Database Reorganization—Principles and Practice

GARY H. SOCKUT

Institute for Computer Sciences and Technology, National Bureau of Standards, Washington, D.C. 20234

ROBERT P. GOLDBERG

BGS Systems, Inc., Box 128, Lincoln, Massachusetts 01773

Database reorganization can be defined as changing some aspect of the way in which a database is arranged logically and/or physically. An example is changing from a one-to-one to a one-to-many relationship. Reorganization is a necessary function in a database system. This paper introduces the basic concepts of reorganization, including why it is performed. Many types of reorganization are described and classified into logical/physical levels. Then pragmatic issues such as reorganization strategies, a survey of several commercial reorganization facilities, case studies, and database administration considerations are covered. Finally, several research efforts are surveyed.

Keywords and Phrases: database, database management, reorganization, restructuring, file maintenance

CR Categories: 2.43, 3.51, 3.73, 4.33, 4 34

INTRODUCTION

We define database *reorganization* as changing some aspect of the way in which a database is arranged logically and/or physically. We use *reorganization* as a generic term that covers what some authors call *restructuring* (changing *logical* structures) and *reformatting* (changing *physical* structures). Some examples of reorganization are adding an attribute, changing from a one-to-one to a one-to-many relationship, deleting a secondary index, changing from sequential to pointer linkages, changing from hashed to indexed access, and eliminating overflow in an indexed sequential access method.

The intended audience for this paper includes data processing operations personnel, database researchers, and students. The paper should be comprehensible to a reader with a basic knowledge of databases, such as that acquired from taking a one-semester university course or from reading sources like DATE77, MART77, or SIBL76. The reader should be familiar with such terms as *database, database management system (DBMS), schema, subschema, record, CODASYL* (or *DBTG*) *set,* and *access method.*

Reorganization may be performed for a variety of reasons. It may be highly desirable (e.g., to improve performance, storage utilization, or human productivity), or it may be necessary (e.g., to change security policies, to create a new database from old databases, or to improve functional capabilities). The following are some circumstances under which reorganization is appropriate:

- The definition of the information changes. For example, if a company initially requires each of its employees to work on only one project at any time but later changes its policy to allow an employee to work on several projects simultaneously, then the one-to-many relationship between projects and employees

CONTENTS

must change to a many-to-many relationship.

- A new type of information is added to a database. This may require increasing the size of a record type to accommodate a new field.
- New legislation requires a change. For example, restricting disclosure of information among government agencies may require splitting records into disclosable and nondisclosable parts.
- A new database is created from one or more old databases or files. For example, a company that acquires another company may merge the two customer databases, which may be associated with different DBMSs and which may be in different forms, thus requiring conversion.
- Empirical characteristics of the information change. For example, in a file of families, the optimal amount of space to reserve for children near a family record results from a time-space trade-off. As zero population growth becomes more popular, more families are small, and the optimal amount of space to reserve may decrease.
- Characteristics of usage change on either a short- or long-term basis. For example, if new sociological research using a population database requires access via a particular key, a new secondary index might be added.
- As the amount of information grows, a database may be moved to larger or faster storage devices. This may require modifying the mapping of records to physical locations.
- Poor performance may lead to "tuning" or redesigning such aspects as hashing parameters.
- The optimal access arrangement varies with time, as in an airline reservation system, where flight information for the next few days may be finely indexed for quick access, while information for later days may be coarsely indexed to save space. At the end of a day, that day's information is archived, and one more day's information is indexed finely.
- Performance can degrade during normal operation as unpredictable insertions are made or deleted records accumulate. For example, if insertions into an indexed sequential file are clustered in one key range, access times for that part of the file increase as overflow structures accumulate. Maintenance may produce a more balanced structure yielding better access times.

A DBMS may select and maintain some storage structures automatically. However, even for such a system, those storage structures, as well as logical structures, may require manual or automatic reorganization at times.

The balance of this paper contains tutorials and surveys on several aspects of reorganization. In Section 1 we describe many types of reorganization and classify them into logical/physical levels. In Section 2 we discuss pragmatic issues. These include 1) strategies used to reorganize, 2) a survey of reorganization facilities provided with several well-known commercial DBMSs,[1] 3) case studies, and 4) database administra-

[1] In several places in this paper we describe selected commercial systems The inclusion or exclusion of a system and the order and content of our descriptions do not imply endorsement or disapproval by the authors or their organizations. We do not certify that the systems operate as described.

tion considerations. We then survey several research efforts in database reorganization in Section 3.

1. TYPES OF REORGANIZATION

In this section we describe several types of reorganization, which are classified using a database accessing model. As explained below, the accessing model describes database constructs at each of several levels, ranging from logical to physical. We classify types of reorganization into levels according to the constructs they change. Reorganization can occur at any level. The classification is presented primarily for pedagogic purposes, but it can also illustrate *data independence*; i.e., changes in constructs at low levels do not affect constructs at high levels, except for differences in performance. BERG80b also describes some types of changes.

1.1 Overview of the Classification

The classification uses the stratification of *DIAM II* [SENK75], the second version of the *Data Independent Accessing Model*, which was developed principally by Michael E. Senko. DIAM II consists of a *data model* and an *accessing model*. A data model is a set of logical data structure types, occurrences of which represent the logical information in a database, plus operations on those types. Examples are the CODASYL (or DBTG) [CODA71, CODA78], relational [CODD70], IMS [IBM77a], and entity-relationship [CHEN76, CHEN78] data models, as well as the many data models described in KERS76. An accessing model is a description of how data are physically stored and accessed.

The classification is useful for all data models, not just DIAM's, since it uses *only* DIAM's accessing model, not its data model. Most of the reorganization examples used in this paper come from CODASYL, IMS, and relational systems. SCHN76 uses DIAM I [ASTR72, SENK73], an earlier version of DIAM, to describe relational systems.

We selected DIAM because it provides a stratification of database constructs and it is well known. The stratification used by the CODASYL Stored-Data Definition and Translation Task Group [CODA77] was also largely based on DIAM's stratification. The classification is general enough to apply to all of the database structures that we examined, but there could be structures requiring other classifications.

Below we describe briefly the levels of DIAM II. The description is not an in-depth one, but it provides sufficient criteria for classifying types of reorganization. The references contain more detailed explanations of the DIAM II levels.

Figure 1 shows an overview of DIAM II. Logical levels appear at the top, and physical levels appear at the bottom. Rectangles represent *constructs* at the various levels, while ovals represent *interlevel mappings* between constructs. The five DIAM II levels are labeled at the left, while the three levels of the *ANSI SPARC* database architecture (a proposed architectural framework for database systems) [TSIC77] are labeled at the right. Senko has pointed out a correspondence between ANSI SPARC and DIAM II levels [SENK75]: The ANSI SPARC *external schema* corresponds to the DIAM II *end-user level*, the *conceptual schema* corresponds to the *infological level*, and the *internal schema* corresponds to the *string level*, *encoding level*, and *physical device level*. Some database terms in common use (*subschema*, *schema*, and *access methods*) that correspond roughly to DIAM II concepts also appear in the figure.

We first describe the *infological level* [SENK75], which defines attributes and relationships among them. It also defines attributes' logical representations (e.g., range of values). This level corresponds roughly to the common notion of *schema*, although schemata in existing DBMSs generally contain information from lower DIAM II levels as well.

The *end-user level* [SENK75] provides users (or applications) with views of the infological level's constructs. A user may view a subset of the database's attributes and relationships, and this subset may have a certain structure (e.g., a hierarchy). This level corresponds roughly to the common notion of *subschema*.

The *string level* [SENK75] defines access paths, which often implement relation-

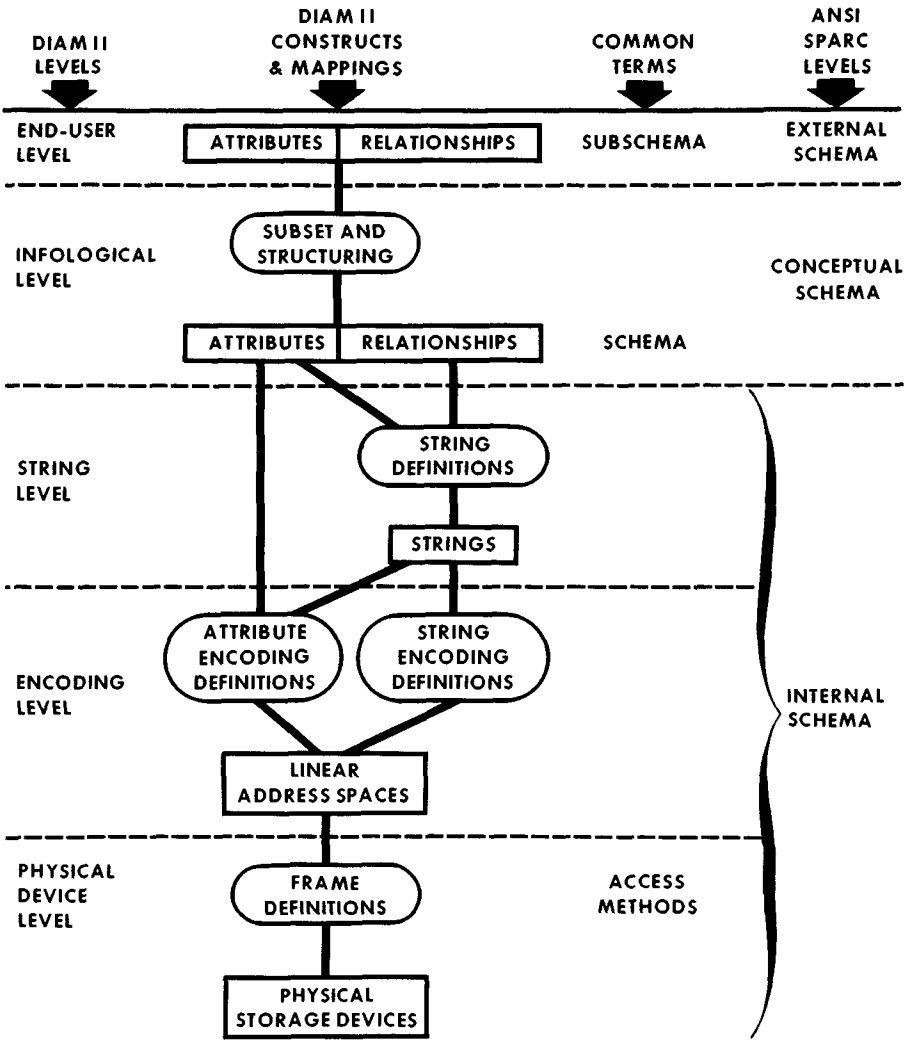


FIGURE 1. Overview of DIAM II.

