$$\begin{aligned}\text{Cost} &= \min(\text{Cost1}, \text{Cost2}) \\ &= \min[\{\text{Cost}(R_1, J_1, \text{join-index}) + \text{Cost}(R_2, Cf_{12}, \text{join-index})\}, \\ &\qquad \{\text{Cost}(R_1, Cf_{21}, \text{join-index}) + \text{Cost}(R_2, J_2, \text{join-index})\}]\end{aligned}$$

But, because we assumed symmetry, the sum of the costs we used implicitly in Design Step 2 is

$$\text{Cost'} = \text{Cost}(R_1, Cf_{12}, \text{join-index}) + \text{Cost}(R_2, Cf_{21}, \text{join-index})$$

Thus, the total error in cost estimation will be

$$\begin{aligned}\text{Error} &= g \times (\text{Cost} - \text{Cost'}) \\ &= g \times \min[\{\text{Cost}(R_1, J_1, \text{join-index}) - \text{Cost}(R_1, Cf_{21}, \text{join-index})\} \\ &\qquad \{\text{Cost}(R_2, J_2, \text{join-index}) - \text{Cost}(R_2, Cf_{12}, \text{join-index})\}]\end{aligned} \tag{1.11}$$

, where g is the frequency of this join.

-Remember, however, that the restriction indexes for both relations had been dropped because their benefits did not compensate for their update and access cost. Hence, it must be either that the frequency of access to the column is not significant, or that the effect of selectivity is small. Therefore, either the frequency of the join we are concerned with is insignificant or the coupling factor approaches the join selectivity -making the error insignificant (see Equation (1.11)). Following this argument, we claim that separability can be applied to all the cases of concern without causing any significant error. Similar situations arise when, on both relations, only some of the restriction columns specified in a query have indexes assigned, while others do not. A similar argument holds for such cases.

The costs of some partial-joins that USC the join index method may be changed as a result of the removal of some indexes in Design Step 2. This could make the sort-merge method more feasible. However, according to an argument analogous to the one above, we claim that the total of errors incurred in such situations could not be significant; otherwise the index would not have been dropped.

1.7 Extensions and Further Study

An extension of nonseparable joins, for instance, the inner/outer-loop join method and the multiple-pass method described in Section 1.3, could be made by means of the following heuristic method. After Design Step 1, each type of two-variable query is considered in turn and its join cost, as determined in Design Step 1, is compared with possible nonseparable joins. If a nonseparable join is cheaper, that query type should be marked to note that this nonseparable join must be used. For a possible shift of the clustering column, after completion of this step, Design Step 1 should be repeated – with the join method for a marked query type fixed to be the nonseparable join method assigned previously. This whole procedure (Design Step 1 and the refinement step with nonseparable join methods) is repeated until the refinement becomes insignifant.

The link structure [BLA 76] can be considered next. For every join path, the total cost of all queries using this join path is compared with the cost based on a hypothetical link. If the latter is less, a link is assigned to that join path. If the join column on the N-side relation of the l-to-N relationship is a clustering column, the link is endowed with the clustering property -otherwise not.

The most attractive prospects for the inner/outer-loop join methods are those queries that use the sort-merge method for the relation on the l-side of the l-to-N relationship, but use the join index method for the other side. Use of the inner/outer loop join method in these cases has the advantage of saving sorting time on one relation and index-searching time on the other (if it has a strong coupling factor). On the other hand, join paths that support many queries using the inner/outer-loop join method would be the most promising prospects for the link structure. Index selection could be done at the conclusion of these steps. By reinforcing the foregoing approach with improved arguments and effective heuristics, we look forward to extending our basic theory to queries of more than two variables.

Finally, although we have developed our theory in terms of the relational system, it should be pointed out that the basic concept of separability is applicable to network database systems as well (Theorem 2 holds for any system, while Theorems 3 and 4 are relevant only for relational systems).

1.8 Conclusion

It has been observed and proved that, with a separable set of join methods, the problem of designing the optimal physical database can be reduced to one of designing optimal individual relations. This can be done independently of one another by using the coupling factors that represent all interactions among the relations. This substantially diminishes the complexity of the problem by partitioning it into disjoint subproblems. The task is made even more manageable by dividing the procedure into two steps-one for deterrninipg the optimal positions of clustering columns, the other for index selection. A proper interface between the two steps was introduced.

Design Step 1 results in a true mathematical optimum. Although, because of the heuristics used in Design Step 2 and for the interface between the two steps, the overall design does not provide a true optimum, it was argued that the deviation would be insignificant.

The key objective of this paper is to propose a formal approach to the design of physical databases that simplifies the problem considerably and, at the same time, provides better insight into underlying mechanisms. We believe that this novel approach can enable substantial progress to be made in the optimal design of multifile physical databases.

2. Estimating Block Accesses in Database Organizations – A Closed Noniterative Formula

2.1 Int roduction

In evaluating the access cost of a query for a database organization in which records are grouped into. blocks in secondary storage [WIE 77], one must often estimate the number of block accesses required to retrieve the records selected by the query. Various formulas have been proposed for this purpose [CAR 75] [ROT 74] [SEV 72] [SIL 76] [WAT 72] [WAT 75] [WAT 76] [YAO 77] [YUE 75]. In particular, Yao [YAO 77] presented the following theorem:[7]

Theorem 1: [Yao] Let n records be grouped into m blocks $(1 \leq m \leq n)$, each containing p = n/m records. If k records are randomly selected from the n records, the expected number of blocks hit (blocks with at least one record selected) is given by

$$b(m,p,k) = m\,[1 - \binom{n-p}{k} / \binom{n}{k}] \tag{2.1}$$

$$= m\,[1 - ((n-p)!(n-k)!)/((n-p-k)!n!)] \tag{2.2}$$

$$= m\,[1 - \Pi_{i=1}^{k}(n-p-i+1)/(n-i+1)] \tag{2.3}$$

when $k \leq n-p$, and

$$b(m,p,k) = m \quad \text{when } k > n-p. \tag{2.4}$$

Earlier Cardenas [CAR 75] suggested the formula

$$bc(m,p,k) = m\,[1 - (1 - 1/m)^k], \tag{2.5}$$

assuming that there are n records divided into m blocks and that the k records are randomly selected from the n records. It is interesting to note that Eq. (2.5) is independent of the blocking factor p.

Yao [YAO 773 showed that Eq. (2.5) is based on the assumption that records are selected ***with replacement,*** i.e., a record can be selected more than once. But this assumption does not hold in practice, since records selected by a query simultaneously must be distinct from one another. Yao eliminated this assumption and proved Theorem 1 under the assumption that records are actually selected ***without replacement,*** i.e., a record cannot be selected more than once at one time.

---

[7] The notation and some of the conditions have been slightly modified

Theorem 1 gives the exact formula under the given assumptions. However, we notice that Eq. (2.3) has an iterative form, which will take excessive time to evaluate if k becomes large. Another way of evaluating Yao's formula is by using the Gamma function (in practice a Log Gamma (LGAM) [IBM 70] function should be used, since the Gamma function grows very steeply). By modifying Eq. (2.2) slightly, we obtain

$$b(m,p,k) = m\,[1 - \exp(LGAM(n-p) + LGAM(n-k) - LGAM(n-p-k) - LGAM(n))]. \quad (2.6)$$

Evaluation of this formula poses a problem in practice, especially when k is small. Since, in the evaluation of the argument of the exponential function, we are subtracting big numbers from equally big numbers to get a very small number, the roundoff error of the computation can become intolerable. For example, when Eq. (2.6) is calculated by using single-precision variables on a 36-bit machine having the resolution of 2^{-27} ($\simeq 10^{-8}$) [DEC 78], it has a 46% error at $p = 10$, m = 1000, n = 10000, and k = 2. The roundoff error is 310% when p = 10, m = 3162, n = 31620, and k= 3. But these values of parameters are well within the range of relevant databases.

