A Physical Database Design Evaluation System for CODASYL Databases

HERMAN LAM, STANLEY Y. W. SU, AND NAGESHWAR R. KOGANTI

Abstract—**This paper describes a physical database design evaluation system which is an interactive design tool for designing CODASYL databases. The system is composed of three main modules: a user interface, a transaction analyzer, and a core module. The user interface allows a designer to interactively enter information concerning a database design which is to be evaluated. The transaction analyzer allows the designer to specify the processing requirements in terms of typical logical transactions to be executed against the database and translates these logical transactions into physical transactions which access and manipulate the physical databases. The core module is the implementation of a set of analytical models and cost formulas developed for the manipulation of indexed sequential and hash-based files and CODASYL sets. These models and formulas account for the situation in which occurrences of multiple record types are stored in the same area. The paper also presents the results of a series of experiments in which key design parameters are varied. A design example is also given to demonstrate the manner in which the evaluation system can be used to aid a designer in making design decisions. The system is implemented in UCSD Pascal running on IBM PC's. It is presently being used by the Navy Ships Supply Control Center and the National Bureau of Standards in their database design efforts.**

Index Terms—**CODASYL database, evaluation system, physical database design.**

I. Introduction

In the past decade, there has been much research on analytical models for performance evaluation of file and database structures [1], [2], [5], [7], [8], [9], [12], [14], [18], [19], [20], [24], [26], [32], [33], [34]. Various formulas have been proposed for estimating the number of physical block accesses required to access records stored in a file based on these structures. Although much of these works have proven valuable, very few of the results obtained have been incorporated into a working, comprehensive system that provides a valuable automated tool to assist a database design team in the management of complex database design processes.

In this paper, we describe a physical database design evaluation system for CODASYL Databases which has been implemented in UCSD Pascal running on IBM PC's. The inputs to the evaluation system are database designs and processing requirements. Database designs are represented in the form of database parameters that are relevant to database performance. Processing requirements are represented in the form of transactions that are representative of the processing needs of an organization. Given a candidate database design to be evaluated and some processing requirements, the output of the evaluation system is the performance evaluation of the candidate database design, expressed in terms of physical data block accesses. By modifying physical database design parameters, alternative physical database designs can be analyzed and compared.

The evaluation system has the following two features. First, it extends some existing models and analytical formulas for data block accesses for blocks which contain record occurrences of multiple record types. The existing analytical models for performance evaluation and the associated cost formulas are typically based on the assumption that a physical block contains only the record occurrences of a single record type under consideration, which we shall call the *object record type*. In the terminology of the CODASYL network data model, the assumption is that there is only one record type assigned to an area. Unfortunately, in practice, there may be record occurrences of other record types that are stored in the same area and, thus, in the same data block along with the records of the object record type.

Secondly, it provides the model and cost formulas for processing CODASYL sets. Although there are analytical models developed for the common file and database structures such as the hash-based files [4], [27], [31], the indexed sequential files [1], [16], [25], B+ trees [3], [15], [23], and others [2], [13], [21], [28]. There are currently no satisfactory models for the analysis of the CODASYL set database structure to suit our purpose. In the existing works, the modeling of the CODASYL set is either performed at the logical level [22] or the effect of placing the owner and member occurrences in the same data blocks have not been studied [6], [10], [11].

This paper is organized as follows. In Section II, the components of an implemented physical database design evaluation system are described. In Section III, the core module is described in detail. The models used for modeling hash-based files, indexed sequential files, and the CODASYL sets are presented. Those for hash-based and indexed sequential file structures are adopted from the existing works [3], [4], [30] with some extension. Those for the CODASYL set structure were developed by the authors and are documented in detail in [29]. Additionally, the concepts of "effective area" and "effective block

Manuscript received January 31, 1986; revised June 30, 1986.

The authors are with the Database Systems Research and Development Center, University of Florida, Gainesville, FL 32611.

IEEE Log Number 8821988.

sizes'' for handling multiple record types within an area are described in this section. In Section IV, experimental results from the testing of the evaluation system are given. A design example is also given to demonstrate the manner in which the evaluation system can be used to aid the database designer in making design decisions. Finally, Section V provides the summary and conclusion for this paper.

II. The Evaluation System

The functional block diagram of the evaluation system is shown in Fig. 1. The evaluation system is composed of three main modules: the user interface, the transaction analyzer, and the core module.

A. The User Interface

The function of the user interface is to assist the user to interactively enter the information concerning the database design to be evaluated. The information includes the values for relevant DDL parameters and estimated values for each record type, set type, and area. Under an editing mode, the designer can make changes to the database design under evaluation, correcting erroneous input or modifying the parameters for subsequent runs. In the compilation mode, the database design inputs are checked for correctness and consistency, and are stored in a database design (DD) dictionary.

The DD dictionary is the key output of the user interface and contains database design information necessary for the transaction analyzer to translate the processing requirement specifications and the core module to perform the evaluation. Internally, the DD dictionary is represented primarily by three tables. The information contents of these tables are specified below:

a) Record Type Definition Table:
1) Record type name
2) Location mode
3) Key attribute
4) Record size
5) Estimated number of occurrences for each record type
6) Area in which the occurrences of the record type is stored
7) Estimated number of occurrences that are initially loaded (for indexed sequential file only)
8) Blocking factor of the blocks in the associated index file (for indexed sequential file only)

b) Set Type Definition Tables
1) Set type name
2) Set mode
3) Owner record type
4) Member record type
5) Set pointer type
6) Membership class
7) The blocking factor of the blocks in the associated pointer file (for set mode equal to pointer array)

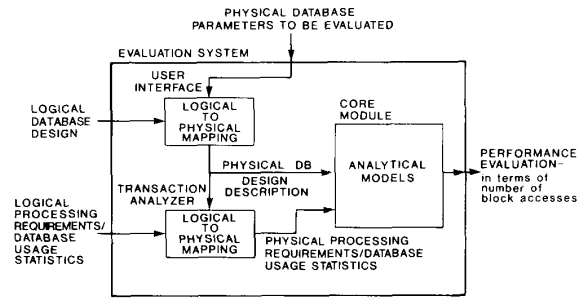


Fig. 1. The evaluation system.

c) Area Definition Table
1) Area name
2) Block size of the blocks in the area
3) Number of blocks allocated for the area
4) Initial loading factor of the blocks in the area
5) Estimated storage patterns for the records stored within the area.

B. The Transaction Analyzer

The transaction analyzer allows the user to specify his/her processing requirements in an interactive manner. Processing requirements are defined in terms of a set of logical transactions. A logical transaction describes a traversal of a database by making references to logical database objects (record types and set types), specifying the operations on the database objects, and defining the traversal paths and the frequency and/or probabilities of these paths. The transaction analyzer makes use of the DD dictionary to translate the logical transactions into a set of physical transactions. A physical transaction describes the same database traversal as the corresponding logical transaction but with reference to physical database objects such as data files, index files, pointer array files, chain set types, and pointer array set types. A physical transaction, which is the output of the transaction analyzer, is represented in the form of a cost equation such as the following example:

```
COST (T1) = 50 * (COST (RETRCALC,
                  REC-B,SOME,KEY) +
                  10 * (COST (RETRMEMB,REC-D,B-D,
                  P-ARR,UNIQUE,FOUND))
```

The above equation states that the cost of Translation T1 is equal to 50 times the cost of a retrieval using CALC mode plus 10 times the cost of retrieving members of a set B-D. Numerical values are frequencies of operations.