ships. *Strings* are linkages among attributes, e.g., a CODASYL record type, which links fields (data items)—or linkages among strings, e.g., a CODASYL set, which links an owner record type and one or more member record types. A string can order the constructs (attributes or strings) that it links. Several string representations may be possible for a given relationship. For example, a one-to-many relationship might be implemented in CODASYL as a set or as a repeating group of fields. Strings are also used in the implementation of *secondary* indices.

The *encoding level* [ALTM72] defines how strings and attributes are physically represented (encoded) as bits in one-dimensional bit streams (called *linear address spaces*). Strings can be represented by physical contiguity (as is usually true for a record) or by pointers (as is usually true for a set). Examples of attribute encoding are character codes and binary or decimal integers.

The *physical device level* [SENK76] maps linear address spaces onto physical storage devices, using constructs called *frames* as an intermediate stage. A frame is a generalization of such physical units as block, track, cylinder, hash bucket, and disk pack.

Frame definitions correspond roughly to *access methods.*

Sections 1.2 through 1.5 classify many types of reorganization within the levels of DIAM II. We use subdivisions within a level for pedagogic purposes, not for finer stratification. We do not list all possible types of reorganization. Most examples come from CODASYL, IMS, and relational systems because these systems are well known. The terms and constructs we use are described in depth in CODA71, CODA78, and TAYL76 for CODASYL, in IBM77a and TSIC76 for IMS, in CODD70 and CHAM76 for relational systems, and in DATE77 and MART77 for all three. A type of reorganization in an existing DBMS may correspond to more than one DIAM level.

1.2 Reorganization at the Infological Level

In this section we discuss changes in definitions of attributes and relationships. The discussion applies to any data model. CHEN77 describes a similar collection of changes for the entity-relationship data model.

In cases in which unchanged subschemata hide infological level changes from application programs, old application programs are not changed. If a subschema is changed, the change affects application programs that use the subschema. Application program conversion, which this paper does not cover, can cost more than data conversion [HOUS77, TAYL79]. We do not address end-user level changes separately from infological level changes. In the following we describe changes at the infological level:

Attributes can be added, deleted, combined, split, or renamed. For example, in a relational database, a relation's *degree* (number of attributes) may change.

Attributes' logical representations can change, and the changes may affect field and record sizes, which are specified at the encoding level. These changes include scale (e.g., inches versus centimeters; monthly versus semimonthly salary) and range of values (including logical field size). Alternatively, one might perform some such changes at the encoding level in order to insulate the infological level from the changes.

Security controls can be changed [HSIA78]. Such changes can be implemented at lower levels.

Relationships can change, although such changes are implemented at the string level, as described in Section 1.3.1. These changes include

1) creating, destroying, or renaming a relationship;
2) changing among a one-to-one, a one-to-many, and a many-to-many relationship;
3) in a one-to-many relationship, moving an attribute between the one and the many.

1.3 Reorganization at the String Level

This section's examples involve changes in the string definitions. The examples are divided into 1) implementing changes in relationships, 2) reimplementing unchanged relationships, and 3) performing other changes at the string level.

1.3.1 *Implementing Changes in Relationships*

The three relationship changes listed at the end of Section 1.2 can be implemented, respectively, by the string level operations described in the following:

1) Create, destroy, or rename a string (e.g., CODASYL record, IMS segment, CODASYL set, or IMS hierarchy).
2) Change the database description among two groups of fields in a single record type, two record types related as parent (OWNER in CODASYL) and child (MEMBER in CODASYL), and three record types related as parent, child, and parent. Figure 2 uses three *data structure diagrams* [BACH69] to illustrate these changes for the CODASYL data model. The diagrams represent the changing relationships between a company's projects and its employees. Similar changes could be illustrated in the relational data model by using one, two, and three relations, respectively.

a) In the first diagram each project employs exactly one employee, who works on only that one project. There is a one-to-one relationship between projects and employees, represented by two groups of fields in a single record type.
b) In the second diagram each project employs any number of employees, each of

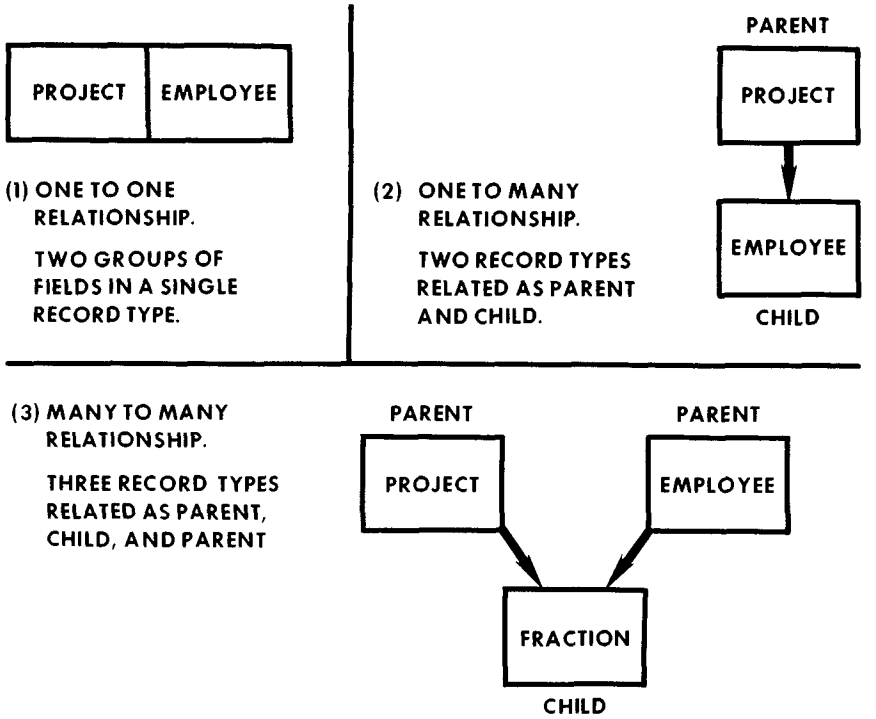


FIGURE 2 Data structure diagrams to illustrate implementing one-to-one, one-to-many, and many-to-many relationships.

whom works on only that one project. There is a one-to-many relationship between projects and employees, represented by two record types related as parent and child.

c) In the third diagram each project employs any number of employees, each of whom works on any number of projects. There is a many-to-many relationship between projects and employees, represented by three record types related as parent, child, and parent, where the child record indicates the fraction of a given employee's time that is spent on a given project.

3) Move a field between parent and child, as shown by two data structure diagrams in Figure 3. The diagrams represent the relationships among projects, employees, and phone numbers. In both diagrams there is a one-to-many relationship between projects and employees, represented by project in a parent record type and employee in a child record type.

a) In the first diagram all the employees on a project have the same phone number, since the projects operate on low budgets. The attribute PHONE NUMBER is associated with a project and is represented as a field in the parent record type. In the relational model it would be an attribute in a project relation.

b) In the second diagram the projects are well funded, and each employee may have his or her own phone number. The attribute PHONE NUMBER is represented as a field in the child record type. In the relational model it would be an attribute in an employee relation.