We propose below a closed noniterative formula that approximates Yao's exact formula with reasonable accuracy, as well as reducing considerably the computation error caused by limited precision.

2.2 A Noniterative Formula

In this section, we introduce the following formula and discuss how it was obtained. Errors of this formula will be discussed in Section 2.3. We assume throughout that m and k have only integer values.

$$\begin{aligned} b_{w1}(m,p,k)/m &= [1 - (1-1/m)^k] \\ &\quad + [1/m^2 p \times k(k-1)/2 \times (1-1/m)^{k-1}] \\ &\quad + [1.5/m^3 p^4 \times k(k-1)(2k-1)/6 \times (1-1/m)^{k-1}] \\ &\qquad\qquad \text{when } k \le n-p, \text{ and} \end{aligned} \quad (2.7)$$

$$b_{w1}(m,p,k)/m = 1 \qquad \text{when } k > n-p \quad (2.8)$$

Let us see how Eq. (2.7) has been derived. When k > n-p, we always have $b_{w1}(m,p,k)/m = 1$ from Eq. (2.4). If we use n = mp, Eq. (2.3) can be transformed to an equivalent form

$$b(m,p,k)/m = 1 - \Pi_{i=0}^{k-1}(1 - 1/m(1 - i/mp)) \quad (2.9)$$

If we perform a series expansion on $1/m(1 - i/mp)$ and take only the first three terms, we obtain

$$b(m,p,k)/m \simeq 1 - \Pi_{i=0}^{k-1}((1 - 1/m) - i/m^2 p - i^2/m^3 p^2)$$

If we expand the multiplication and keep the first three terms, we get

$$b(m,p,k)/m \simeq [1-(1-1/m)^k] + [1/m^2p \text{ X } k(k-1)/2 \text{ X } (1-1/m)^{k-1}] + [1/m^3p^2 \text{ X } k(k-1)(2k-1)/6 \text{ X } (1-1/m)^{k-1}]. \quad (2.10)$$

Eq. (2.10) is only an approximation of Eq. (2.9), since we took only a few terms from the expansions. Two factors were added to Eq. (2.10) to derive Eq. (2.7). The factor 1.5 has been introduced empirically to compensate for the errors at small values of p, i.e., $p \simeq 1$. It was chosen especially to reduce the error to zero when p = 1, $k = \lfloor n-p \rfloor$, as n goes to infinity ($n \to \infty$), in which case Eq. (2.10) has the most significant error. The factor $1/p^2$ has been introduced empirically to reduce the effect of the third term for higher values of p, for adding the third term at these values of p increases the error (although it reduces the error at lower values of p). We shall show later that the approximation formula derived here constitutes a practically negligible deviation from the exact formula.

2.3 Error Analysis

We note that the first term of Eq. (2.7) is identical to Cardenas' formula, Eq. (2.5). The second term compensates for the major error of Eq. (2.5), while the third term provides a finer adjustment to further reduce the error. The third term has been empirically modified to get a better approximation.

Derived in Theorem 2 and plotted in Figure 2-l for various values of p and $\kappa = k/n$ is a formula that gives the limiting values of the error $ERR(m,p,k) = (b(m,p,k) - b_{w1}(m,p,k))/b(m,p,k)$ as the total number of blocks m (and, accordingly, the total number of records n) goes to infinity.

Theorem 2:

$$\lim_{m \to \infty} ERR(m,p,k) = 1-(1-e^{-p\kappa}(1-p\kappa^2/2-\kappa^3/2p))/(1-(1-\kappa)^p), \quad (2.11)$$

where $ERR(m,p,k) = (b(m,p.k) - b_{w1}(m,p,k))/b(m,p,k)$, and p and κ have fixed values.

Proof: To derive this formula, we need the following form of Yao's formula, which has the iteration on the blocking factor p rather than on the number of selected records k.

$$b(m,p,k) = 1 - \Pi_{i=1}^{p}((n-k-i+1)/(n-i+1)) \quad (2.12)$$

This formula is easily derivable from Eq. (2.2). If we subtract Eq. (2.7) from Eq. (2.12) and divide the result by Eq. (2.12), we can obtain Eq. (2.11) by taking the limit as $m \to \infty$ (accordingly $n \to \infty$) and by using the identity $\lim_{m \to \infty}(1-1/m)^m = e^{-1}$. Q.E.D.

Eq. (2.12) is also a convenient formula for evaluating the exact value when we have integer blocking factors. In fact, all computed values for integer blocking factors that we shall employ later in this section were produced by using Eq. (2.12). The limiting values, as the blocking factor p goes to infinity with m and κ fixed, are proved to be zero in the following theorem.

Theorem 3: $\lim_{p\to\infty} ERR(m,p,k) = 0$, where m and κ = k/n have fixed values. Here $0 \le k \le (m-1)p$, and k is an integer.

Proof: If κ =0, both $b(m,p,k)/m$ and $b_{w1}(m,p,k)/m$ simply become 1, so ERR(m,p,k) must be zero. If $\kappa > 0$, we know that $\lim_{p\to\infty} b(m,p,k) \ge 1$, since at least one block must be hit. Therefore, the denominator of ERR(m,p,k) is always at least 1 and cannot be 0. To study the behavior of the numerator, let us look at Eq. (2.12). In Eq. (2.12), $(n-k-i+1)/(n-i+1) \le (n-k)/n = 1-\kappa < 1$. Therefore, $\lim_{p\to\infty} \Pi_{i=1}^{p}((n-k-i+1)/(n-i+1)) \le \lim_{p\to\infty}(1-\kappa)^p = 0$. Thus, $\lim_{p\to\infty} b(m,p,k) = 1$. But it is clear from Eq. (2.7) that $\lim_{p\to\infty} b_{w1}(m,p,k) = 1$ also, since $\lim_{p\to\infty}(1-1/m)^{\kappa mp-1} = \lim_{p\to\infty}(1-1/m)^{\kappa mp} = \lim_{p\to\infty} e^{-\kappa p} = 0$, and an exponential order can suppress any polynomial order of p. Hence, $\lim_{p\to\infty} ERR(m,p,k) = 0$. Q.E.D.

The errors that occur when both n and p are finite were investigated by performing an exhaustive computer calculation. These analyses show that Eq. (2.7) yields at most 3.7%(-3.7% if the sign is considered) of deviation from the exact formula, Eq. (2.3), over the entire range of $p \ge 1, m \ge 1, 0 \le k \le n-p$, where m and k are integers. This maximum error occurs at p = 1 + $\sqrt{2}$, k = $\lfloor n-p \rfloor$ as $m \to \infty$ (This can be observed in Figure 2-1. In fact the maximum error and the value of p at which this error occurs can also be derived from Eq. (2.11), once we know that this occurs at κ = 1, as $m \to \infty$.) The maximum positive error (2.5%) occurs at p = 1.5, k = 3, and m = 3. The maximum positive error when $m \to \infty$ is 2.1% at p = 1.7 and k = 0.65n.

The dependence of the error on the values of n/p = m is shown in Figure 2-2, where k is set to be equal to $\lfloor n-p \rfloor$ (note that the maximum error occurred at this k value). At low values of m and p there is a short range within which errors are changing by a large amounts, since at these values of m and p, $k = \lfloor n-p \rfloor = \lfloor (m-1)p \rfloor$ is in the range where high positive errors occur, as we see in Figure 2-1 (see the value when p = 2, m = 3, n = 6, and k = 4, for example). The dependence of the error on m is otherwise very flat, as

in Figure 2-3, which shows the values when $\kappa = 0.65$ with corresponding k values rounded to the nearest integers. In Figure 2-3 the values at m=1 and m = 2 are 0 from Eq. (2.4) and Eq. (2.8), since at these points k = 0.65n > n-p.