The set of cost equations is represented by a stack and a cost table. Each entry in the cost table represents the cost of a single operation within a transaction. The stack is used to implement a post order implementation of the cost equations. The information content of the cost table is as follows:

1) Operation type—retrieval through a record type, retrieval through a set type, insertion, deletion, or update of record, connnection or disconnection of a set.

2) Record type that is referenced by the operation.
3) Set type, if any, that is referenced by the operation.
4) Retrieval type—unique, some, or all.
5) Expected retrieval result—found or not found.
6) Search condition—key or non-key.

C. The Core Module

The core module consists of a set of cost equations for evaluating a candidate physical database design based on some specified processing requirements. The cost equations are formulated based on some analytical models developed for various physical structures and different types of database operations on these structures. The inputs to the core module are the DD dictionary from the user interface and the set of physical transactions from the transaction analyzer. The output of the core module is a quantitative measure of I/O cost involved in carrying out the specified processing of a candidate design. The core module is described in more detail below.

III. The Core Module

The core module implements a set of analytical models each of which consists of a set of cost formulas defined as functions of the types of database operations to be performed (e.g., RETRIEVE, DELETE, RETRMEMB, unique, some, all, key, non-key, found, not found, etc.) and the characteristics of the referenced data objects such as record types, set types, and areas. In this work, analytical models and cost formulas have been developed for three categories of file and database structures; indexed sequential, hash-based and the CODASYL set structures. They are detailed in [17], [29], and are summarized in the following subsections.

A. Indexed Sequential and Hash-Based Structures

An indexed sequential file consists of an ordered physical sequential file and a hierarchy of block indexes, each ordered by primary key values in the same way in which the data file is ordered. A model for the indexed sequential file is shown in Fig. 2, consisting of three types of storage blocks: 1) index blocks, 2) primary data blocks, and 3) overflow data blocks.

The indexed sequential records are loaded into the primary data blocks in the order of their key sequence during the initial loading of the file. Each time a primary data block is filled, an index entry, representing, for example, the value of the highest key in that data block, is stored in an index block. A new index entry is stored for every index block that is filled and a hierarchy of index blocks is built in this fashion with a single block at the topmost level.

Any further additions to the indexed sequential file are made by inserting the records into the appropriate data block based on their index keys. The records are stored in the overflow blocks chained to the primary data block if the primary data block is full. A primary data block together with its overflow blocks will be referred to as a *data node*.

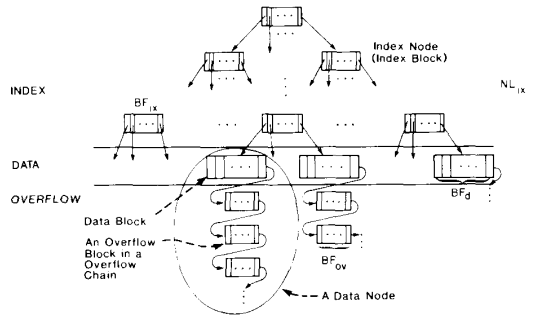


Fig. 2. A model for an indexed sequential structure.

Access to a specific data record is performed by searching through the index blocks to arrive at the data node that contains the record, and sequentially searching through the data records in the node. Those transactions that are sequential in nature are satisfied efficiently by bypassing the index and performing a sequential scan of all data nodes.

Given below are various parameters that are relevant to the performance of the indexed sequential file. They are either provided in the schema of a database or are given as estimated values by the database designer.

- LF The initial loading factor for a data block; i.e., the fraction of the data block to be filled during initial load. The loading factor affects the number of data nodes.
- BF_d Blocking factor for the data block in a data node; i.e., the number of data records that can be stored in a data block.
- BF_{ov} Blocking factor for an overflow block; i.e., the number of data records that can be stored in an overflow block.
- BF_{ix} Blocking factor for a block in the index file; i.e., the number of index records that can be stored in an index block.
- NR_o The number of data records that are stored in the indexed sequential file at the initial load.
- NR The number of data records that are stored in the indexed sequential file currently.

Based on the parameters listed above, other parameters relevant to the performance of an indexed sequential file can be calculated. They include the number of index levels, the number of data nodes, the effective block sizes, effective blocking factors, and relevant expected values.

Associated with the model of the indexed sequential structure is a set of cost formulas. The cost formulas are developed for the four major database operations: retrieval, insertion, deletion, and modification. The cost formulas for retrieval are further classified in three types:

1) retrieval of a unique record
2) retrieval of some records; i.e., all those records that satisfy some search condition
3) retrieval of all records; i.e., a scan of the entire file.

Furthermore, the cost is dependent on whether the search condition is based on the key or non-key attributes, and whether the expected search result is found or not found.

A hash-based file is one whose records are stored and retrieved based on addresses computed using a hashing function. A model of the hash-based file is shown in Fig. 3. Two types of storage blocks are distinguished: 1) primary data blocks and 2) overflow data blocks. A primary data block together with its chain of overflow data blocks is referenced as a data node. All records whose attribute values are hashed to the same data node will be first stored in the primary block and then the overflow chain. Access to a specific data record is performed by arriving at the particular data node and performing a sequential search of the data and overflow blocks.

Given below are various parameters that have effect on the performance of the hash-based file. They are either provided in the schema or are given as estimated values by the database designer.

- BF_d Blocking factor for the data block in a data node.
- BF_{ov} Blocking factor for an overflow block in an overflow chain.
- NB_d The number of data blocks allocated for the hash-based file.
- NR The number of data records that are stored in the hash-based file.

As in the case of an indexed sequential file structure, other parameters relevant to the performance of the hash-based file can be calculated based on the parameters given above. They include the number of data nodes, the effective block sizes, the effective blocking factors, and relevant expected values.

Associated with the model of the hash-based file structure is a set of cost formulas which are developed for the four major database operations: retrieval, insertion, deletion, and modification. The cost formulas for retrieval are also a function of retrieval type (unique, some, or all), search condition (key or non-key) and expected search result (found or not found).

The models for both the indexed sequential and hash-based structures and their associated cost formulas are extensions of the works reported in [3], [4], [30]. They are given in detail in [29].

B. CODASYL Set Structure

A CODASYL set type consists of an owner record type and a member record type. The member record occurrences can be related to their corresponding owner record occurrence through one of two set modes: CHAIN and POINTER ARRAY.

A model of a set occurrence with a set mode of CHAIN is shown in Fig. 4. At the logical level, as shown in Fig. 4(a), the owner record occurrence O_i is linked to the first member record occurrence m_1, m_1 to m_2, and so on. The last member record occurrence m_M is connected back to the owner record O_i.

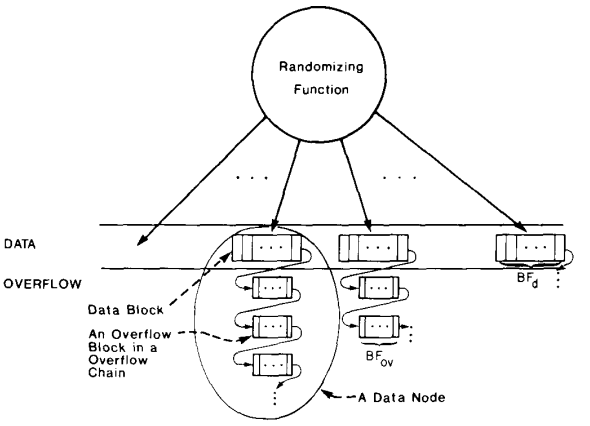


Fig. 3. A model for hash-based structure.