1.3.2 *Reimplementing Unchanged Relationships*

Unchanged relationships can be reimplemented with different strings. Below are three types of reimplementation:

1) Change the implementation of a one-

to-many relationship. For example, Figure 4 shows diagrams of record *occurrences* (*not* data structure diagrams) for three possible CODASYL implementations of a one-to-many relationship between projects and employees. A tutorial writing project employs Sockut and Goldberg, while a declaration writing project employs Jefferson.

a) In the first diagram there is a single record type which contains a nonrepeating field (or group of fields) for project

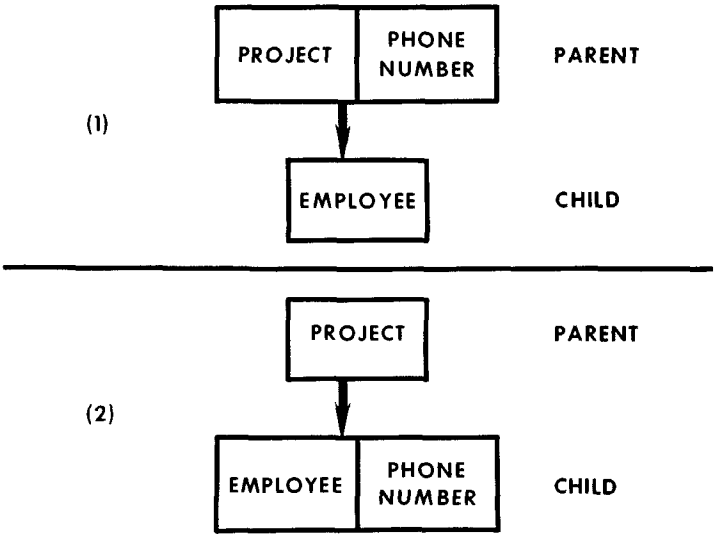
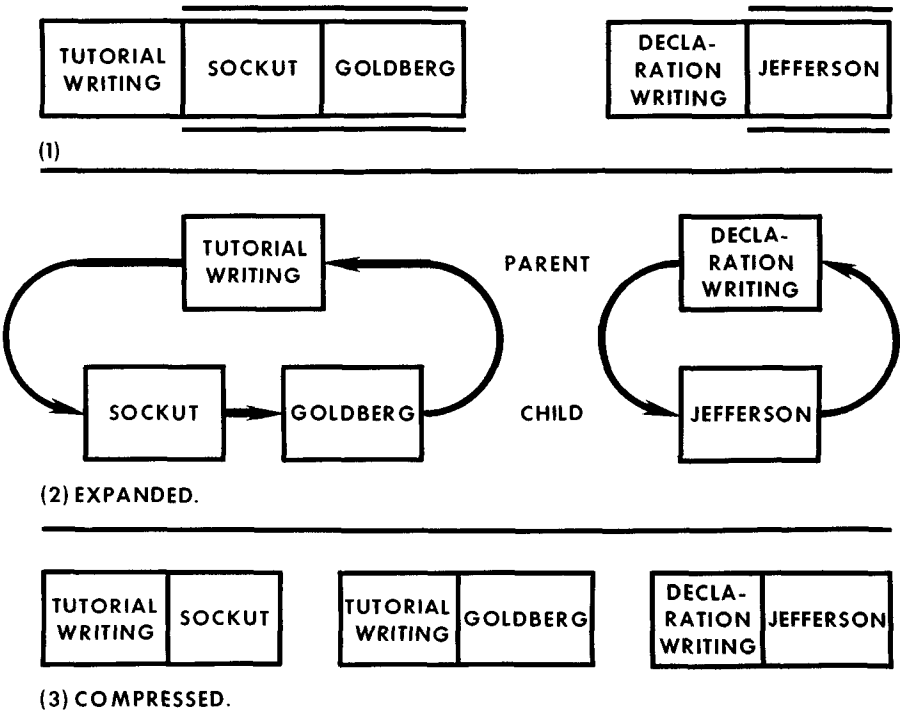


FIGURE 3. Data structure diagrams to illustrate moving a field between parent and child.

FIGURE 4. Occurrence diagrams to illustrate implementing a one-to-many relationship.



and a repeating field (or group of fields) for employee. There is one record occurrence for each project. This corresponds to an *unnormalized* relation [CHAM76] in the relational model.

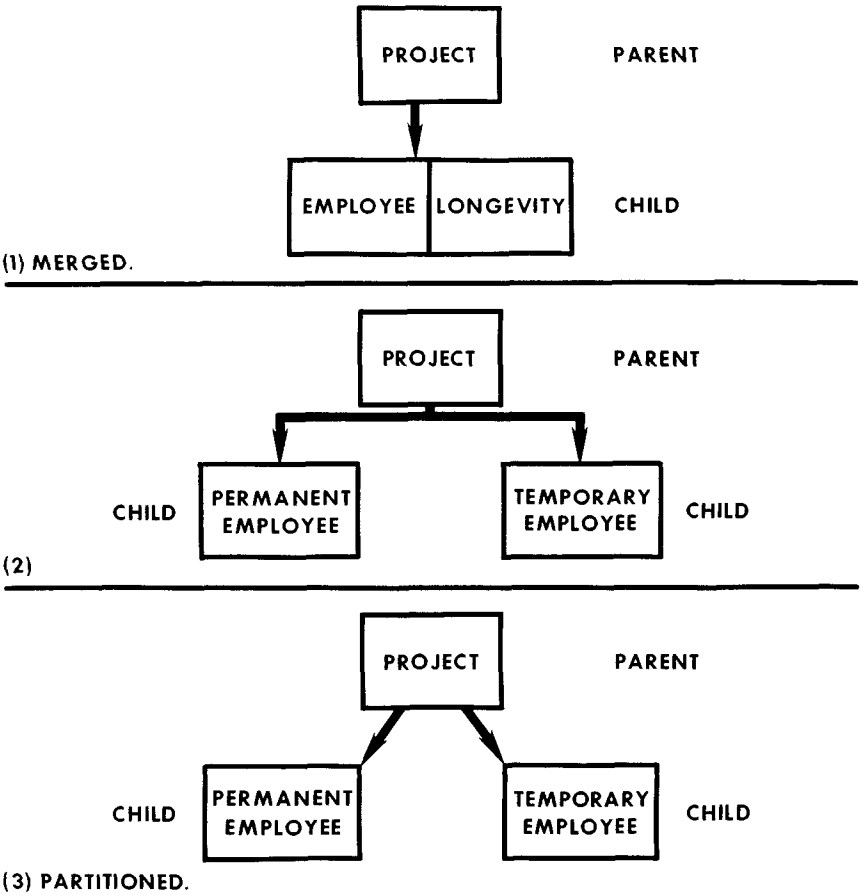
b) In the second diagram there are two record types, where the parent contains a nonrepeating field for project and the child contains a nonrepeating field for employee. There is one parent record occurrence for each project and one child record occurrence for each employee. The University of Michigan's Data Translation Project calls this arrangement *expanded* [NAVA76]. In the relational model the second and third diagrams correspond to relations that are in *first* (or higher) *normal form.*

c) In the third diagram there is a single record type which contains a nonrepeating field for project and a nonrepeating field for employee. There is one record occurrence for each employee. The Data Translation Project calls this arrangement *compressed.*

Changing from embedded attribute values to pointers to attribute values is similar to changing from compressed to expanded arrangements, where an OWNER pointer is used in the expanded case to point to the attribute value.

2) Change the implementation of a one-to-many relationship involving two (or more) types of descendants. For example, Figure 5 shows data structure diagrams for three possible CODASYL implementations of a one-to-many relationship between projects and employees where we wish to distinguish permanent from temporary em-

FIGURE 5. Data structure diagrams to illustrate distinguishing two types of descendants in a one-to-many relationship.



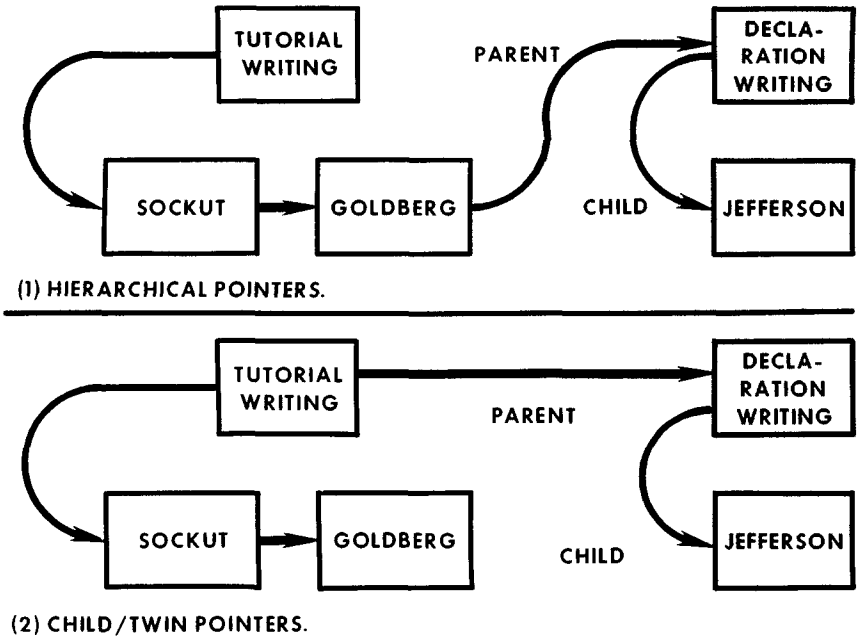


FIGURE 6. Hierarchical versus child/twin pointers in an IMS hierarchy occurrence.

ployees. In all three diagrams there is one parent record type.

a) In the first diagram there is one child record type, which contains a field (LONGEVITY) that indicates permanent or temporary. The Data Translation Project and the entity-relationship model call this arrangement *merged*. In the relational model LONGEVITY would be an attribute in an employee relation.
b) In the second diagram two child record types (one for permanent employees and one for temporary employees) are members of the same set. Similarly, the relational model would use two relations for employees.
c) In the third diagram two child record types are members of different sets. The Data Translation Project calls this arrangement *partitioned*, while the entity-relationship model calls it *split*. The relational model would again use two relations for employees.