The values of variables we used in the exhaustive computer calculation are as follows, with the constraint that mp $\leq 10^7$ (10^6 for noninteger blocking factors):

- m: 1, 2, 3, 10, 32, 100, 316, 1000, 3162, 10000, 31623, 100000, 316228, 1000000
- p: 1, 2, 3, 4, 5, 10, 32, 100,316, 1000, 3162; 1.1, 1.2, ... 1.9; 2.1, 2.2, ... 2.9
- k/n: 0.0, 0.02, 0.05, 0.1, 0.15, 0.2, ... 1.0
- k: $\lfloor n - p \rfloor$, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 32,100

2.4 Computational Error due to Limited Precision

The major computational error is due to the evaluation of (1 − l/m) in Eq. (2.7). For example, if m = 10^6, we need better resolution than 10^{-6}. However, it is shown in [WHA 81] that the number of valid digits required by Eq. (2.7) is roughly proportional to log&n), while that required by Eq. (2.6) using the Gamma function is proportional to $\log_{10}(mn \ln(n))$ for the same precision in the result. In the exhaustive calculation using a DEC System 20 with single-precision variables, we obtained a maximum error of 0.2% when m= 10^6 over the range of variables shown in Section 2.3.

2.5 Comments on Related Work

Formulas essentially identical to Cardenas' and Yao's formulas were derived independently by Waters and Karayiannis [WAT 72][WAT 75][WAT 76]. Waters summarized three related formulas in [WAT 76], which are

$$b_{WAT1}(m,p,k) = m\,[1 - (1 - k/n)^p], \tag{2.13}$$

$$b_{WAT2}(m,p,k) = m\,[1 - (1 - p/n)^k], \text{ and} \tag{2.14}$$

$$b_{WAT3}(m,p,k) = m\,[1 - \Pi_{i=1}^{k}(1 - p/(n-i+1))]. \tag{2.15}$$

Eq. (2.14) and Eq. (2.15) are identical to Eq. (2.5) and Eq. (2.3), respectively. Eq. (2.13) was derived in [WAT 72][WAT 75], as follows:

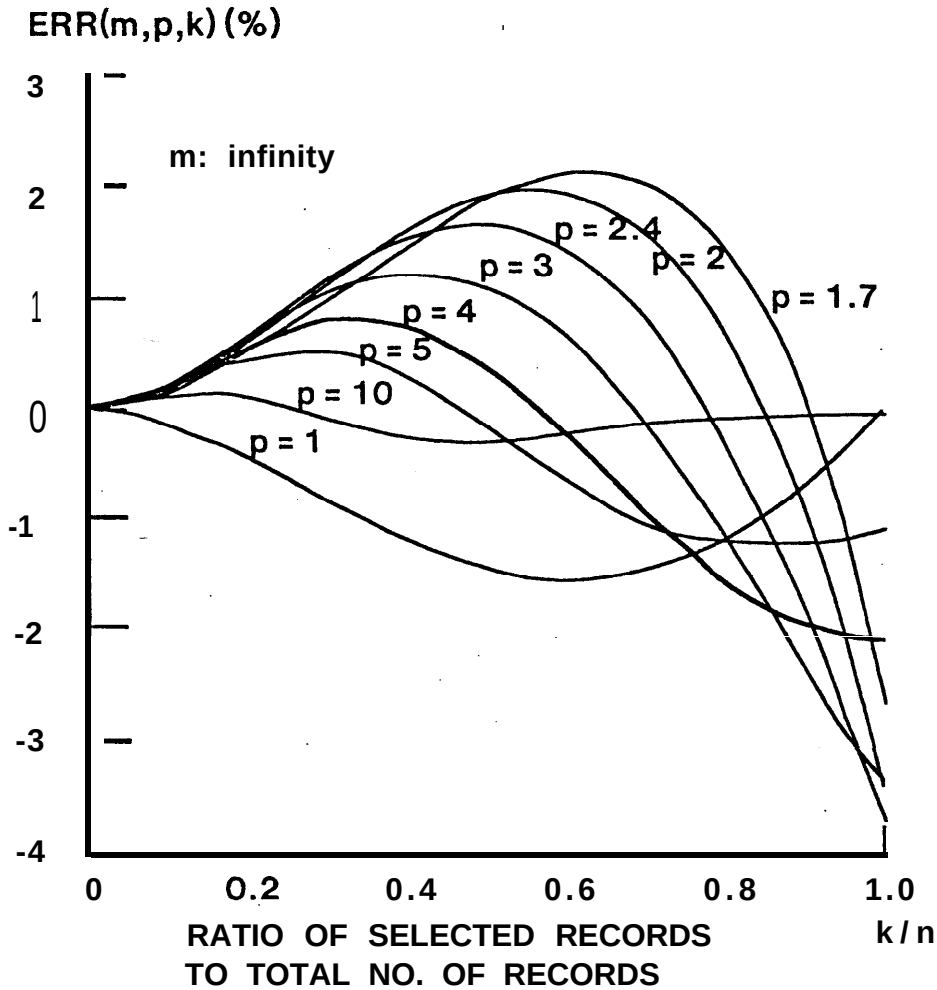

Figure 2-1: Er'ror of Eq. (2.7) as m Goes to Infinity.