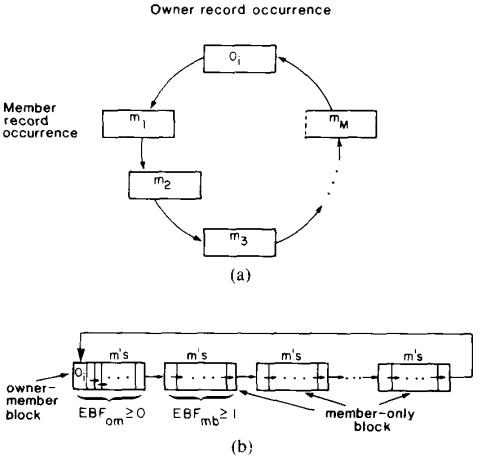


Fig. 4. Models of a chain set occurrence. (a) Logical level. (b) Physical level.

At the physical level, a set occurrence with a set mode of CHAIN is modeled as shown in Fig. 4(b). The owner and the corresponding members are still chained together. However, they are now contained in data blocks. Two types of data blocks are distinguished. Those blocks that contain an owner record and some member record occurrences are called *owner-member blocks* and those that contain members only are called *member-only blocks*.

The effective blocking factor for the owner-member block, represented by EBF_{om} in Fig. 4(b), is the number of member records that are stored in the same block as their owner. EBF_{om} is greater than or equal to zero. EBF_{om} is zero when the owner record occurrence is stored in a separate block. The effective blocking factor for the member-only block EBF_{mb} represents the number of member records that can be stored in the member-only block. Note that the optional prior and owner pointers are not shown in the models in Fig. 4 since they have no effect on the determination of the EBF's. The effects of these pointers

on the database performance are reflected in the cost formulas for the CODASYL set structure given in the Appendix.

A model of a pointer array set occurrence is shown in Fig. 5. Fig. 5(a) shows the pointer array set occurrence at the logical level. The owner record occurrence O_i points to a pointer array which contains pointers to the member record occurrences of the set. Each of these member record occurrences contains a pointer back to its owner.

A pointer array set occurrence at the physical level is modeled as shown in Fig. 5(b). A pointer from the owner record occurrence O_i points to the pointer array which could be stored in several blocks. The pointer array consists of pointers pointing to the member record occurrences. Each of the member record occurrences has a pointer back to the owner record occurrence.

The effective blocking factor of the owner-member block EBF_{om} in Fig. 5(b) again represents the number of member record occurrences that can be stored in the same block with their owner. The effective blocking factor for a member-only block EBF_{mb} represents the number of member record occurrences that can be stored in the member-only block, and the blocking factor for the pointer array block BF_{pa} represents the number of pointers which can be stored in a pointer array block.

Given below are various parameters that are associated with a CODASYL set structure. They are relevant to the performance of the CODASYL set structure, and are either given in the schema or given as estimated values by the database designer.

- BF_d Blocking factor for the primary data block in a data node.
- BF_{ov} Blocking factor for an overflow block in an overflow chain.
- BF_{pa} Blocking factor for a pointer block.
- NR_{own} The number of record occurrences of the owner record type.
- NR_{mb} The number of record occurrences of the member record type.

Based on the parameters listed above, other parameters relevant to the performance of a CODASYL set structure can be calculated. They include the effective block sizes, the effective blocking factors, and the relevant expected values.

Associated with the model of the CODASYL set structure is a set of cost formulas. The cost formulas are developed for the various types of retrieval, insertion/connection, deletion/disconnection, and modification operations. Retrieval operations include retrieve member, retrieve owner, retrieve prior, retrieve next and retrieve pointers from a pointer array. Furthermore, the cost is dependent on the set mode (CHAIN or POINTER ARRAY), search type (unique, some, or all), and expected search result (found or not found). The model and the associated cost formulas that have been developed are described in detail in [29].

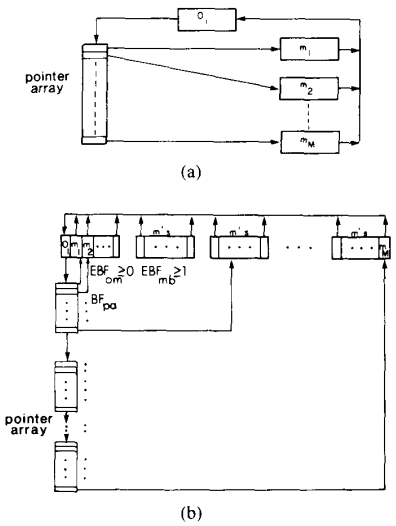


Fig. 5. Models of a point array set occurrence. (a) Logical level. (b) Physical level.

C. Effective Areas and Effective Blocking Factors

The derivation of the cost formulas for an indexed sequential file and for a hash-based file given in [4], [30] is based on the assumption that the object record type under consideration is the only record type in the area. However, in practice, it is likely that record occurrences of other record types may be stored in the same area and in the same data block along with the records of the object record type. In this case, in order to utilize the models of the indexed sequential file and the hash-based file and the associated cost formulas, the concepts of *effective area* and *effective block size* need to be introduced.

At any point of a design evaluation, we are concerned with only the records of the object record type and its access. Consequently, we can "factor out" the other records present in the same area and determine what fraction of a data block or an overflow block is allocated to the object record type. Thus, the initial block size (IBS) of a physical area is now reduced to the *effective* block size (EBS) of an effective area. We can visualize the effective area to contain only the records of the object record type, stored in data blocks having the block size of EBS_d and overflow blocks having the block size of EBS_{ov}. In this manner, the models of indexed sequential files and hash-based files from the previous sections and the associated cost formulas can be utilized. What is necessary, however, is a method to determine the effective block size and effective blocking factor for both the data block (EBS_d, EBF_d) and the overflow blocks (EBS_{ov}, EBF_{ov}) for the object record type. The effective block size (EBS) and the effective blocking factor (EBF) are both units of measurement of the size of a block. The former is the size of a block in number of bytes and the latter is the size in number of records.

Similarly, for the model of a CODASYL set and its cost formulas, we need to determine the effective block size

and effective blocking factor of the member record type in the block containing the associated owner occurrence (EBS_{om}, EBF_{om}), and the effective block size and the effective blocking factor of the member record type in blocks which contain members only (EBS_{mb}, EBF_{mb}).

1) Effective Block Size EBS_d and Effective Blocking Factor EBF_d: To provide the reader with some idea of the method we use to determine the effective block sizes and effective blocking factors, the derivation for EBS_d and EBF_d is presented here. The effective block size EBS_d is that part of the *primary data* block of a node in an area which is allocated to the object record type. As an example, consider the area in Fig. 6(a), which contains four record types: RECA, RECB, RECC, and RECD; and three set types: SETAB, SETBC, and SETAD. The idea is that no matter how many other types of record occurrences are in the same area, we are only concerned with the records of the object record type and how they are intermixed with the "other" record occurrences.

A model to represent this concept is given in Fig. 7. With respect to the object record type, the record distribution of an area can be described by the following parameters:

1) An initial cluster of records of the other record types—represented in Fig. 7 as the initial gap size INITGS.

2) and repeating groups of the following:
   a) A cluster of object records—represented in Fig. 7 as the cluster size CLS.
   b) A cluster of other records—represented in Fig. 7 as the gap size GAPS.