3) Change string options associated with a one-to-many relationship. Included are

a) Changing from hierarchical pointers (which follow the depth-first hierarchical order of occurrences) to child/twin pointers (which indicate the first child and next twin) in IMS. These are shown in the occurrence diagrams in Figure 6, where we assume that PROJECT (e.g., tutorial writing) is not the root of a hierarchy.
b) Adding or deleting back pointers. Examples include adding or deleting PRIOR and LAST pointers in CODASYL, as shown in the occurrence diagram in Figure 7, and changing between two-way and one-way pointers in IMS.
c) Adding or deleting OWNER pointers in CODASYL, as shown in the occurrence diagram in Figure 7.

Since CODASYL and IMS implement 3a)-c) as pointer options, it may appear odd to classify them on the string level rather than on the encoding level. However, they change the direct access paths that can be followed in a relationship, and thus they belong at the string level.

1.3.3 *Performing Other Changes at the String Level*

A secondary index or CODASYL *singular set* (i.e., with OWNER = SYSTEM) can be

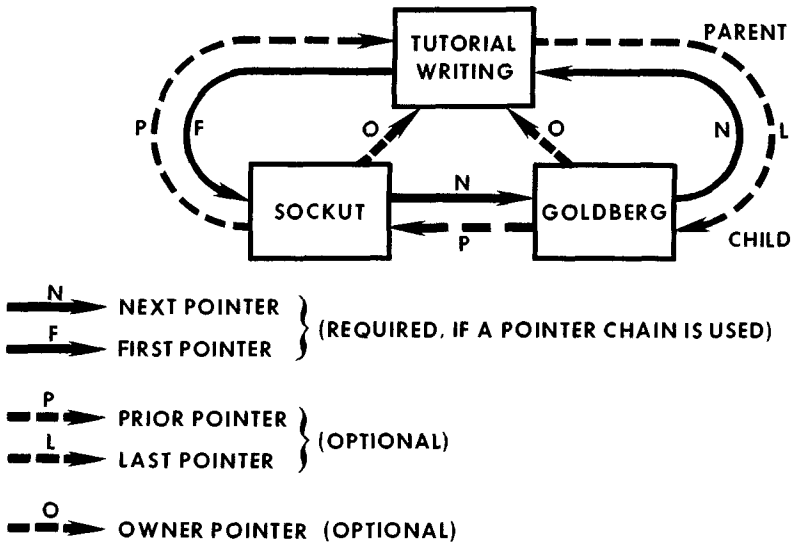


FIGURE 7. Optional pointers in a CODASYL set occurrence.

created or destroyed. Such strings serve primarily as access paths, not as implementations of relationships.

The contents of strings can be changed, for example, by adding or removing a key in a densely indexed record type, physically duplicating or not duplicating a parent's field in its children (e.g., CODASYL ACTUAL SOURCE versus VIRTUAL SOURCE), and storing a field versus calculating its value dynamically (e.g., CODASYL ACTUAL RESULT versus VIRTUAL RESULT).

Ordering within strings can be changed by reordering fields in a record type and changing the ordering strategy in a CODASYL set or IMS hierarchy. Examples of such changes in a set are changing the key(s) on which to order, changing from ascending to descending order, and changing the ordering criterion among key, time of connection into the set, and system default. Changing from ordering by time of connection to another ordering may destroy some information at the infological level.

Changed security controls can be implemented. Mechanisms include Access Control Locks in CODASYL [CODA78], Program Communication Blocks in IMS [DATE77], and access grants in System R [GRIF76], a relational DBMS.

1.4 Reorganization at the Encoding Level

In the following examples string and attribute definitions are invariant. Only their physical representations (encodings) change.

1) Changes in relationship encoding definitions include

a) Changing between sequential and direct organization in IMS. Sequential organization (HSAM or HISAM) uses contiguity to represent hierarchical relationships, while direct organization (HDAM or HIDAM) uses pointers. These four acronyms denote Hierarchical Sequential Access Method, Hierarchical Indexed Sequential Access Method, Hierarchical Direct Access Method, and Hierarchical Indexed Direct Access Method [DATE77], respectively.
b) Changing among embedded, directory, and bit map pointers [MART77].

These changes may also affect the string level if they change the direct access paths that can be followed.

2) Examples of changes in attribute encoding definitions appear below. These changes are performed at the encoding level if they are to be invisible at the infological level. A DBMS may interpret encodings dynamically.

a) Change in basic representation (e.g., "APRIL" versus "4").
b) Change in scale (e.g., inches versus centimeters; monthly versus semimonthly salary).
c) Change in character encoding (e.g., ASCII versus EBCDIC).
d) Change in integer encoding (e.g., binary, packed decimal, decimal characters).
e) Change in field size (e.g., 16-bit integers).
f) Change in length (between fixed length and variable length).
g) Change in encryption.
h) Change between data compression and noncompression [ALSB75]. Examples of data compression are eliminating leading zeros, eliminating trailing blanks, and encoding common data values. Data compression may require inclusion of a length indicator with the data.

1.5 Reorganization at the Physical Device Level

In this section the representations of attributes and relationships are invariant. Only their physical placement changes.

Changing frame definitions includes changing access keys and access methods. Examples of the latter include changing from CALC to VIA in CODASYL and from ISAM (Indexed Sequential Access Method) [IBM73] to VSAM (Virtual Storage Access Method) [IBM76], from HSAM to HISAM, or from HDAM to HIDAM in IMS.

Frame parameters can also change within an unchanged basic frame definition. These parameters include the number of levels in an index hierarchy, the presence of an index table in CALC page headers, the distribution of free space for future insertions, the overflow handling method, and hash parameters, such as hashing algorithm, bucket size, and hash width (e.g., a modulus) [MART77].

The mapping of the lowest level of frames to physical device subdivisions can be redefined. For example, CODASYL areas, VSAM Control Intervals, and VSAM Control Areas are mapped to tracks and cylinders. Also, portions of a database can move to new storage devices permanently or temporarily.

Finally, *maintenance* operations (changes to frame *occurrences*) are performed repeatedly (periodically or upon demand) to improve access time and/or storage utilization. These operations are specific to the access method. In steady state (no growth), certain access methods may perform them only rarely (or never). They are logically device independent, but performance can be device dependent; so an algorithm may be tailored to a specific device. Examples are

1) Perform a *split* in VSAM, which is an access method that supports growth in an index hierarchy by splitting data areas (and, if necessary, indices) in two when required for insertion of data. In Figure 8 the first diagram shows part of an index hierarchy occurrence in which we wish to insert an entry with key 41. The appropriate index (23-47) has no available space. Therefore VSAM splits this index in two, creates a new entry (36) in its parent index, and inserts the entry with key 41, as shown in the second diagram. If the parent index had had no available space for the new key and pointer, then the parent would have been split in two, and a new entry would have been inserted in *its* parent. Such propagation of splitting can continue to the level of the root index, which can be split if necessary (in which case the depth of the index hierarchy increases by 1).
2) Eliminate or reduce overflow, e.g., remove ISAM overflow or move a hash synonym to its home slot if the record there is deleted. Figure 9 shows an example of the latter. In the first diagram, record occurrence 1 is in its home hashing slot, while synonym records 2, 3, and 4 form a chain. After record 1 is deleted, one of the synonym records can be moved to the home slot, as shown for record 4 in the second diagram.
3) In a linked list of unallocated areas, merge two contiguous areas into one larger one [ARME70].
4) Balance an index hierarchy.
5) Compact to make space occupied by deleted records contiguous.
6) If physical order is logically unimpor-

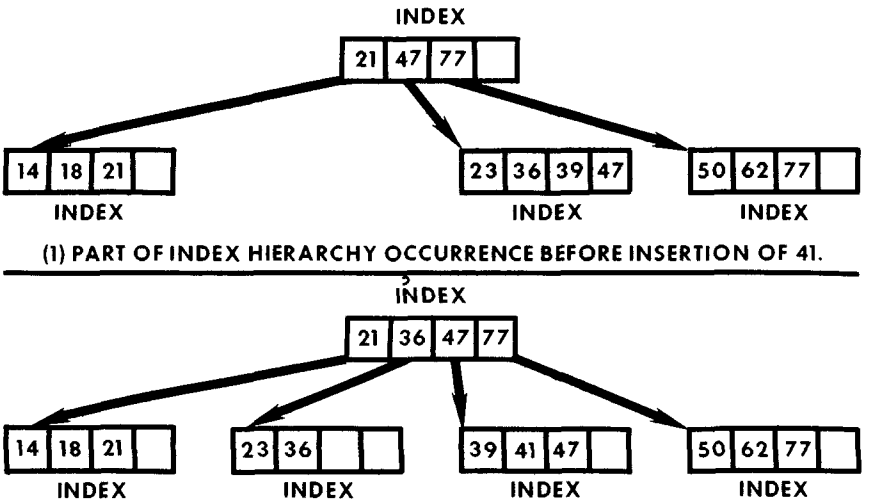
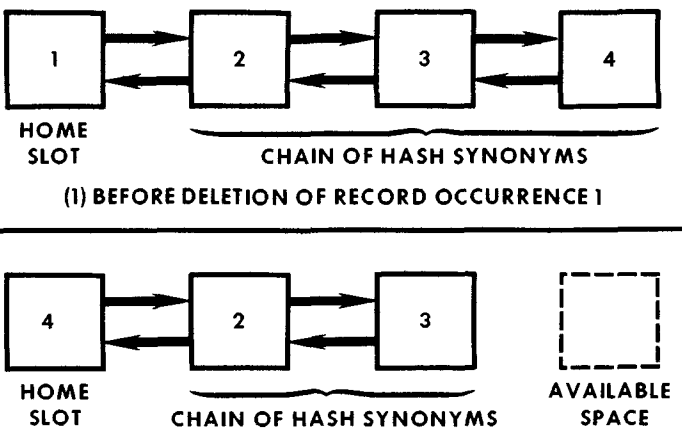


FIGURE 8 A VSAM split.