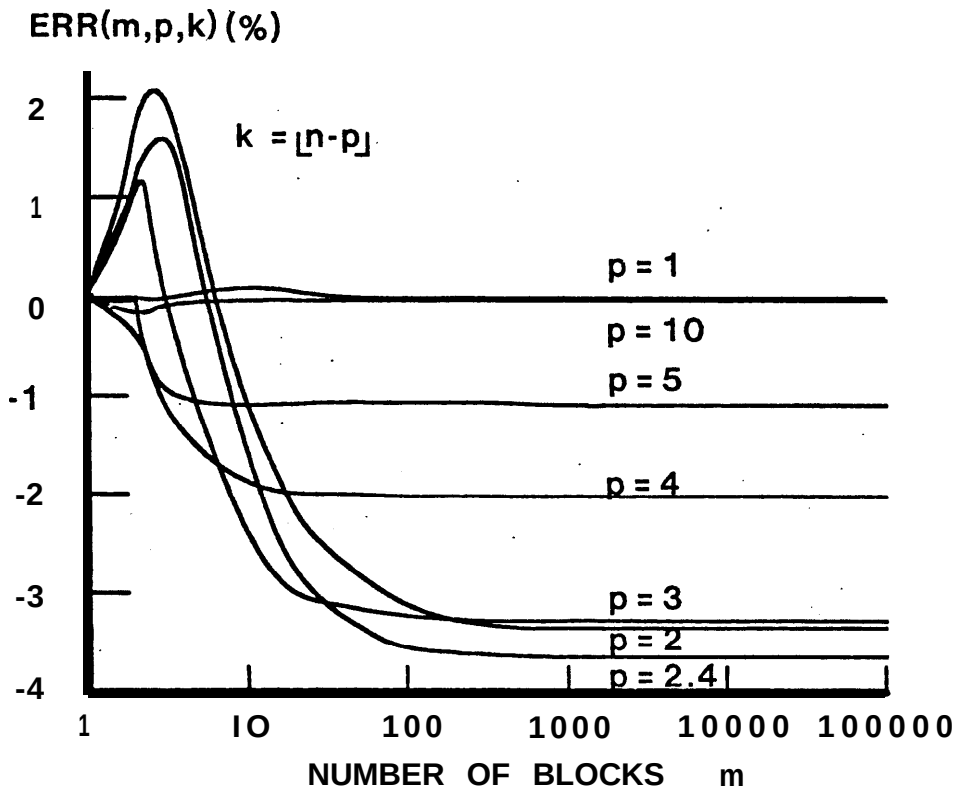

Figure 2-2: Error of Eq. (2.7) when k = ⌊n-p⌋

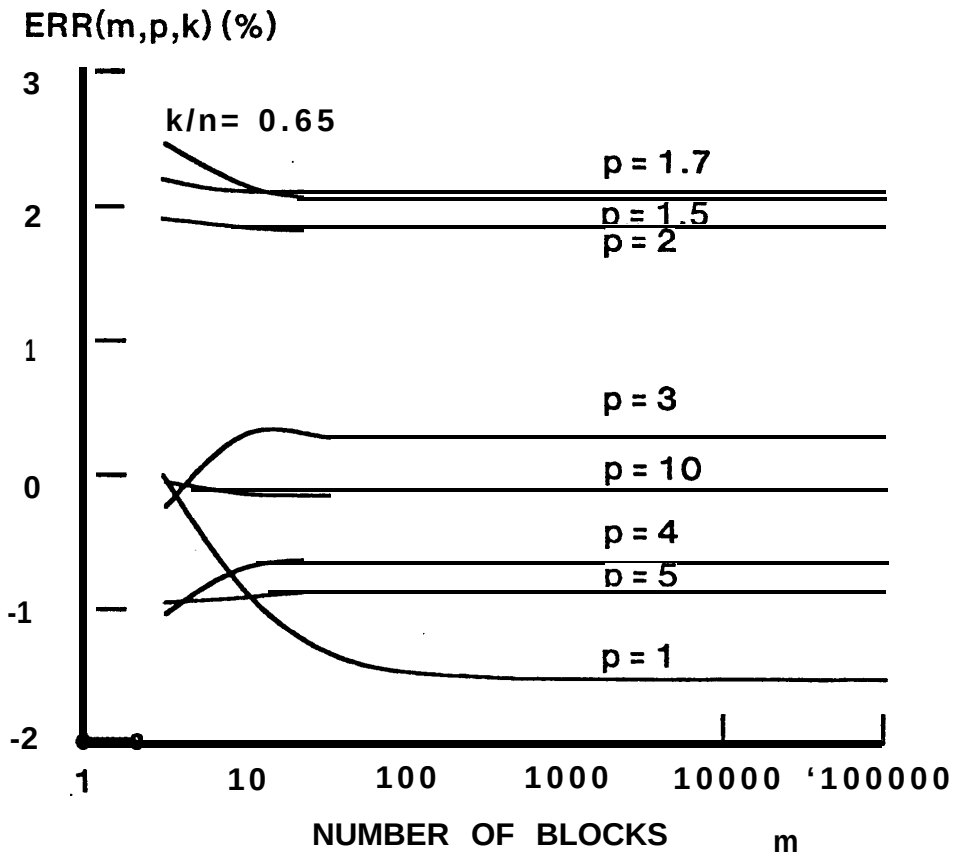

Figure 2-3: Error of Eq. (2.7) when k = 0.65n

RHR = number of distinct records hit / total number of records in the file
= probability that any particular record is hit
= k/n.

∴ 1- RHR = probability that any particular record is not hit.
∴ $(1 - RHR)^p$ = probability that any particular block is not hit.
∴ 1- $(1 - RHR)^p$ = probability that any particular block is hit.

Subsequently, during one of Waters' lectures, Karayiannis (then a student) suggested that Eq. (2.13) was incorrect, pointing out that Eq. (2.13) gives an incorrect result where m = 1 (correct result is b(m,p,k)= 1 if k > 0). He further suggested Eq. (2.14) as an alternative formula. Later Waters [WAT 76] announced that Eq. (2.13) and the above derivation were incorrect and instead suggested Eq. (2.15) as an alternative formula.

However, we note that the derivation of Eq. (2.13) is correct if we make the ***independence*** assumption in calculating the probability that any particular block will not be hit. More rigorous derivation should use *conditional* ***probability,*** since the ***events*** of each record's being hit are not mutually probabilistically independent.

We note that, if we interchange p and k, Eq. (2.12) bears the same relationship with Eq. (2.13) as Eq. (2.3) does with Eq. (2.14). In this sense, Eq. (2.12) and Eq. (2.13) are a *dual* of Eq. (2.3) and Eq. (2.14).

It was observed in [YAO 77] that Eq. (2.14) yields a good approximation when k << n (κ << 1) or p >> 1. Hence, Eq. (2.13) will give a good approximation when p << n (m >> 1) or k >> 1 by duality. This means that one formula will result in a good approximation when its counterpart yields a poor one, and vice versa. Therefore, an obvious alternative approach to the one presented in this paper is to combine these two formulas in such a way as to get a good approximation over the entire range. As an example, we suggest here the following formula:

$$b_{W2}(m,p,k) = \max\{b_{WAT1}(m,p,k), b_{WAT2}(m,p,k)\} \quad (2.16)$$
$$= \max\{m[1 - (1- k/n)^p], m(1 - (1- p/n)^k]\},$$

where 'max' represents the minimum of the two arguments. This equation will be a good approximation, since either formula always produces a value smaller than the exact formula. (This can be easily understood by examining the underlying assumptions.)

2.6 Application

An implicit assumption'made throughout the development of all the formulas is that a block is accessed no more than once. We encounter this situation in practice when the records selected are accessed in TID (tuple identifier or database key) order.

Two typical applications of these formulas are in query optimization [YAO 79] and, physical-database design [HAM 76] [WI-IA 81]. The formulas are used to estimate the number of block accesses, which is an important measure of cost. In an approach employed in [WHA 81], they are also used to estimate the number of logical groups of records selected. A *logical* ***group*** is a set of records grouped according to certain criteria- for example, common possession of the same value on a certain field. Close estimation of the number of logical groups selected is necessary in analyzing the interactions among relations in the design of a physical database. In this application, we are very likely to have low grouping factors (number of records in a group) that correspond to the blocking factors of a block (physical group). For example, we have a grouping factor of 1 when the records are grouped according to the values of a key field.

Although Cardenas' formula (currently used in System R [SCH 81]) gives a reasonable approximation in many cases, it is expccially prone to failure at low blocking factors (particularly when $p < 10$). Eq. (2.7) proves to be very useful in these situations.

2.7 Conclusion

A closed noniterative formula for estimating the number of block accesses was introduced. It improves Yao's exact formula in the sense that it significantly reduces the computation time by eliminating the iterative loop, while providing a practically negligible deviation (maximum error = 3.7%) from the exact formula over the entire range of variables involved. The computational error due to the machine's limited precision has been greatly reduced as compared with a method using the Gamma function based on Yao's formula. It significantly improves Cardenas' earlier formula, which has a maximum error of e^{-1} = 36.8% (at p = 1).

Appendix A. Relationships between Relations

In this section, we demonstrate that the assumption that we made in Section 1.2 excluding M-to-N relationships from consideration for optimization is reasonable.

Relations can have various relationships (not necessarily semantically meaningful ones) depending on the characteristics of the domains of the attributes that are related. For example, if we relate a key attribute (or set of attributes) in relation R_1 and a nonkey attribute (or set of attributes) in relation R_2, then R_1 and R_2 have a l-to-N relationship with respect to these attributes considered. Relations R_1 and R_2 will have a l-to-l relationship if attributes considered in both relations are key attributes, and an M-to-N relationship if both are nonkey attributes.

In this section, we shall show that a relation scheme any of whose relation instance is a join of two relations which has an M-to-N relationship with respect to a set of attributes A has a multivalued dependency (MVD) – assuming that the only predicate that relates these two relations is the one that represents the join on A.