As an example, let us assume that the typical storage pattern of record distribution for the area in Fig. 6(a) is represented as shown in Fig. 6(b). In this case, an occurrence of RECA, A_i, is stored first, followed immediately by A_i's corresponding member record occurrences of RECB, B_{ij} (along with B_{ij}'s corresponding record occurrences of RECC C_{ijk}); followed by A_i's corresponding member record occurrences of RECD, D_{ij}.

If the object record type (the record type under consideration) is RECA, then the values for the model parameters are as follows:

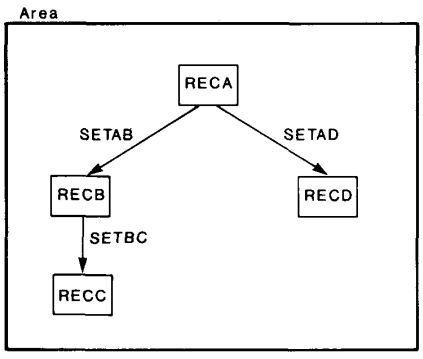
```
INITGS = 0
CLS = 1*RECSZ(A)
GAPS = 3*(RECSZ(B) + 2*RECSZ(C)) +
  4*RECSZ(D)
```

If the object record type is RECB, then the values for the model parameters are as follows:

```
INITGS = 1*RECSZ(A)
CLS = 1*RECSZ(B)
GAPS = (2(2*RESCZ(C)) + 1*(2*RECSZ(C)
  +4*RECSZ(D)+1*RECSZ(A))) / 3
```

If the object record type is RECC, then the values for the model parameters are as follows:

```
INTITGS = 1*RECSZ(A) + 1*RECSZ(B)
CLS = 2*RECSZ(C)
```



(a)

Assume there are N occurrences of RECA. For each owner occurrence of RECA, there are 3 member occurrences of RECB and 4 member occurrences of RECD. Additionally, for each owner occurrence of RECB, there are 2 member occurrences of RECC.

$A_1B_{11}C_{111}C_{112}B_{12}C_{121}C_{122}B_{13}C_{131}C_{132}D_{11}D_{12}D_{13}D_{14}A_2B_{21}C_{211}C_{212}B_{22}C_{221}C_{222}B_{23}$

$C_{231}C_{232}D_{21}D_{22}D_{23}D_{24} \cdots A_nB_{n1}C_{n11}C_{n12}B_{n2}C_{n21}C_{n22}B_{n3}C_{n31}C_{n32}D_{n1}D_{n2}D_{n3}D_{n4}$

(b)

Fig. 6. Multiple record types within an area and a sample distribution. (a) Multiple record types within an area. (b) An example storage pattern of record distribution.

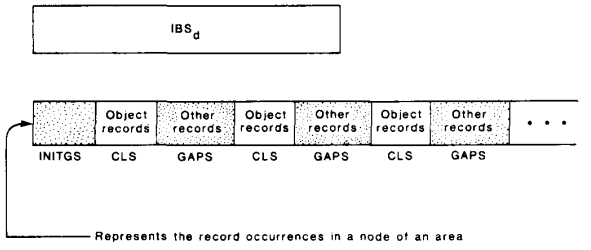


Fig. 7. A model for deriving the formula for EBS_d.

```
GAPS = (2*RECSZ(B) + 1*(4*RECSZ(D)
  +RECSZ(A)+RECSZ(B))) / 3
```

If the object record type is RECD, then the values for the model parameters are as follows:

```
INITGS = RECSZ(A) + 3*(RECSZ(B)
  +2*RECSZ(C))
CLS = 4*RECSZ(D)
GAPS = RECSZ(A) + 3*(RECSZ(B)+2*RECSZ(C))
```

Of course, if a different storage pattern is specified by the database designer, then values for the model parameters would be different. However, the method of calculation is the same.

2) Parameters and Fomulas for EBS_d and EBF_d: In this section, we shall define the parameters and formulas that are relevant to the calculation of EBS_d and EBF_d. Three classes of such parameters are given in Table I: parameters given in the schema that are relevant to the calculation of EBS_d and EBF_d, parameters given as estimated values by the database designer, and model parameters for the model shown in Fig. 7.

TABLE I
PARAMETERS FOR THE CALCULATION OF EBS_d AND EBF_d

Class I—Parameters Given in the DDL Definition

$RECSZ_{obj}$—Record size of the object record type.
$RECSZ_i$ —Record size of the other record types in the area.
IBS_d —Initial block size for the primary data block.

Class II—Parameters Given as Estimated Values by the Database Designer

NR_{obj}—Estimated number of occurrences for the object record type
NR_i —Estimated number of occurrences for each of the other record types in the area.
Typical storage pattern of record distribution—specified by the database designer.

Class III—Model Parameters

INITGS—Initial gap size: The size of the initial cluster of records of other record types in the primary data block; INITGS > = 0.
CLS —Cluster size: The size of a cluster of object records in the possible repeating groups that follow the INITGS.
GAPS —Gap size: Between clusters of object records, there may be clusters of other records. This is represented by the GAPS.

Based on these parameters, other parameters and formulas can be derived. The general procedure for the derivation of EBS_d and IBF_d is illustrated in Fig. 7 and the resultant parameter and formula derivation is given in Table II. The task is to determine what fraction of the primary data block, represented by IBS_d in Fig. 8, that is allocated to the records of the object record type. Graphically, it is seen from Fig. 8 that the part of the primary data block that is *not* allocated to the object record types are the space occupied by the "other" records, including the space represented by INITGS and the GAPS. Consequently, the space represented by INITGS is first subtracted from the initial block size IBS_d to obtain IBS'_d. If IBS'_d is an even multiple of the group size (GROUPS = CLS + GAPS), then EBS_d is equal to (CLS/GROUPS)*IBS'_d. However, this is generally not the case. Therefore, the space represented by REMS in Fig. 8 must be handled as a special case, as shown in Table II. As mentioned before, the effective block factor EBF_d is simply another unit of measurement of the size of a primary data block, expressed in the number of object records. Therefore, $EBF_d = EBS_d/RECSZ_{obj}$.

In this section, we provided the reader with a general idea of the method in which the effective block sizes and effective blocking factors introduced in the previous section are determined. The derivation for EBS_{ov}, EBS_{om}, and EBS_{mb}, although more complex, are carried out in a similar manner. For additional information concerning the concepts of effective area and effective block sizes, refer to [17], [29].

D. Cost Computation by the Core Module

The core module is the implementation of the analytical models described in the previous sections. For each transaction provided by the transaction analyzer, the core module determines the cost for that transaction based on the data objects (record types, set types, and areas) it references and the operations that need to be performed. Based on the information stored in the DD dictionary (output of the user interface) concerning each referenced data object, a cost is determined for that transaction using the appropriate cost functions. By taking the sum of the costs of individual transactions, the total cost of the entire set of representative transactions is determined. Alternative database designs can be evaluated by modifying values for a selected set of parameters under the edit mode of the user interface, and recalculating the total costs for comparison.

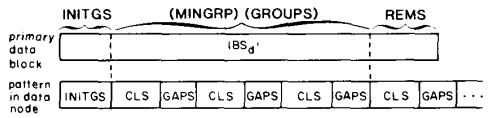


Fig. 8. Illustration for the calculation of EBS_d and EBF_d.