FIGURE 9 Occurrence diagram to illustrate moving a hash synonym to its home slot.



tant, move frequently accessed objects to easily accessible positions.

7) Make occurrences of physical proximity reflect occurrences of logical proximity. Examples are moving children closer to each other or to their parents and arranging a VSAM file so that the cylinder sequence reflects the key sequence (which may not be the case after a series of splits).

1.6 Other Terminology

There is no universally used set of terms for types of reorganization. Several research groups interested in conversion and logical reorganization (e.g., see SHU75, SHOS75, NAVA76) call logical reorganization *restructuring* and physical reorganization *reformatting*, and they use *reorganization* (as we do) as a generic term to cover both. For CODASYL databases, BCS75 defines *restructuring* as changing the schema, *strategy reorganization* as changing the description using the Data Storage Description Language (DSDL) [CODA78], and *physical placement reorganization* as changing the physical arrangement below the DSDL level.

Another dimension of classification is the distinction between *one-shot* and *maintenance* operations. One-shot operations (e.g., adding a secondary index) change definitions of constructs and generally are not planned to be performed repeatedly for a database with stable characteristics. Maintenance operations (e.g., eliminating overflow in an indexed sequential access method) only change occurrences, not definitions, and they *are* performed repeatedly (either periodically or upon demand). Maintenance operations appear only at the lowest (physical device) level in the classification, while there are one-shot operations at all levels. Several researchers in maintenance [SHNE73, YAO76, MARU76, TUEL78] use the term *reorganization* to denote what we call *maintenance.*

2. PRAGMATIC ISSUES

In this section we describe issues important to the implementation and management of the types of reorganization described earlier. The issues include strategies for reorganization, commercial facilities, case studies, and database administration considerations. BERG80a also describes case studies and database administration considerations for conversion.

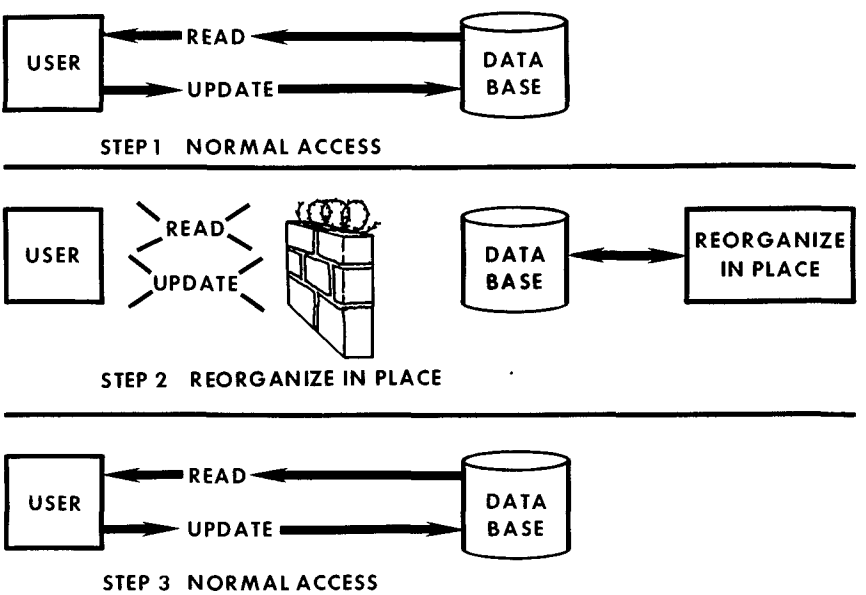
2.1 Strategies for Reorganization

We describe four strategies for reorganizing a database. Current DBMSs commonly use the first three, as shown in Section 2.2. The fourth appears to be used mainly for database unloading in some DBMSs. For the first two strategies, the database, or at least the portion to be reorganized, is usually taken off-line (i.e., is made unavailable for normal usage) overnight or over a weekend.

Strategy 1. In step 1 of reorganization in place (Figure 10), users can access the database normally. Step 2 blocks all user access (as symbolized by the wall in the figure) while reorganization is performed in place. After reorganization ends, normal access resumes, as in step 3. A variation on this strategy, which is possible in some cases, is merely to redefine the database without physically reorganizing it.

Strategy 2. Step 1 of reorganization by unloading and reloading (Figure 11) allows normal user access. Step 2 (unloading onto an unload area) and step 3 (reloading in

FIGURE 10. Reorganization in place.



reorganized format) block all user access. When reorganization ends, normal access resumes, as in step 4. A variation on this strategy is to reorganize by copying from one area to another without using an intermediate unload area.

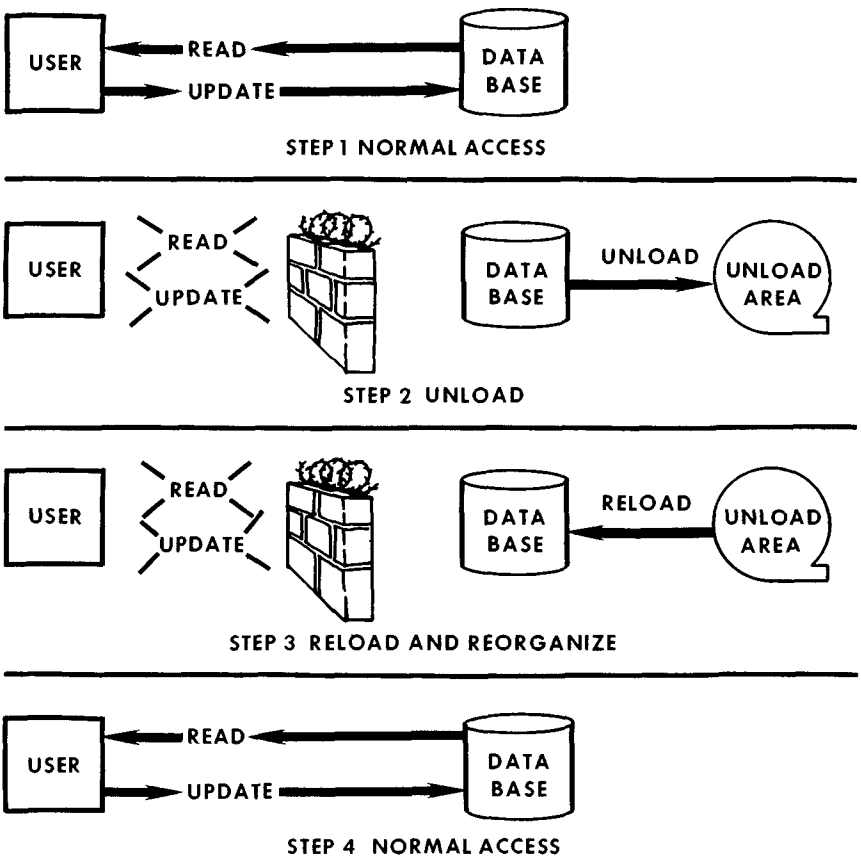
Strategy 3. A strategy that does not involve bringing the database off-line is incremental reorganization triggered by references to objects. In this strategy any needed reorganization occurs incrementally when a user references an object in the database. For example, the system may move a hash synonym to its home slot when a user deletes the record that was in the home slot.

Strategy 4. Another strategy that does not involve bringing the database off-line is reorganizing concurrently with usage of the database. Under this strategy users have access to the reorganized portion of the database while one or more processes reorganize it (in place or by unloading and reloading), as shown in Figure 12.

2.2 Commercial Facilities

Current commercial and special-purpose DBMSs provide facilities for performing various types of reorganization using the strategies described above. The number of applicable types may vary from one system to another, but every system requires some type. Below we survey functional capabilities of reorganization facilities for several well-known commercial DBMSs: ADABAS, DMS 1100, IDMS, IMS, SYSTEM 2000, and TOTAL. FRY76 lists these and other systems. For each system we survey, some types of reorganization require customer-written programs that use the normal user interface. Most of the information comes from the vendors, not from first-

FIGURE 11 Reorganization by unloading and reloading.



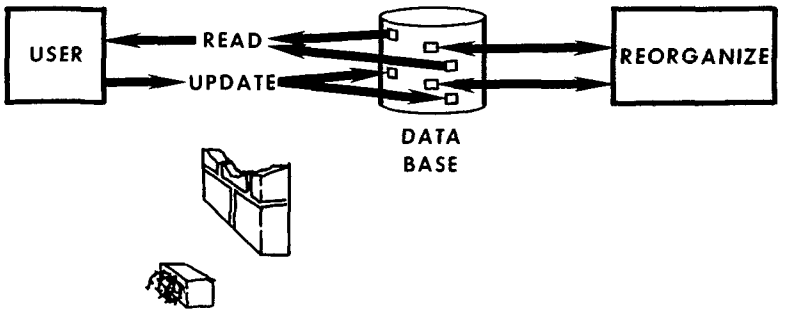


FIGURE 12 Concurrent reorganization and usage

hand experience. The survey illustrates the current state of the art in types and methods of reorganization. It does not compare the merits of the systems on the basis of their reorganization capabilities.