Intuitively, if a relation scheme R has an MVD $A \rightarrow\rightarrow B$ (and accordingly $A \rightarrow\rightarrow R - B$), where A and B are sets of attributes in R, then in a specific relation instance r of R, given a specific value of A, the values of R- B are completely replicated for every distinct value of B. Because of this replication, sets of attributes B and R – B do not bear much meaningful relationship, and thus it does not make much sense to have both sets of attributes together in a single relation.

. We believe, in accordance with the above argument, that joining two relations that have M-to-N relationships with respect to the set of attributes on which the join is performed is relatively infrequent. In Section 1.2, on the basis of this argument, we excluded from consideration as prospects for optimization join operations on relations that bear an M-to-N relationship.

We have the following theorems:

Theorem 1: If a relation scheme R has an MVD $A \rightarrow\rightarrow B$, where A and B are sets of attributes of R, then every relation r for R is a natural join of projections of r on the relation schemes R_1 = A, R_2 = AUB, R_3 = AU(R − B), respectively, where R,, R,, and R_3 possess the relationships shown in Figure A-l.

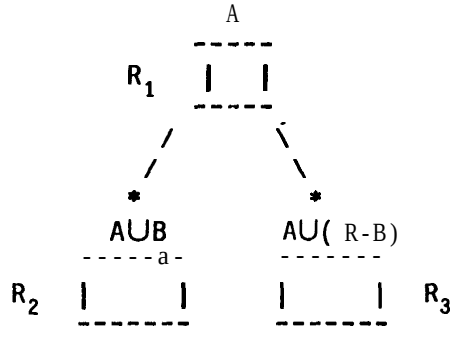

Figure A-l: Relation Schemes and Their Relationships.

In this figure – – * represents a l-to-N relationship with respect to **A.**

Proof: R,, R,, and R_3 can be obtained by two consecutive lossless join decompositions, i.e., decomposition of R into AUB and AU(R-B) and decomposition of AUB into A and AUB. These two decompositions are lossless, since we have an MVD $A \rightarrow\rightarrow B$ [ULL 80]. Thus, the overall join decomposition of R into R,, R,, and R_3 is also lossless. Therefore, for any relation r for R, $r = JOIN_{i=1}^{3} \Pi_{R_i}(r)$.

To prove that R_1 and R_2 has a l-to-N relationship, we note that A in R_1 is a key, since it is the only attribute (or set of attributes) in R,. However, A in R_2 is generally not a key. So we have a l-to-N relationship from R_1 to R_2

When A in R_2 is a key, we have a l-to-l relationship between R_1 and R_2, which can be considered as a special case of a l-to-N relationship. Similarly, R_1 and R_3 have a l-to-N relationship. Q.E.D.

Theorem 2: A relation scheme R has MVDs $A \rightarrow\rightarrow B$ and $C \rightarrow\rightarrow D$ if any relation r for R is a natural join of some relations r_1, r_2, and r_3 for relation schemes R,, R,, and R,, respectively, where R,, R,, and R_3 have the relationships shown in Figure A-2.

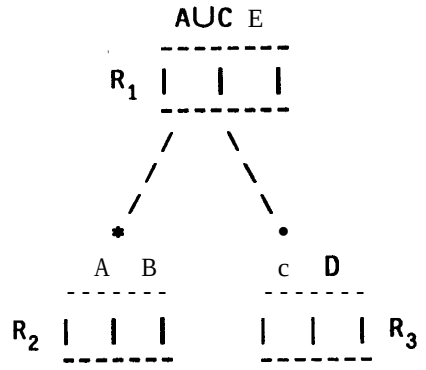

Figure A-2: Relation schemes and their relationships.

In that figure — — * represents a l-to-N relationship with respect to A on the left side and one with respect to C on the right side.

Proof: Consider tuples t and s with $t[A] = s[A]$ in a relation r for R. Since r is a natural join of some relations r_1, r_2, and r_3, respectively, there must exist tuples u_1, u_2 in r_1; v_1, v_2 in r_2; and w_1, w_2 in r_3 such that

$t[A] = u_1[A] = v_1[A]$ and $t[C] = u_1[C] = w_1[C]$
$s[A] = u_2[A] = v_2[A]$ and $s[C] = u_2[C] = w_2[C]$.

Since $t[A] = s[A]$, we have $u_1[A] = u_2[A]$. But since R_1 and R_2 have a l-to-N relationship from R_1 to R,, and they are connected through A, A must have unique values in r_1. Hence $u_1 = u_2$ and accordingly $u_1[C] = u_2[C] = w_2[C]$.

Therefore r will contain a tuple z where

$z[A] = v_1[A] = t[A] = s[A]$
$z[B] = v_1[B] = t[B]$
$z[R - AUB] = w_2[R\text{-}AUB] = s[R - AUB]$.

Thus R has an MVD $A \rightarrow\rightarrow B$. By a similar argument, R has $C \rightarrow\rightarrow D$. Q.E.D.

Corollary: Let relation schemes R_1 and R_2 have an M-to-N relationship with respect to a set of attributes A. The relation scheme R whose relation instances are natural joins on A of two relations r_1 for R_1 and r_2 for R_2 has MVDs $A \rightarrow\rightarrow (R_1 - A)$ and $A \rightarrow\rightarrow (R_2 - A)$.

Proof: We can consider a two-relation join of r_1 and r_2 as a three-relation join of r_1, r_2, and an imaginary relation $\Pi_A r_1 \cup \Pi_A r_2$. Then the relation scheme R_3 corresponding to this imaginary relation has l-to-N relationships with R_1 and R,, with respect to A, as shown in Figure A-3.

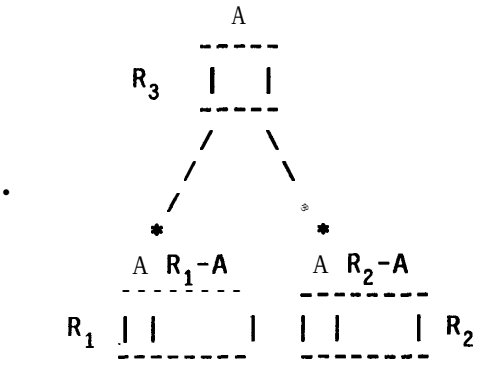

Figure A-3: Relation R_3 has l-to-N Relationships with R_1 and R_2.

Thus relation scheme R has MVDs $A \rightarrow\rightarrow (R_1 - A)$ and $A \rightarrow\rightarrow (R_2 - B)$ from Theorem 2. Q.E.D.

Appendix B. Equivalent Restriction Frequency of a Partial-Join

In Section 1.6, the ***equivalent restriction frequency*** of a partial-join using the join index method was defined as the ratio of the gain in access cost by having the restriction indexes in a partial-join to the gain in access cost that the same restriction indexes would yield in the joint restriction with the join index. We shall show in this section that this equivalent restriction frequency of a partial join using the join index method performed on relation R_2 can be calculated, with one exceptional case, as Cf_{12}/F_a, where Cf_{12} is the coupling factor from relation R_1 to relation R_2 and F_a is the selectivity of the join columns of relation R_2

By formulating the partial-join cost and the cost of the joint restriction in both cases in which the restriction index is used and in which the restriction index is not used (or does not exist), we shall show that the number of block accesses saved in a partial-join is the same as the number of block accesses saved in the joint restriction of the join index and the restriction index used in the partial-join multiplied by Cf_{12}/F_a.

We have three general cases: in Case 1 both the join index and the restriction index are nonclustering; in Case 2 the join index is nonclustering, while the restriction index is clustering; in case 3 the join index is clustering, while the restriction index is nonclustering.