TABLE II
CALCULATED PARAMETERS AND FORMULAS FOR EBS_d AND EBF_d

Class IV—Calculated Parameters and Formulas

IBS'_d—Initial block size minus the initial gap size.
IBS'_d—max(0, IBS_d—INITGS)

Note that if INITGS is greater than IBS_d, then obviously there will be no space left in the data block for the object record type.

GROUPS—Group size.
GROUPS = CLS + GAPS.
MINGRP—Minimum number of groups that can fit *completely* into a data block.
MINGRP = trunc(IBS'_d/GROUPS).

EBS_d = MINGRP*CLS + REMS

where REMS

$$= \begin{cases} \text{CLS} & \text{If REMS} > = \text{CLS} \\ IBS'_d - (\text{MINGRP})*(\text{GROUPS}) & \text{if REMS} < \text{CLS}. \end{cases}$$

$EBF_d = EBS_d/RECSZ_{obj}$.

IV. EXPERIMENTAL RESULTS

A series of experiments has been designed and performed to test and demonstrate the capabilities of the evaluation system. Some results are presented here. They are presented in two sections: In Section IV-A, the results are given for a series of experiments in which different physical database design parameters were varied systematically. It will be shown that these results from the evaluation system are consistent with the expected results based on established design guidelines. In Section IV-B, a design example is given to demonstrate the manner in which the evaluation system can be used to aid the database design team in the physical database design process. The examples used in this section are all based on the sample logical database design shown in Fig. 9. There are six record types and five set types. The task in physical database design is to determine the values for the relevant physical database design parameters.

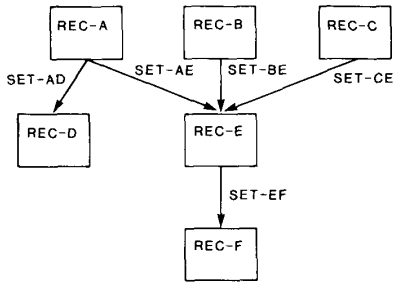


Fig. 9. A sample database.

A. Experimental Results with Varying Parameters

In this section, the results of five experiments will be presented in the form of five sets of graphs illustrating the effect of various parameters on database performance. The physical database design parameters to be used are the block size, the number of data nodes (for hash-based files), initial loading factor (for indexed sequential files), and the effect of the presence of other records in the same area (i.e., record clustering). In these experiments, one of these parameters are varied while holding the other parameters and test transactions on the sample database constant. Each set of graphs consists of three graphs, one for the retrieval of a unique record, one for the retrieval of some records, and one for the retrieval of all records. Each graph plots the number of physical block accesses required to complete the transaction as a function of the parameter being varied.

In the first two examples, we assume that the location mode of the REC-A is CALC (hash-based) and REC-A is placed in an area by itself. We aso assume that the test translation, Transaction T1, shown below is used. The effect of varying the number of data nodes upon the database performance of a hash-based file is shown in Fig. 10 in which the number of overflow block accesses is plotted against the number of data nodes.

TRANSACTION T1
RETRCALC, REC-A, UNIQUE, KEY, FOUND
RETRCALC, REC-A, SOME, KEY
RETRCALC, REC-A, ALL

The number of nodes are varied so that, at the lowest value, there are a number of records in the overflow chain. The nodes are then progressively increased to a value where there is absolutely no overflow at all. It is obvious from the graphs that the number of overflow block accesses decreases with increase in the number of nodes. The number of primary data block (not shown in the graphs) accesses remains constant except for the case where all records are to be retrieved. In this case, the primary data block accesses increase in direct proportion to the increase in the number of nodes.

The second example, shows the effect of varying the presence of other records in the same area as the object record on the database performance of a hash-based file.

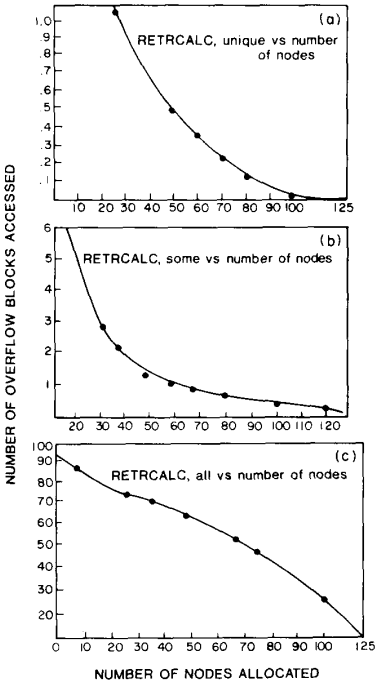


Fig. 10. RETRCALC versus number of nodes.

The test Transaction T1 is again used for this example. The result is shown in Fig. 11.

As expected, the number of block accesses required to perform the transaction increases as the number of other records stored with the object record type increases. Thus, it is clear that if transactions involving only the object record type are to be executed much more frequently than those involving other record types, it is best not to store other records in the same area as the object record type.

In the next example, we assume that the location mode of the REC-C is indexed sequential and the record type is placed in the area all by itself. Transaction T2, shown below, is used. We are interested in determining the effect of varying block size on the performance of the indexed sequential file. The block size is varied over the range of 2000 to 11 000 bytes in steps of 1000. The results are shown in Fig. 12.

TRANSACTION T2
RETRIXSQ, REC-C, UNIQUE, KEY, FOUND
RETRIXSQ, REC-C, SOME, KEY
RETRIXSQ, REC-C, ALL

Figs. 12(a) and 12(c) agree with our intuitive expectation that an increase in block size results in a decrease in the number of block accesses, as is evident from the steep decline of curves for increasing block sizes. However, the curve for retrieval of some records in Fig. 12(b) behaves in an unexpected fashion. It increases for increasing block sizes. Upon closer examination, it is clear that the increase in the block size resulted in an increase in the number of records loaded into each block at the initial load

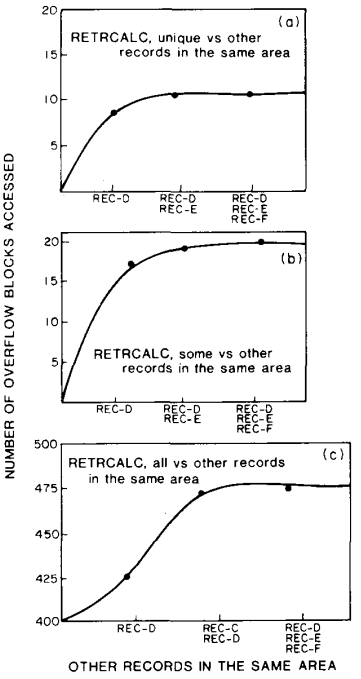


Fig. 11. RETRCALC versus other records in the same area.

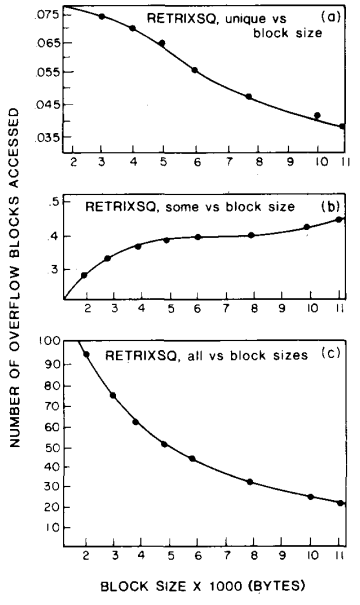


Fig. 12. RETRIXSQ versus block size.

time. Consequently, the resultant number of data nodes decreases. As subsequent records are inserted, there are less nodes to accommodate them. The result is that the overflow chain of each node becomes longer. Since the retrieval of some records is a measure of the length of the overflow chain, it increases with block size.