2.2.1 *ADABAS*

ADABAS [SOFT77] performs some types of maintenance incrementally as objects are referenced. When a record occurrence is deleted, that space within its physical block is recovered by compaction. As a file grows, filled blocks in its hierarchical index are split in two dynamically.

Database redefinition can change security controls without physical reorganization. Similarly, a field can be added (or deleted) at the end of a record type by just redefining the record type. When the system then fetches a record occurrence, it takes the field value to be null until a nonnull value is stored.

To optimize performance of sequential processing, an unload and reload utility can reorder the data in a file or portion of a file, thus making the physical order correspond to the logical order. This utility can also add or delete a field elsewhere than at the end of a record, change between data compression and noncompression, and change to or from hashed access. An unload and reload utility can balance an index hierarchy without reorganizing the data.

A utility can read a file and create a specified index, and another utility can read files and create a specified relationship by writing in a coupling structure. Most reorganization utilities lock at the file level, not at the whole database level. Certain changes in a schema, such as changing an attribute's representation or reordering fields, require customer-written programs, although a schema-subschema mapping can perform such changes in a subschema interpretively.

2.2 2 *DMS 1100*

DMS 1100 is a CODASYL DBMS. One reorganization utility [SPER78] operates by unloading and reloading specified areas. Currently available functions include adding, deleting, or renaming areas; moving a record type and its occurrences from one area to another; rebuilding access linkages (hash chains, index sequential chains, and pointer arrays); moving set occurrence members closer to each other; moving displaced record occurrences back to their home pages; and changing parameters such as hashing functions, page load factors, page sizes, the number of pages, and overflow page distributions.

A new field can be added at the end of a record type by redefining the record type, without using a reorganization utility, if the original definition specifies a large enough blank filler area at the end.

When a user program deletes a record occurrence, the system marks the space as deleted but does not make it immediately available for reuse. If there is no room on a page for a new record occurrence, then the system compacts the page in an attempt to avoid creating an overflow page. A utility can operate in place to compact each page within a specified area.

Some other types of reorganization require customer-written programs. Examples are changing an attribute's represen-

tation and changing from a one-to-one to a one-to-many relationship.

The user program library includes a DMS 1100 reorganization utility [EDEL76], which involves five steps:

1) Unload selected record tapes.
2) Delete them from the database and compact their pages.
3) Manipulate the unloaded records with a *data editor* (a processor for a programming language described below).
4) Reload the records.
5) Relink sets.

A user of this utility specifies old and new schemata and writes a data editor program that explicitly scans the old data and manipulates them from the old schema form into the new. The data editor's language includes variables, control, data manipulation facilities similar to CODASYL DML, and editing facilities similar to those of a general-purpose text editor. Some types of reorganization that the utility can perform are changing a hashing function, changing a set's ordering criterion, changing a set's pointers between a chain and an array, renaming or reordering fields in a record type, moving a record type from one area to another, changing the physical format of fields, changing from VIA to CALC, and adding or deleting record types, set types, fields, OWNER pointers, or PRIOR pointers. Difficulties in manipulating individual fields make it hard to perform some other types of reorganization, such as changing from a one-to-one to a one-to-many relationship or changing an attribute's logical representation. The data editor's language has approximately the same power as an ordinary application programming language, but it is oriented toward reorganization.

2 2.3 *IDMS*

IDMS is a CODASYL DBMS. One reorganization utility [CULL78] operates in place. Its capabilities include adding, deleting, or reordering fields in a record type; changing all occurrences of a field to a constant value; changing the length of a field; adding or deleting a set; adding or deleting optional PRIOR or OWNER pointers in a set; changing a record type between fixed and variable length; and adding or deleting a database procedure for a record type (e.g., to change between data compression and noncompression). It is also possible to execute a database procedure when the utility operates (e.g., to change an attribute's representation). After reorganization, new sets are empty until an application program explicitly connects record occurrences into them.

An unload/reload utility permits changing an area's size and changing between hashing and indexing. Another utility can change page sizes by copying without using an intermediate unload file.

When a user deletes a record occurrence that is a member of a set without PRIOR pointers, the system physically deletes the data but merely flags the occurrence's header as deleted. If the system later traverses that part of the set in update mode, it unlinks the record occurrence, checks to see if the occurrence still has links to other such sets, and physically deletes the header if it has no links. Utilities are also available to identify, unlink, and delete (physically) such flagged occurrences.

The system performs some other types of maintenance incrementally as objects are referenced: 1) When a record occurrence is physically deleted, that space within its page is recovered by compaction; 2) if all or part of a record occurrence moves off its home page when it grows, and if space later becomes available, then the occurrence can move back when it is next updated.

A new field can be added at the end of a record type by redefining the record type, without physically reorganizing it, if the original definition specified a large enough blank fill area at the end.

Some types of logical reorganization require customer-written programs. An example is changing from a one-to-one to a one-to-many relationship.

2 2.4 *IMS*

IMS provides utilities [IBM77b] that perform maintenance (recovering space and making the physical order reflect the hierarchical order) by unloading and reloading a file or a portion of a file. These utilities

can also perform some other types of reorganization if a new database description is specified for reloading. There can be no change in the hierarchical relationship of existing *segment* (record) types that remain in the reorganized database. Changes that are allowed, subject to the above restriction, include deleting a segment type (if all occurrences have already been deleted), adding a segment type, changing pointer options, changing, adding, or deleting a nonkey field in a segment type, changing a segment size, changing access methods, adding or deleting a secondary index, and changing a hash parameter. During unloading, users can read from the area being unloaded.

It is possible to add a new field at the end of a variable-length segment type by redefining the segment type without physically reorganizing it. When the system later reads a segment occurrence, the occurrence's length field indicates whether a value has been stored in the new field.

The system performs some types of maintenance incrementally as objects are referenced. Under HDAM and HIDAM (but not under HISAM), space occupied by a deleted segment occurrence is available for reuse. VSAM, which IMS often utilizes, will support a growing hierarchical index by splitting data areas and indices in two dynamically, as shown in Figure 8 and described earlier.

Customer-written programs must perform some other types of reorganization, e.g., changing an attribute's representation, changing a relationship, and reordering fields.

2.2.5 *SYSTEM 2000*

SYSTEM 2000 [MRI78] can unload records and indices in logical order and then reload in physical order; this reorganization makes the logical and physical orders the same and makes available space contiguous. Noncontiguous space may become available when record occurrences are deleted. The reload command can be directed to reload only those record occurrences that satisfy specified criteria. A record type or field is added or deleted by unloading and reloading with a new database definition, if occurrences of the record type or a descendant record type exist. If no occurrences exist, then these operations require only database redefinition, not physical reorganization.

A command is available to add or delete one or more indices without reorganizing the actual data. An index can be rearranged to make its entries contiguous without reorganizing the actual data. These operations block users from only the affected portion of the database.

Certain types of logical reorganization require customer-written programs. The user interface provides for operations upon all occurrences of a record type, e.g., halving an attribute to change from monthly to semimonthly salary.

2.2.6 *TOTAL*

TOTAL [CINC78] performs a type of maintenance incrementally when a record occurrence to be deleted is located in its home hashing slot: A hash synonym, if any (usually the physically most distant one), is moved to the home slot. When any record occurrence is deleted, the space that it (or a synonym) occupies is made available for reuse.

If a record type's original definition specified a large enough blank fill area at the end, e.g., for the purpose of adding new fields later, then a new field can be added at the end by redefining the record type, without physically reorganizing it. If there is no such fill area, or if a field is to be added somewhere else than at the end, then unload and reload utilities are used. Unload and reload utilities can also move record occurrences closer together, move a field from one record type to another, change relationships, and change a file's size (at which time the hash width may be changed). During unloading, users can read from the area being unloaded. A new record type and its relationships can be added without unloading and reloading existing record types if the old record types have been defined to include enough blank space for new pointers.

An in-place reorganization utility can change all occurrences of a field from one set of specific values to another, e.g., from "APRIL" to "4," but customer-written pro-

grams must perform more general field changes, e.g., changing from monthly to semimonthly salaries by halving all occurrences.

2.3 Case Studies

Several installations have provided statistics on the types of reorganization they perform, the strategies and facilities they use, and the amount of time and effort these operations require. One large database [DARD77] occupies 22 disk packs, each having a capacity of 200 megabytes. This installation reorganizes by unloading and reloading the entire database. Users can read (but not update) during unloading, and no access is allowed during reloading, which requires approximately 40 hours. Tape I/O errors and other factors have caused failures on several reloads, which have had to be restarted from the beginning, since at this installation reloading cannot continue if an error occurs.

Another case study involves the FBI's National Crime Information Center [BUEL77, WEIS78], which is available 24 hours per day whenever possible. It maintains up-to-date information on crimes such as car theft. It does not use a commercial DBMS, and there are no relationships among files, but there are similarities to DBMSs. Most files use ISAM [IBM73]. Every two weeks maintenance is performed in two steps:

1) Copy (and reorganize) from disk to disk. This requires at least 6 hours. Users can read or delete records (from the old files) but cannot perform any other types of updates. The system keeps track of deletions.
2) Replace the old files by the reorganized files. This requires approximately 45 minutes and blocks all user access. The system now performs in the reorganized files the deletions that it performed in the old files during step 1. Such deletions must be allowed in order to prevent situations, for example, like the following: A stolen car is recovered during a reorganization period, the owner then drives it from the police station, and shortly thereafter the police spot the car and mistakenly arrest the owner because the stolen car record has not yet been deleted.