Case 1: both the join index and the restriction index are nonclustering

a. When the restriction index is used

Joint restriction cost = $b(m,p,F_a \times F_i \times n)$

Partial-join cost = $(Cf_{12}/F_a)\, b(m,p,F_a \times F_i \times n)$

In a joint restriction, the number of records selected is $F_a \times F_i \times n$. We assume that these records are evenly spread and are accessed in TID order. Thus we get $b(m,p,F_a \times F_i \times n)$ block accesses. In a partial-join, we are following the join index in the order of join column value, and $F_a \times F_i \times n$ records are accessed for a distinct join column value. Since these records are spread over the entire file and are accessed in TID order, we get $b(m,p,F_a \times F_i \times n)$ block accesses. This proccdure is rcpcatcd for every distinct join column value selected by

the coupling effect and the join selectivity (i.e., according to the coupling factor). The total number of distinct join column values are $1/F_a$. Therefore, as the partial-join cost, we have $(Cf_{12}/F_a)\, b(m,p,F_a \times F_i \times n)$.

b. When the restriction index is not used (or does not exist)

Joint restriction cost = $b(m,p,F_a \times n)$

Partial-join cost = $(Cf_{12}/F_a)\, b(m,p,F_a \times n)$

An analysis applies that is the same as above except that the restriction index is not used. Thus, we have $F_a \times n$ selected records instead of $F_a \times F_i \times n$.

Case 2: the join index is nonclustering while the restriction index is clustering

There are two cases to be considered separately: when $F_i \geq 1$ and when $F_i < 1$.

1. When $F_i \geq 1$

a. When the restriction index is used

Joint restriction cost = $b(F_i \times m, p, F_a F_i \times n)$

Partial-join cost = $(Cf_{12}/F_a)\, b(F_i \times m, p, F_a F_i \times n)$.

This case is almost identical to Case 1, except that the restriction index is clustering and the range within which the selected records can bc found is limited to $F_i \times m$ blocks instead of m (the number of blocks of the entire file). To use b function it is required that $F_i \times m \geq 1$.

b. When the restriction index is not used (or does not exist)

Joint restriction cost = $b(m,p,F_a \times n)$

Partial-join cost = $(Cf_{12}/F_a)\, b(m,p,F_a \times n)$

This case is exactly the same as Case l-b.

2. When $F_i \times m < 1$

a. When the restriction index is used

Joint restriction cost = $F_i \times b(1/F_i, F_i \times n, F_a \times n)$
Partial-join cost = $(Cf_{12}/F_a) \times F_i \times b(1/F_i, F_i \times n, F_a \times n)$.

Since $F_i \times m < 1$ and the restriction index is clustering, all records selected according to the restriction index will be confined in an area smaller than 1 block (let us call this a ***selected area).*** Let us assume that this selected area resides within a physical block (i.e., we ignore the case in which this selected area resides on the' border of two blocks). If we assume that the file is divided into logical blocks of the same size as this selected area, the probability that this selected area will be hit by a joint restriction is

$(1/(1/F_i))\, b(1/F_i, F_i \times n, F_a \times n)$.

This is also the probability that the physical block containing the selected area will be hit (note that there are $1/F_i$ logical blocks in the file). This is also the number of physical blocks to be hit by the joint restriction, since the physical block containing the selected area is the only one that can possibly be accessed.

In a partial-join, the same analysis is valid for each distinct join column value, assuming that the same block must be fetched again if a repeated forward scan inside this block is to be performed. Thus the partial-join cost is the product of (Cf_{12}/F_a) and the joint restriction cost.

b. When the restriction index is not used

Joint restriction cost = $b(m,p,F_a \times n)$
Partial-join cost = $(Cf_{12}/F_a)\, b(m,p,F_a \times n)$

This case is exactly the same as Case l-b.

Case 3: the join index is clustering, while the restriction index is nonclustering

1. When $F_a \times m \geq 1$

a. When the restriction index is used

Joint restriction cost = $b(F_a \times m, p, F_a \times F_i \times n)$
Partial-join cost = $(Cf_{12}/F_a)\, b(F_a \times m, p, F_a \times F_i \times n)$.

An analysis similar to Case 2-l-a applies, except that the range of the selected records is limited to $F_a \times m$ blocks instead of $F_i \times m$.

b. When the restriction index is not used

Joint restriction cost = $F_a \times m$

Partial-join cost = $(Cf_{12}/F_a) \times F_a \times m = Cf_{12} \times m$.

Since the join index is clustering, the number of blocks accessed is proportional to the number of records selected.

2. When $F_a \times m < 1$

a. When the restriction index is used

Joint restriction cost = $(1/(1/F_a))\, b(1/F_a, F_a \times n, F_i \times n)$

Partial-join cost = $b(m, 1/(F_a m), Cf_{12} \times b(1/F_a, F_a \times n, F_i \times n))$.

The joint restriction cost can be obtained by a similar analysis used in Case 2-2-a, except that the roles of F_a and F_i are interchanged.

In the partial-join, the entire file is divided into $1/F_a$ logical blocks, each of which contains $F_a \times n$ records. According to the restriction index, $F_i \times n$ records are selected; the number of logical blocks selected by this restriction is $b(1/F_a, F_a \times n, F_i \times n)$.

The coupling factor Cf_{12} determines how many distinct join column values are actually selected. Since one logical block corresponds to one distinct join column value, the number of logical blocks selected according to the coupling factor and the selectivity of the restriction index is $Cf_{12} \times b(1/F_a, F_a \times n, F_i \times n)$.

To calculate the number of physical blocks hit, let us assume that the entire file consists of m blocks, each of which contains $1/(F_a m)$ logical blocks. Since $Cf_{12} \times b(1/F_a, F_a \times n, F_i \times n)$ logical blocks are selected, the number of physical blocks that will be hit is $b(m, 1/(F_a m), Cf_{12} \times b(1/F_a, F_a \times n, F_i \times n))$.

b. When restriction index is not used (or does not exist)

Joint restriction cost = 1

Partial-join cost = $b(m, 1/(F_a m), Cf_{12}/F_a)$

This can be easily derived from Case 3-2-b by setting F_i to 1.

We have seen, in all situations except Case 3-2, that the partial-join cost is equivalent to Cf_{12}/F_a times the joint restriction cost. Accordingly, the cost saved by having the restriction index in a partial-join is Cf_{12}/F_a times the cost saved by having the restriction index in the joint restriction.

Case 3-2 is the only case in which the equivalent restriction frequency of a partial-join using the join index method cannot be represented as Cf_{12}/F_a. The reason is that, in a partial-join, the logical blocks are accessed in a serial order, and thus several logical blocks may cause only one block access. In the case of joint restriction, we need one block access in any case if at least one record is selected.

The derivations of the formulas were introduced to show how we can formulate cost formulas with the b function, as well as to show that, in most cases, equivalent restriction frequency has a simple form, Cf_{12}/F_a.

While the detailed form of cost formulas depend on the specific cost models, we believe that the same principle we used in the derivation can be easily applied to any given model.

Appendix C. Computational Errors

C.1 Comparison of Computational Errors

In this appendix we develop the prediction of the computational errors which occur in the estimation of block accesses discussed in Section 2.4. These computational errors occur due to the limited precision of the computing system used.

Theorem 1: Calculation of Eq. (2.6) to d digits of precision with a possible error of ± 1 in the least significant digit (LSD) requires at least $\log_{10}(mn \log(n)) + d$ valid digits with a possible error of ± 1 in the LSD.

Proof: We shall use a pseudo equality symbol $\doteq$ throughout this proof and the proof of Theorem 2, ighoring the deviation from equality whenever it neither affects the logical flow of the proof nor changes the numerical result significantly.