Although the overflow chain is longer, there are less numbers of them. Therefore, with a larger block size, the total number of overflow blocks (retrieval of ALL) is less. For the retrieval of a unique record, the cost is proportional to the length of the overflow chain *divided* by the number of records in the entire node (primary data block and overflow data block). Although the overflow chain is longer, the number of records in the primary data block (which requires 0 overflow block access) is also larger. In this case, the increase in the denominator of the term dominates the increase in the numerator.

In the next example, the same file structure and transaction are used. However, the initial loading factor is varied and its effect on the performance of the indexed sequential file is studied. The initial loading factor for a file specifies the fraction of the data block that is to be filled by the records at initial load time. An initial loading factor of less than 1 is used to leave space for subsequent insertions so that overflow accesses are minimized.

The effect of varying the initial loading factor on the performance of an indexed sequential file is shown in Fig. 13. In this particular case, 1500 records of REC-C are loaded initially and it is expected that the number of records would grow to 2000 eventually. The graphs are plotted for both the number of primary block accesses and overflow block accesses versus the increase in initial loading factor from 0.25 to 0.95 in steps of 0.1. The resultant curves behave as expected. The curves for the overflow block accesses in all cases rise for increasing blocking factor. The curves for primary data block accesses decrease for increasing blocking factor. Both results are a direct consequence of the fact that increasing the initial loading factor results in a decrease in the number of data nodes. The increase in the overflow block accesses for retrieval of some records can be explained in the same manner as that of increasing block size in the last example. However, the increase in the overflow block accesses for retrieval of unique or retrieval of all records is unlike that of the previous example. The reason is that, although the number of nodes decreases, there is no corresponding increase in the block size of the primary data block (like the last example). Therefore, the longer overflow chain will cause more overflow block accesses in both types of retrieval.

One would expect that 75 percent would be the optimal loading factor for this given file because the records at the initial load constitute 75 percent of those that would eventually be stored in the file. This may be true for an ideal case where all the records are distributed evenly. In reality, this is not the case. Some data nodes will have more records mapped to them that the other nodes. Consequently, overflow would result. This can be avoided by using a lower loading factor, thereby ensuring that there is little or no chance of overflow. The curves in the graphs indicate that a value of 0.65 would be most appropriate for this example and will result in minimum data and overflow block accesses.

For the next two examples, it is assumed that REC-B,

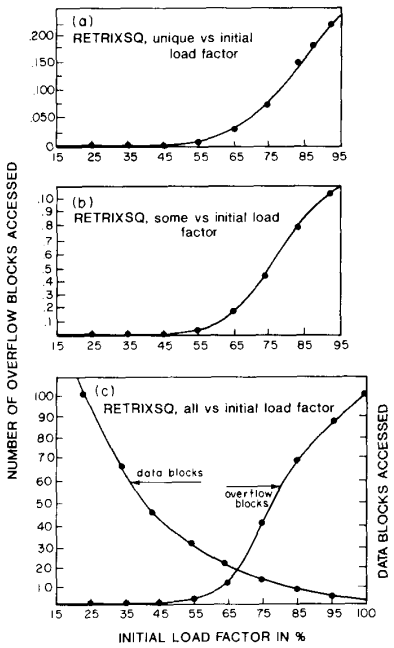


Fig. 13. RETRIXSQ versus initial load factor.

REC-E, and REC-F are stored in the same area. The location mode of REC-A is assumed to be indexed sequential and that of REC-E and REC-F is assumed to be via-set. The set mode of both SET-BE and SET-EF is assumed to be CHAIN. The first example is used to show the effect of different kinds of chain pointers on the database performance. In particular, the chain pointer type of the set SET-BE is to be identified as FORWARD ONLY, FORWARD and BACKWARD, or FORWARD and OWNER. Transaction T3 is used in this example.

TRANSACTION (T3)
RETROWNR, REC-B, SET-BE, CHAIN

The results are shown in Fig. 14. As is evident from the steep decline of the curve, the number of member blocks accessed is less in the case where the FORWARD and BACKWARD pointer is used instead of FORWARD pointer alone. Obviously, no member-only block access is required when OWNER pointers are employed. However, these pointers take up extra storage and should be used only when access time is important.

In the second example, the performance of the above mentioned file structure is studied for varying block sizes. Transaction T4 given below is used. Also, in this example, the chain pointer type is defined to be FORWARD ONLY.

TRANSACTION T(4)
RETRIEVE, REC, SET-BE, CHAIN, UNIQUE, FOUND
RETRIEVE, RECE, SET-BE, CHAIN, SOME
RETRIEVE, RECE, SET-BE, CHAIN, ALL

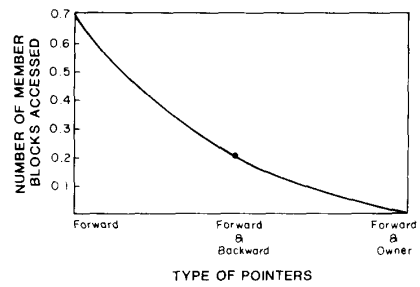


Fig. 14. RETRMEMB versus type of pointers in chain set.

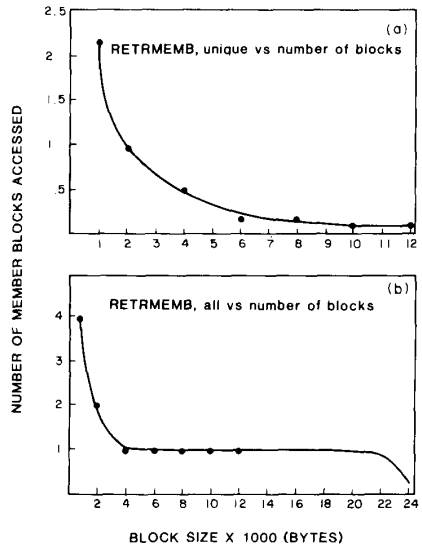


Fig. 15. RETRMEMB versus number of blocks.

The results are summarized in Fig. 15. The curves for both cases, retrieval unique and retrieval all, decrease as the block size increases. In fact, the curve attains a zero value if the block size is increased to an extent that all the member records fit into the owner-member blocks and there are no member-only blocks.

B. A Design Example

In this section, we illustrate the manner in which the evaluation system can be used in the physical database design process given a logical database design and a set of representative transactions. As an example, the logical database design shown in Fig. 9 and the following set of representative transactions are used.