Approximately four times per year, other types of reorganization are performed via the same copying strategy. Examples are adding new fields, changing the locations of fields, adding and deleting secondary indices, changing the access method used for secondary indices, changing the number of levels in index hierarchies, and changing the locations of master indices.

The final case study involves an airline reservation system [SIWI77], which is available 24 hours per day whenever possible. It is not a general-purpose DBMS. An installation's initial logical structure is designed carefully, and logical changes are rarely made later. At least four types of reorganization have been performed:

1) Occasionally (approximately every 2 years) a record type is added or a file's size is expanded. Reorganization involves unloading all the disks and then reloading them. During unloading, users can read from the database, and updates are saved in a special area. The system later writes the updates into the database. No user access is allowed during reloading. Unloading and reloading require 5–10 minutes each per disk pack, and an installation typically uses 2 to 200 disk packs.
2) One installation transferred its database from one set of disks to a faster set of disks and moved it geographically at the same time. This move required approximately 24 hours, during which users could read from the old area, while updates were saved in a special area. The system later wrote the updates into the new area. The careful planning and writing of the special programs that were used required several months.
3) At any time, flights for the next n days have a fine index for quick access, while flights for later days have a coarse index to save space. Every night, the system purges records for flights that have already flown and constructs fine indices for the flights for the nth day. The granularity of locking is fine enough so that this maintenance blocks a user transaction for only a few seconds. The system marks as deleted the space occupied by purged records but does not make it immediately available for reuse.

4) A second type of maintenance involves traversing the records and indicating in a storage allocation table that the space occupied by purged records is available for reuse. No compacting is performed, since all records within one area are of the same size. The system performs this type of maintenance, like the first type, concurrently with usage. It is performed at least once a week and typically requires 3 to 20 hours.

2.4 Database Administration Considerations

The *database administrator* (DBA) [MELT75, LYON76, LEON78] usually manages reorganization of databases like those described above. The DBA is the individual (or group of individuals) responsible for logical and physical definition of the database, setting security and integrity policies, monitoring performance, and, in general, supervising use of the database. The DBA determines that reorganization is necessary. The DBA must consider the following issues in connection with reorganization:

- Recognize the *need* to reorganize. The reasons for reorganization have been set forth earlier in the paper.
- Decide *what* new structures are to result from reorganization, for example, a new hash placement, balanced index hierarchies, or a new set.
- Decide *when* to perform reorganization. It may be necessary to reorganize overnight or over a weekend. For maintenance, there may be an optimal period between reorganizations.
- Know *how* to execute the reorganization. 1) Strategies may involve unloading and reloading, reorganizing in place but offline, reorganizing incrementally as objects are referenced, or reorganizing concurrently with usage. 2) The DBA must select the appropriate reorganization facilities. FRY78 and TUFT79 discuss DBA tools for reorganization and other purposes. 3) If reexecution of reorganization would take longer than recovery, then journaling facilities would be useful to recover from errors.
- Determine how much the enterprise will *benefit* from reorganization. Benefits may include improved performance, increased functional capabilities, or better storage utilization. In the case of reorganization to improve performance, performance prediction tools (see, e.g., BUZE78, DEUT78) are useful.
- Assess how much reorganization will *cost*. Costs include 1) human and computational resources consumed during planning, actual reorganization, software changes (if any), and personnel retraining (if any); 2) either the denial of resources to users during off-line reorganization or degraded user performance during concurrent reorganization.
- Be aware of *who* and *what* will be *affected* by reorganization. The first phase of at least one reorganization utility [SPER78] analyzes a proposed reorganization and lists the affected portions of the database. Another tool that often can determine effects of reorganization is a *data dictionary/directory* [LEON77], which maintains information such as what structures are in the database and which applications use them. Some applications may benefit from the reorganization, while others may suffer if the database is no longer optimized toward them. The DBA must act as arbitrator. The DBA must also see that affected software is revised and that any affected users are retrained.
- *Document* any changes that result from reorganization. The data dictionary/directory may provide some of this documentation.
- *Certify* that reorganization has yielded the desired result; for example, check to determine that new pointers correctly implement a new relationship.

3. RESEARCH EFFORTS

We have seen that many situations require reorganization and that reorganization requires time and effort. In this section we survey research efforts in three aspects of reorganization: 1) conversion, 2) maintenance, and 3) concurrent reorganization and usage. BERG80a also surveys research in conversion. Concepts that have been studied may lead to future DBMS trends.

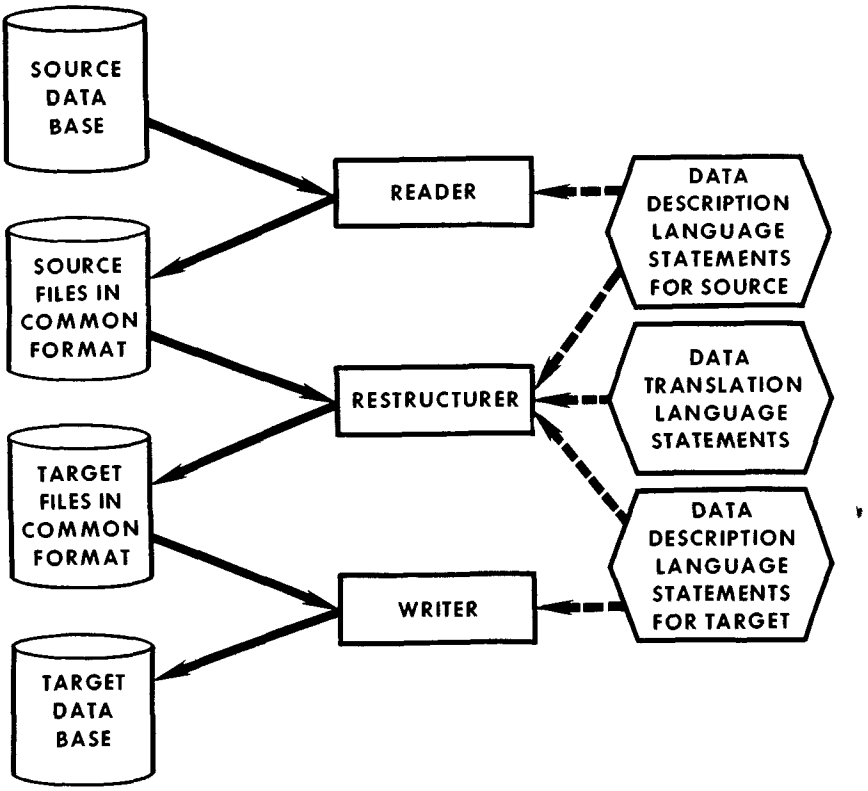


FIGURE 13. A data translation procedure.

3.1 Conversion

Several research groups, e.g., those at the University of Pennsylvania [SMIT71, RAMI74], the University of Michigan [SIBL73, FRY74, LEWI75, NAVA76, SWAR77], the University of Florida [SU74, NATI78], the IBM San Jose research laboratory [SHU75, SHU77], and System Development Corp. [SHOS75], have been developing semantics and languages for specification of logical reorganization. These languages are to be used during database conversion—e.g., changing from one DBMS's definition (a *source*) to another's (a *target*)—within a procedure such as that shown in Figure 13. The DBA defines old and new structures in a *data description language* and defines their correspondence in a *data translation* (logical reorganization) *language*. Data translation language statements indicate how source structures are mapped into target structures. A *reader*, which is driven by the data description language statements for the source, converts the source into a common format for reorganization. A *restructurer*, which is driven by the data translation language statements and by both sets of data description language statements, produces a translated target file in common format. Finally, a *writer*, which is driven by the data description language statements for the target, converts the target file into its final format. A system might compile rather than interpret the translation procedure.