By Stirling's approximation [KNU-a 73],

$$\Gamma(n+1) = \sqrt{2\pi}\, n(n/e)^n, \text{ and}$$
$$\ln(\Gamma(n+1)) = \ln(\sqrt{2\pi}) + 0.5\ln(n) + n(\ln(n) - 1)$$
$$\doteq n \ln(n),$$

since we are considering relatively large n's.

From Eq. (2.6),

$$b(m,p,k) = 1 - \exp[LGAM(n-p) + LGAM(n-k) - LGAM(n-p-k) - LGAM(n)] \quad (2.17)$$
$$\doteq -LGAM(n-p) - LGAM(n-k) + LGAM(n-p-k) + LGAM(n).$$

Let us consider the case in which k = 1. At this k value, all four terms in Eq. (2.17) are close to $n \ln(n)$, the result is the smallest possible, and we shall get the maximum error. If we assume that evaluation of Eq. (2.17) causes the error of ± 1 in the LSD, then the error of the result will be

$$10^{-x} \times n \ln(n),$$

where x is the number of significant digits.

The exact value of the result of Eq. (2.17) must be l/m, since only one block will be hit. Therefore, the relative error caused by the computation with x significant digits will be

$$(10^{-x} \text{ X } n \ln(n))/(1/m) = (mn \ln(n)) \text{ X } 10^{-x}. \quad (2.18)$$

If we require this to have an error of less than 10^{-d}, so that we have d digits of precision in the result with a possible error of ± 1 in the LSD, Eq. (2.18) must be less than 10^{-d}. Therefore,

$x \geq \log_{10}(mn \ln(n)) + d.$ Q.E.D.

Theorem 2: $x \geq (\log_{10} m)+d + \log_{,,}(d)+ 1$ valid digits with a possible error of ± 1 in the LSD are sufficient in the calculation of Eq. (2.7) to d digits of precision.

Proof: The major cause of the error is in the calculation of 1- l/m as m gets larger, since it requires as many digits as log,, m. We shall use the equality $(1- 1/m)^m = e^{-1}$ throughout, assuming that m is sufficiently large. For convenience let us consider only the first term of Eq. (2.7), since the other terms behave similarly and their absolute values are always less than $(1 - 1/m)^k$.

Let us divide the values of k into 3 ranges: $k < 0.1m$, $k > \ln(10)XdXm$, and $O.lm \leq k \leq \ln(10)XdXm$.

(1) $k < 0.1m$

From a Taylor expansion we have

$(1-1/m)^k = 1-k/m+k(k-1)/2 \times (1/m)^2 \ldots \doteq 1-k/m$, and thus

1- $(1- 1/m)^k \doteq$ k/m.

In the calculation of $(1 - l/m)$ we have an error of 10^{-x}, so that, as a result of computation, we get

$(1-1/m+ 10^{-x})^k \doteq 1- k(1/m- 10^{-x})$.

(For-convenience let us consider only a positive error. Negative errors can be treated similarly.) Accordingly, the error of the overall calculation will be

$(k(1/m- 10^{-x})-k/m)/(k/m) = -10^{-x}$ X m.

Thus, we get a precision of d digits in the result if and only if

10^{-x} X m < 10^{-d}, or

$x \geq (\log_{10} m) + d.$

(2) $k > \ln(10)$ X d X m

In this case $0 < (1 - 1/m)^k < 10^{-d}$. Hence,

$$1 > 1-(1-1/m)^k > 1-10^{-d} \geq 0.9,$$

assuming $d \geq 1$. However, actual computation may yield

$$1-(1-1/m+10^{-x})^k.$$

Since

$$x \geq (\log_{10} m)+d+1,$$

we have

$$10^{-x} \leq (1/m)10^{-(d+1)}.$$

Since

$$\begin{aligned}&(1- 1/m+ 10^{-(d+1)}/m)^k\\ &= (1-(1-10^{-(d+1)})/m)^k\\ &\leq (1-(1-10^{-(d+1)})/m)^{\ln(10)\times d\times m}\\ &= 10^{-(1-10^{-(d+1)})\times d} \doteq 10^{-d}\end{aligned}$$

assuming $d \geq 1$, the relative error, $((1- 1/m)^k - (1- 1/m+ 10^{-x})^k)/0.9$, cannot be greater than $(1/0.9)(10^{-d}) \doteq 10^{-d}$. Thus we have a precision of d digits in the result.

(3) $0.1m < k \leq \ln(10) \times d \times m$

We have

$$\begin{aligned}&\ln[((1- 1/m)+ 10^{-x})/(1-1/m)^k]\\ &= k(\ln(1- 1/m+ 10^{-x})-\ln(1- 1/m))\\ &\doteq k((1- 1/m+ 10^{-x})-(1- 1/m))\\ &= k \times 10^{-x}.\end{aligned}$$

Accordingly,

$$\begin{aligned}&((1-1/m+10^{-x})^k-(1-1/m)^k)/(1-1/m)^k\\ &\doteq \exp(k\times 10^{-x})-1.\end{aligned}$$

a) $m \leq k \leq (\ln 10) \times d \times m$

The relative error will be

$$\begin{aligned}&((1-1/m+10^{-x})^k-(1-1/m)^k)/(1-(1-1/m)^k)\\ &< ((1-1/m+10^{-x})^k-(1-1/m)^k))/(1-1/m)^k\\ &\doteq \exp(k\times 10^{-x})-1\\ &\leq \exp((k/md)\times 10^{-(d+1)})-1\end{aligned}$$

$$\leq \exp(\ln(10)\times 10^{-(d+1)}) - 1$$
$$< \ln(10)\times 10^{-(d+1)}$$
$$= 0.23\times 10^{-d}.$$

Thus, we have a precision of d digits.

b) $0.1\ m \leq k \leq m$

We have

$$(1- 1/m)^k \doteq 1 - k/m \leq 0.9.$$

Hence, the relative error will be

$$((1-1/m+10^{-x})^k-(1-1/m)^k)/(1-(1-1/m)^k)$$
$$\leq (1/0.1)((1-1/m+10^{-x})^k-(1-1/m)^k)$$
$$\leq 10((1-1/m+10^{-x})^k-(1-1/m)^k)/(1-1/m)^k$$
$$\doteq 10(\exp(k\times 10^{-x}) - 1)$$
$$\leq 10(\exp((k/m)\times 10^{-(d+1)}) - 1)$$
$$\leq 10((k/m)\times 10^{-(d+1)})$$
$$\leq 10\times 10^{-(d+1)}$$
$$= 10^{-d}.$$

This shows that we have a precision of d digits. Q.E.D.

Corollary: Eq. (2.7) requires at least $x \geq (\log_{,,} m)+ d$ valid digits to get d digits of precision in the result.

Proof: This follows from the case (1) of Theorem 2. Q.E.D.

Applying Theorem 2 and its corollary, the actual requirement will be

$$(\log_{10} m) + d \leq x \leq (\log_{10} m) + d + \log_{10}(d) + 1.$$

Example 1: Let us calculate thc number of valid digits required by the evaluationof Eq. (2.7) and Eq. (2.6), respectively, when $m = 10^6$, $p = 10$, $n = 10^7$, and we need a precision of 2 digits in the result.

(a) For Eq. (2.7),

$$\log_{10}(10^6)+2+\log_{10}(2)+1=9.3,$$
$$\log_{10}(10^6)+2=8, \text{ and}$$
$$\mathbf{8 \leq x \leq 9.3.}$$

(b) For Eq. (2.6),

$$x = \log_{10}(10^6 \times 10^7 \times \ln(10^7)) + 2$$
$$= 16.3.$$

We note that Eq. (2.6) requires roughly twice as many valid digits as does Eq. (2.7). □

In the exhaustive calculation we made over the range specified in Section 2.3, the maximum error (0.2%) occurred at $m = 10^6$, $p = 1$, and $k \ll m$ (i.e. k), which actually corresponds to the lower bound given in the corollary.