TRANSACTIONS

T(1) = 195* (RETRIEVE, REC-A, UNIQUE, KEY, FOUND);

T(2) = 1* (RETRIEVE, REC-A, UNIQUE, KEY, FOUND
+
RETRMEMB, REC-D, SET-AD, CHAIN, UNIQUE, FOUND);

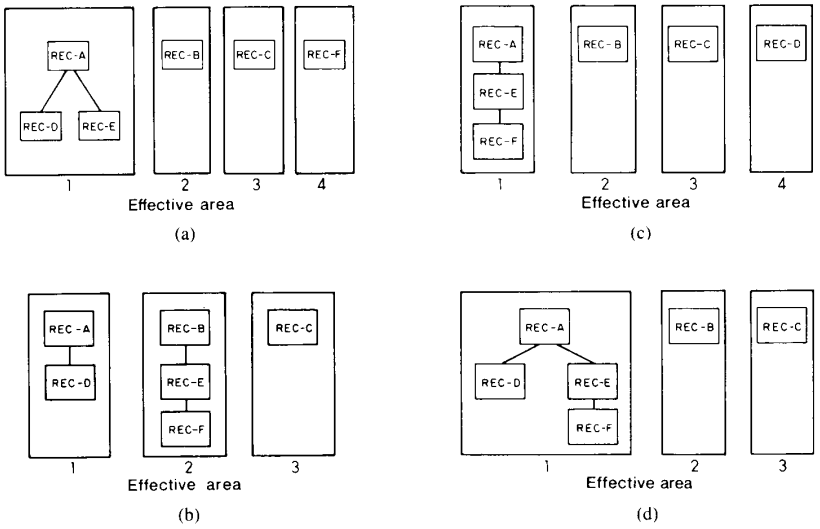


Fig. 16. Some design examples. (a) Design alternative 1. (b) Design alternative 2. (c) Design alternative 3. (d) Design alternative 4.

TABLE III
NUMBER OF BLOCKS ACCESSED

Run ID	Index	Data	Overflow	Pointer Array	Owner Member	Member Only
(a)	820	618	213.12	0	1	1668.80
(b)	1130	618	156.39	0	1	301.33
(c)	820	618	213.12	0	1	1697.33
(d)	820	618	519.39	0	1	1679.42

```
T(3) = 1* (RETRIEVE, REC-A, UNIQUE,
      KEY, FOUND
         +
       RETRMEMB, REC-E, SET-AE, CHAIN, ALL
         +
       RETRMEMB, REC-F, SET-EF, CHAIN,
       UNIQUE, FOUND);
T(4) = 10* (RETRIEVE, REC-A, UNIQUE,
      KEY, FOUND
         +
       RETRMEMB, REC-E, SET-AE,
       CHAIN, ALL);
T(5) = 1* (RETRIEVE, REC-A, UNIQUE,
      KEY, FOUND
         +
       RETRMEMB, REC-E, SET-AE,
       CHAIN, ALL
         +
       RETROWNR, REC-B, SET-BE, FORWARD);
T(6) = 10* (RETRIEVE, REC-B, UNIQUE,
      KEY, FOUND);
T(7) = 300* (RETRIEVE, REC-B, UNIQUE,
      KEY, FOUND
         +
       RETRMEMB, REC-E, SET-BE, CHAIN,
       UNIQUE, FOUND);
T(8) = 100* (RETRIEVE, REC-C, UNIQUE,
      KEY, FOUND);
```

In this design, some design choices can be quite straightforward and can be done without the aid of an evaluation system. For example, it is obvious in this sample design that REC-C should be stored in an area by itself since it is not used in conjunction with any other record. However, some other design choices may not be so obvious. For example, it is unclear in this sample design whether REC-E should be stored in the same area as REC-A or REC-B. It is for these types of design choices that the evaluation system can be very helpful.

In this example, let us assume that we have enough information to complete the physical database design up to the point as follows. The location mode of REC-A is CALC (hash-based) and the location mode of REC-B and REC-C and INDEXED SEQUENTIAL. The block sizes are fixed at 12 000 bytes. Furthermore, other factors such as initial loading factor, number of data nodes, etc., are also fixed. The physical database design parameter under consideration is the clustering strategy for the record types.

Examining the transactions, we see that most of the transactions involve the retrieval of REC-A with either REC-D or REC-E. It appears that the records REC-A,

REC-D, and REC-E should be clustered together. Also, REC-B, REC-C, and REC-F should be stored separately in different areas, since they are not used in conjunction with any other record type. The resulting clustering strategy is shown in Fig. 16(a). The evaluation output corresponding to this design is shown in row (a) of Table III.

We then tried various clustering strategies shown in Figs. 16(b), (c), and (d) and compared the outputs. The results are summarized in Table III, rows (b), (c), and (d), respectively.

The intuitive design we arrived at proved to be better than the two other designs, except for the case in Fig. 16(b), in which records REC-B, REC-E, and REC-F are stored together in one area, REC-A and REC-D in another area, and REC-C in an area all by itself. If we examine the transactions closely, we see that the transaction involving the retrieval of REC-B in conjunction with REC-E is a very high frequency transaction and outweighs the other transactions which involve the retrieval of REC-E in conjunction with other record types.

V. Summary and Conclusion

The design and implementation of a physical database design evaluation system for CODASYL databases has been described. The evaluation system has been implemented in UCSD Pascal running on IBM PC's. It is presently being used by the Navy Ships Supply Control Center and the National Bureau of Standards in their database design efforts. The system is composed of three main modules: a user interface, a transaction analyzer, and the core module. The user interface allows a design to interactively enter information concerning a database design which is to be evaluated. This information includes the values for relevant DDL parameters and estimated values for each record type, set type, and area. The output of the user interface is the DD dictionary. The transaction analyzer allows the user to specify the processing requirements in terms of logical transactions and translates these logical transactions into physical transactions which access and manipulate the physical database. The core module is the implementation of a set of analytical models. In this work, models and cost formulas have been developed for indexed sequential, hash-based and CODASYL set file and database structures. The cost formulas for indexed sequential and hash-based file structures are extensions of some existing analytical models. They account for the situation in which occurrences of multiple record types are stored in the same area. The concepts of effective area and effective blocking factors were introduced to determine what fraction of each data block or overflow block is occupied by the object record type. Also, an analytical model has been developed for the CODASYL set structure. The cost for each physical transaction (the output of the transaction analyzer) is determined based on the data objects that the transaction references, their physical structures (DD dictionary information) and the operations specified.

A series of experiments has been carried out by using the implemented system to determine the effects of varying different design parameters. Some results were given in Section IV. A design example was also given to demonstrate the manner in which the evaluation system can be used to aid a designer in making design decisions. We believe that, as the magnitude and complexity of a design task increase, a database design tool such as the one presented here would become not only useful but essential.

Our current effort is directed toward the enhancement of the current system and the development of the evaluation system to tailor it to some specific commercial DBMS's. One enhancement would be in making the evaluation system a more active system. In other words, in addition to evaluation designs, the system would also actively propose good designs. For example, design heuristics can be included to incorporate design knowledge to narrow the design problem space. Then the evaluation system would automatically generate the full range of test values so that the top n designs are presented to the designer. After the best design is selected, automatic generation of the resultant schema definition for a specific DBMS can be performed in a straightforward manner.

Finally, a comprehensive database design system can be realized through the integration of a logical database design system with a physical database design system. The logical database design system can be used to narrow the design problem space to a manageable size using logical design considerations. Then the physical database design system using design information from the logical design can be used to make the final selection and to map the design to a specific DBMS.

References

[1] H. D. Anderson and P. B. Berra, "Minimum cost selection of secondary indexes for formatted files," *ACM Trans. Database Syst.*, vol. 2, no. 1, pp. 68-90, Mar. 1977.

[2] D. S. Batory, "On searching transposed files," *ACM Trans. Database Syst.*, vol. 4, no. 4, pp. 531-544, Dec. 1979.

[3] —, "B+ trees and indexed sequential files: A performance comparison," in *Proc. ACM/SIGMOD 1981 Int. Conf. Management of Data*, Ann Arbor, MI, ACM, 1981, pp. 30-39.

[4] —, "Optimal file designs and reorganization points," *ACM Trans. Database Syst.*, vol. 7, no. 1, pp. 60-81, Mar. 1982.

[5] —, "A model of transactions on physical databases," *ACM Trans. Database Syst.*, vol. 7, no. 4, pp. 509-539, Dec. 1982.

[6] E. Berelian and K. B. Irani, "Evaluation and optimization," in *Proc. 4th Int. Conf. Very Large Data Bases*, 1978, pp. 545-555.

[7] M. W. Blasgen and K. P. Eswaren, "Storage and access in relational databases," *IBM Syst. J.*, vol. 16, no. 4, pp. 363-377, 1977.

[8] I. Casas-Raposo, "Analytic modeling of database systems: The design of a System 2000 performance predictor," Dep. Comput. Sci., Univ. Toronto, Ont., Canada, Tech. Rep. 35, 1981.

[9] D. Comer, "The ubiquitous B-tree," *Comput. Surveys*, vol. 11, no. 2, pp. 121-138, June 1979.

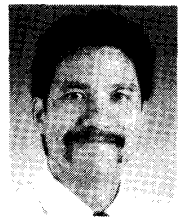
[10] T. J. Gambino and R. Gerritsen, "A data base design decision support system," in *Proc. 4th Int. Conf. Very Large Data Bases*, 1978, pp. 534-544.

[11] R. Gerritsen, "Cost effective database design: An integrated model," in *Proc. Conf. Advanced Information Systems Development Techniques Symp.*, Apr. 1977.

[12] H. Guttman, "R-trees: A dynamic index structure for spatial searching," in *Proc. ACM/SIGMOD Conf.*, 1984, pp. 47-56.

[13] J. A. Hoffer, "A clustering approach to the generation of subfiles for

the design of a computer database," Ph.D. dissertation, Cornell Univ., Ithaca, NY, 1975.
[14] R. Katz and E. Wong, "An access path model for physical database design," in *Proc. ACM/SIGMOD Conf.*, 1980, pp. 22-29.
[15] D. G. Keehn and J. O. Lacy, "VSAM data set design parameters," *IBM Syst. J.*, vol. 13, no. 3, pp. 186-212, 1974.
[16] W. F. King, "On the selection of indices for a file," IBM Res. Lab., San Jose, CA, Rep. RJ 1341, 1974.
[17] H. Lam and S. Y. W. Su, "Modeling effective area and block size for physical design of CODASYL databases," Database Systems Research and Development Center, Univ. Florida, Gainesville, Tech. Rep., 1986.
[18] P. A. Larson, "Analysis of index-sequential files with overflow chaining," *ACM Trans. Database Syst.*, vol. 6, no. 4, pp. 671-680, Dec. 1981.
[19] P. A. Larson, "Performance analysis of linear hashing with partial expansions," *ACM Trans. Database Syst.*, vol. 7, no. 4, pp. 566-587, Dec. 1982.
[20] ——, "Linear hashing with overflow-handling by linear probing," *ACM Trans. Database Syst.*, vol. 10, no. 1, pp. 90-96, Mar. 1985.
[21] S. T. March and D. G. Severance, "The determination of efficient record segmentations and blocking factors for shared data files," *ACM Trans. Database Syst.*, vol. 2, no. 3, pp. 279-296, Sept. 1977.
[22] M. F. Mitoma and K. B. Irani, "Automatic data base schema design and optimization," in *Proc. First Int. Conf. Very Large Data Bases*, 1975, pp. 286-321.
[23] B. Niamir, "Attribute partitioning in a self-adaptive relational database system," M.Sc. thesis, M.I.T., Cambridge, MA, Rep. MIT/LCS/TR-192, 1978.
[24] J. Nievergelt and H. Hinterberger, "The grid file: An adaptable symmetric multikey file structure," *ACM Trans. Database Syst.*, vol. 9, no. 1, pp. 38-71, Mar. 1984.
[25] M. Schkolnick, "The optimal selection of secondary indices for files," *Inform. Syst.*, vol. 1, pp. 141-146, 1975.
[26] P. Selinger *et al.*, "Access path selection in a relational database system," in *Proc. ACM/SIGMOD*, 1979, pp. 23-34.
[27] D. G. Severance and R. Duhne, "A practitioner's guide to addressing algorithms," *Commun. ACM*, vol. 19, no. 6, pp. 314-326, June 1976.
[28] D. G. Severance and G. M. Lohman, "Differential files: Their application to the maintenance of large databases," *ACM Trans. Database Syst.*, vol. 1, no. 3, pp. 256-267, Sept. 1976.
[29] S. Y. W. Su and H. Lam, "A design tool for the analysis and evaluation of physical databases," U.S. Dep. Navy, Bethesda, MD, Final Rep. DDT-TR-83-01, June 1983.
[30] T. J. Teorey and J. P. Fry, *Design of Database Structures*. Englewood Cliffs, NJ: Prentice-Hall, 1982.
[31] J. A. Van der Pool, "Optimum storage allocation for a file in steady state," *IBM J. Res. Develop.*, pp. 27-38, 1973.
[32] E. Veklerov, "Analysis of dynamic hashing with deferred splitting," *ACM Trans. Database Syst.*, vol. 10, no. 1, pp. 90-96, Mar. 1985.
[33] S. B. Yao, "Approximating block accesses in database organizations," *Commun. ACM*, vol. 20, no. 4, pp. 260-261, Apr. 1977.
[34] S. B. Yao, "Optimization of query evaluation algorithms," *ACM Trans. Database Syst.*, vol. 4, no. 2, pp. 133-155, June 1979.



Herman Lam received the B.S. degree from the Georgia Institute of Technology, Atlanta, in 1972, and the M.E. and Ph.D. degrees in electrical engineering from the University of Florida, Gainesville, in 1974 and 1979, respectively.

Currently, he is an Associate Professor in Electrical Engineering at the University of Florida and a member of the research staff of the Database Systems Research and Development Center at the University of Florida. His major software interests include object-oriented database systems, database performance evaluation, and database management in CAD/CAM. His major hardware interests include computer architecture for nonnumerical processing and custom/semicustom VLSI design.

Stanley Y. W. Su received the M.S. and Ph.D. degrees in computer science from the University of Wisconsin, Madison, in 1965 and 1968, respectively.

He is a Professor of Computer and Information Sciences and of Electrical Engineering, and is the Director of the Database Systems Research and Development Center, University of Florida, Gainesville.

He was one of the founding members of the IEEE Computer Society Technical Committee on Database Engineering. He served as the Co-chairman of the Second Workshop on Computer Architecture for Non-numeric Processing (1976), the Program Chairman and organizer of the Workshop on Database Management in Taiwan (1977), the U.S. Conference Chairman of the Fifth VLDB Conference (1979), and the General Chairman of the ACM's SIGMOD International Conference on Management of Data (1982). He is an Editor of the *International Journal on Computer Languages*, *Information Sciences: An Internationl Journal*, and an Area Editor for the *Journal on Parallel and Distributed Computing*.

Nageshwar R. Koganti was born in Guntur, India. He received the Bachelor's degree in electronics and communication engineering from the University of Madras, India, in 1982. Since then he was a Research and Teaching Assistant in the Department of Electrical Engineering at the University of Florida, Gainesville. He received his M.E. degree from the University of Florida in 1984.

He presently works for Applied Data Research, Dallas, TX.