Example 2: The error of 0.2% is equivalent to a precision of 2 digits according to our definition, since 0.998 compared with 1.0 clearly has an error exceeding 1 in the third digit, and the first and second-digits are the only valid digits with possible error of ± 1 in the LSD. Thus, the number of valid digits x of the computer required by Eq. (2.7)) when $m = 10^6$ will be

$$8 \leq x \leq 9.3$$

The DECSYSTEM-20 has 2^{-27} of resolution, approximately corresponding to 8 valid digits, which confirms our result. □

C.2 Computational Error in an Extended Range

The maximum computational error when the number of blocks m is extended to 10^7 is 4.3%; it occurs at $k = 1$ for all values of p.

We assumed throughout that m has only integer values. However, computer calculation performed over all combinations of the following range shows that the maximum deviation of Eq. (2.7) from the exact formula is 3.7%, even for the real values of m.

- $1.1 \leq p \leq 3.9$ with increments of 0.1,
- $1 \leq p \leq 10$ where p is an integer,
- $1.1 \leq m \leq 3.9$ with increments of 0.1.

References

[BAT 80] Batory, D. S. and Gotlieb. C. C., "A Unifying Model of Physical Databases," Tech. report CSRG-109, Computer Systems Research Group, University of Toronto, April 1980.

[BAY 72] Bayer, R. and McCreight, E., "Organization and Maintenance of Large Ordered Indices," ***Acta Informatica, Vol.*** 1, ***1972, .***

[BLA 76] Blasgen, M. W. and Eswaren, K. P., "On the Evaluation of Queries in a Database System," IBM Research Report RJ1945, IBM, San Jose, Calif., April 1976.

[CAR 75] Cardenas, A. F., "Analysis and Performance of Inverted Database Structures," ***Comm. ACM,*** Vol. 18, No. 5, May 1975, pp. 253-263.

[DEC 78] Digital Equipment Corporation, *DECsystem-10/DECSYSTEM-20* ***Hardware Reference*** *Manual-* ***Central Processor, 1978.***

[ELM 80] El-Masri, R. and Wiederhold G., "Properties of Relationships and Their Representation," ***Natl. Computer*** *Conf.,* AFIPS, Vol. 49, May 1980, pp. 191-192.

[GAM 77] Gambino, T. J. and Gerritsen, R., "A Database Design Decision Support System," ***Proc. Intl.*** *Conf.* ***on Very Large Databases,*** Tokyo, Japan, IEEE, October 1977, pp. 534-544.

[GOT 75] Gotlieb, L., "Computing Joins of Relations," *Proc. Intl. Conf.* ***on Management*** *of* ***Data,*** San Jose, Calif., May 1975, pp. 55-63.

[HAM 76] Hammer, M. and Chan, A., "Index Selection in a Self-Adaptive Database Management System," *Proc. Intl. Conf.* ***on Management*** *of* ***Data,*** Washington, D.C., ACM SIGMOD, June 1976, pp. 1-8.

[HSI 70] Hsiao, D. and Harary, F., "A Formal System for Information Retrieval from Files," ***Comm.*** *ACM,* Vol. 13, No. 4, February 1970, pp. 67-73, Also see Comm. ACM 13, 4, April 1070, p.266.

[IBM 703 IBM, ***System 360*** *Scientific* ***Subroutine Package,*** 1970.

[KAT 80] Katz, Ii. H. and Wong, E., "An Access Path Model for Physical Database Design," ***Proc. Intl.*** *Conf.* ***on Management*** *of* ***Data,*** Santa Monica, Calif., ACM SIGMOD, May 1980, pp. 22-29.

[KIN 74] King, W. F., "On the Selection of Indices for a File," IBM Research Report RJ1341, IBM, San Jose, Calif., 1974.

[KNU-a 73]
Knuth, D., ***The Art*** *of* ***Computer Programming- Fundamental Algorithms,*** Addcson-Wesley, , Vol. 1, 1973.

[KNU-b 73]
Knuth, D., ***The Art*** *of* ***Computer Programming- Sortng and Searching,*** Addison-Wesley, , Vol. 3, ***1973.***

[PEC 75] Pechcrcr, R. M., "Efficient Evaluation of Expression in a Relational Algebra," ***ACM Pacific 7.5 Regional*** *Conference,* San Francisco, April 1975, pp. 44-49.

[ROT 74] Rothnie, J. B. and Lozano, T., "Attribute based file organization in a paged memory environment," ***Comm. ACM,*** Vol. 17, No. 2, February 1974, pp. 63-69.

[SCH 75] Schkolnick, M., "The Optimal Selection of Secondary Indices for Files," ***Information Systems,*** Vol. 1, March 1975, pp. 141-146.

[SCH 81] Schkolnick, M., private communication,.

[SEL 79] Selinger, P. G. et al., "The Optimal Selection of Secondary Indices for Files," *Proc. Intl. Conf.* ***on Management*** *of* ***Data,*** Boston, Mass., May 1979, pp. 23-34.

[SEV 72] Severance, D. ***G., Some Generalized Modeling Structures*** *for* ***Use in Design*** *of File* ***Organizations,*** PhD dissertation, University of Michigan, Ann Arbor, Mich., 1972.

[SEV 75] Severance, D. G., "A Parametric Model of Alternative File Structures," ***Information Systems,*** Vol. 1, No. 2, 1975, pp. 51-55.

[SIL 76] Siler, K. F., "A stochastic evaluation model for database organizations in data retrival systems," ***Comm. ACM,*** Vol. 19, No. 2, February 1976, pp. 84-95.

[SMI 75] Smith, J. and Chang, P., "Optimizing. the Performance of a Relational Algebra Database Interface," ***Comm.*** *ACM,* Vol. 18, No. 10, October 1975, pp. 568-579.

[ULL 80] Ullman J., ***Principles*** *of* ***Database Systems,*** Computer Science Press, Potomac, Maryland, 1980.

[WAT 72] Waters, S. J., "File design fallacies," ***The Computer Journal,*** Vol. 15, No. 1, 1972, pp. l-4.

[WAT 75] Waters, S. J., "Estimating magnetic disc seeks," ***The Computer Journal,*** Vol. 18, No. 1, 1975, pp. 12-17.

[WAT 76] Waters, S. J., "Hit ratios," ***The Computer Journal,*** Vol. 19, No. 1, 1976, pp. 21-24.

[WHA 81] Whang, K., "Separability – An Approach to Physical Database Design," Tech. report, Stanford University, 1981, to be published.

[WIE 77] Wicderhold, ***G., Database Design,*** McGraw-Hill Book Company, New York, 1977.

[WIE 79] Wicdcrhold, G. and El-Masri, R., "The Structural Model for Database Design," *Proc. Intl. Conf.* ***on Entity Relationship Approach,*** Los Angeles, Calif., December 1979, pp. 247-267.

[WON 76] Wong, E. and Youseffi, K., "Decomposition – A Strategy for Query Processing," ACM ***Trans. Database Systems,*** Vol. 1, No. 3, September 1976, pp. 223-241.

[YAO 77] Yao, S. B., "An Attribute Based Model for Database Access Cost Analysis," ***ACM Trans. Database Systems,*** Vol. 2, No. 1, March 1977, pp. 45-67.

[YAO 79] Yao, S. B., "Optimization of Query Evaluation Algorithm," ***ACM Trans. Database Systems,*** June 1979, pp. 133-155.

[YUE 75] Yuc, P. C. and Wong, C. K., "Storage cost considerations in secondary index selection," ***International Journal*** *of* ***Computer and Information Sciences,*** Vol. 4, No. 4, 1975, pp. 307-327.