3.2 Maintenance

Several research efforts (see, e.g., SHNE73, YAO76, MARU76, TUEL78) have modeled database performance deterioration, improvement through maintenance, and maintenance costs. Section 1.5 describes some types of maintenance, such as overflow removal. Figure 14 illustrates roughly how storage structure deterioration, file growth, and maintenance affect the average

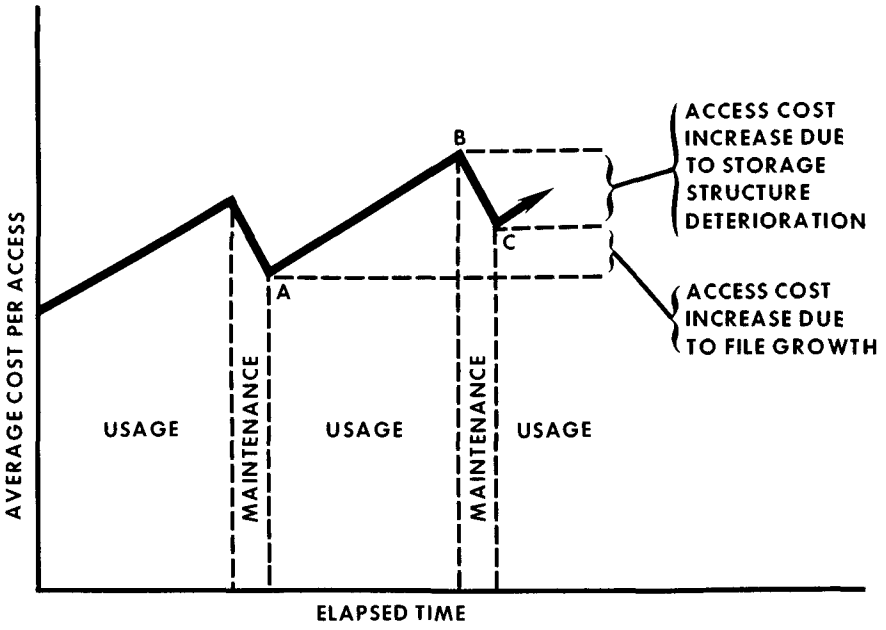


FIGURE 14 Access costs and maintenance.

cost per file access. Point A marks the end of a period of maintenance and the start of a period of usage. During usage, access time and cost increase owing to both file growth and deterioration of storage structures. At point B, the file is taken off-line for maintenance. The "cost" we plotted during the maintenance period represents the improvement in the storage structures. When maintenance ends and usage resumes at point C, the storage structures are optimal, but the access cost can still exceed that of point A because of file growth. The research efforts have analyzed such factors as characteristics of access cost increases due to file growth, characteristics of access cost increases due to storage structure deterioration, and the cost of performing maintenance. The models differ in their assumptions regarding linearity of deterioration and growth, and uniformity of usage period lengths. Results include policies for deciding when to perform maintenance so as to minimize the total cost of access and maintenance over a period of time.

3.3 Concurrent Reorganization and Usage

Another research area deals with performing reorganization concurrently with database usage. Off-line reorganization is unsatisfactory for two classes of databases. 1) If an essential computer utility, such as a military, hospital, police, or reservation database, is to be available 24 hours per day, then it cannot be brought off-line for significant periods of time. 2) For a very large database, such as census data, reorganization might require much longer than a holiday weekend. "A very large database is a database whose reorganization by reloading takes a longer time than the users can afford to have the database unavailable" [WIED77, p. 449]. Also, many DBAs prefer 24-hour availability, even if it is not essential. Thus it is appropriate and increasingly necessary to use techniques such as reorganizing concurrently with full usage of the reorganized portion.

Several research studies have dealt with this area. One study [SOCK77] has produced requirements for concurrent reorganization [SOCK76] (e.g., locking), the classification of types of reorganization described earlier, and a performance model [SOCK78]. The model predicts degradation of user response time and reorganization time under concurrent reorganization and usage. Another study [WILS79] has produced a set of operational rules to ensure correctness of con-

current reorganization and usage for CODASYL databases. A prototype implementation can add, delete, and reorder fields within a record type. An alternative to in-place concurrent reorganization is to record attempted user updates during unloading and reloading and to execute them later. SEVE76 describes a technique for managing this strategy and determining which objects have been updated. Another alternative is to reorganize incrementally as objects are referenced. GERR76 describes such a strategy for managing coexisting "generations" of schemata. BCS79 reports on work in progress on extending the CODASYL Data Storage Description Language [CODA78] to support versions of storage structures, so that physical reorganization can proceed off-line, concurrently with usage, or incrementally as objects are referenced. HULT79 proposes concurrent maintenance in which the locking granularity of a given type of maintenance determines the periods during which the maintenance is scheduled.

4. CONCLUSIONS

We draw the following conclusions from our study:

- Database reorganization is a necessary function. Failure to reorganize can result in high expense (as extra space and time are consumed), user dissatisfaction (as the mean and variance of response time increase), and limitations on functional capabilities (as desired new information cannot be represented). All DBMSs require some type of reorganization, and a database designer cannot predict all future instances of reorganization.
- Many varieties of reorganization exist—e.g., overflow removal or addition of a relationship. Current DBMSs generally provide facilities to perform many but not all types of reorganization.
- Performing reorganization can be time consuming and hence expensive, especially in very large databases. A long period of off-line reorganization can be intolerable in essential 24-hour utilities.
- The database administrator must determine an installation's reorganization policy, since reorganization can affect all users.
- Results of research efforts in reorganization will increase in importance as very large databases become more common.

ACKNOWLEDGMENTS

The authors would like to thank Deborah A. Sheetz, David K. Hsiao, Stuart E. Madnick, Donald R. Deutsch, John L. Berg, Belkis W Leong-Hong, Peter S. Mager, Jesse M Draper, and Tom B. Wilson for reviewing an earlier draft of this paper Peter P.-S. Chen, Ugo O Gagliardi, and Herbert S. Meltzer reviewed related earlier work. The late Michael E Senko provided valuable advice in the classification of types of reorganization. Several vendor representatives clarified our understanding of their systems. The referees suggested several improvements.

REFERENCES

ALSB75 ALSBERG, P. A. "Space and time savings through large data base compression and dynamic restructuring," *Proc IEEE* **63**, 8 (Aug. 1975), 1114-1122.

ALTM72 ALTMAN, E. B., ASTRAHAN, M. M., FEHDER, P. L, AND SENKO, M. E "Specifications in a data independent accessing model," in *Proc. ACM SIGFIDET Workshop Data Description, Access, and Control*, Nov. 1972, pp. 363-382.

ARME70 ARMENTI, A. W., GALLEY, S. W., GOLDBERG, R P., NOLAN, J. F., AND SHOLL, A. "LISTAR—Lincoln information storage and associative retrieval system," in *Proc. Spring Jt. Computer Conf.*, Vol. 36, AFIPS Press, Arlington, Va., May 1970, pp. 313-322.

ASTR72 ASTRAHAN, M. M., ALTMAN, E. B., FEHDER, P. L., AND SENKO, M. E. "Concepts of a data independent accessing model," in *Proc. ACM SIGFIDET Workshop on Data Description, Access, and Control*, Nov. 1972, pp. 349-362.

BACH69 BACHMAN, C. W "Data structure diagrams," *Data Base* **1**, 2 (Summer 1969), 4-10.

BCS75 BCS/CODASYL Data Description Language Committee Data Base Administration Working Group, Report, June 1975 (available from chairman, DBAWG, British Computer Society, London, England).

BCS79 BCS/CODASYL Data Description Language Committee Data Base Administration Working Group, *Reorganization*, standing paper 12, March 1979 (available from chairman, DBAWG, British Computer Society, London, England)

BERG80a BERG, J L (Ed.) *Data base directions—the conversion problem*, Nat. Bur. Stand Special Publ, 1980, to appear in *Data Base* and *SIGMOD Record* (ACM).

BERG80b BERG, J L., GRAHAM, M, AND WHITNEY, V. K. (EDS) *Database architectures—a feasibility workshop report*, Nat Bur Stand Special Publ, 1980

BUEL77 BUELL, F. B. Private communication, March 1977.

BUZE78 BUZEN, J P., GOLDBERG, R. P , LANGER, A M., LENTZ, E., SCHWENK, H. S JR , SHEETZ, D. A., AND SHUM, A W-C. "BEST/1—design of a tool for computer system capacity planning," in *Proc 1978 AFIPS NCC*, Vol. 47, AFIPS Press, Arlington, Va., June 1978, pp 447-455.

CHAM76 CHAMBERLIN, D. D. "Relational database management systems," *Comput. Surv.* (special issue on database management systems) **8,** 1 (March 1976), 43-66.

CHEN76 CHEN, P P-S. "The entity-relationship model—toward a unified view of data," *ACM Trans. Database Syst.* **1,** 1 (March 1976), 9-36.

CHEN77 CHEN, P. P.-S. "The entity-relationship model—A basis for the enterprise view of data," *Proc. 1977 AFIPS NCC*, Vol. 46, AFIPS Press, Arlington, Va., June 1977, pp. 77-84

CHEN78 CHEN, P P.-S. *The entity-relationship approach to logical data base design*, Data Base Monograph 6, Q.E.D. Information Sciences, Inc., Wellesley, Mass., 1978

CINC78 CINCOM SYSTEMS, INC *TOTAL/8 data base administration reference manual*, April 1978

CODA71 CODASYL Programming Language Committee, *Data base task group report*, April 1971 (available from ACM, New York).

CODA77 CODASYL Systems Committee, Stored-Data Definition and Translation Task Group, "Stored-data description and data translation a model and language," *Inform Syst.* **2,** 3 (1977), 95-148.

CODA78 CODASYL Data Description Language Committee, *Journal of development*, Jan. 1978 (available from Materiel Data Management Branch, Dep. of Supply and Services, Canadian Gov., Hull, Que., Canada)

CODD70 CODD, E. F. "A relational model of data for large shared data banks," *Commun. ACM* **13,** 6 (June 1970), 377-387.

CULL78 CULLINANE CORP. *IDMS utilities*, Release 5.0, Sept. 1978

DARD77 DARDWIN, D G "Reloading tough work with huge data base," *Computerworld* **11,** 45 (Nov 7, 1977), 38.

DATE77 DATE, C. J. *An introduction to database systems*, 2nd ed., Addison-Wesley, Reading, Mass., 1977.

DEUT78 DEUTSCH, D. R. "Modeling and measurement techniques for evaluation of design alternatives in the implementation of database management software," D.B.A. Diss., College of Business and Management, U. of Maryland, College Park, Md , Dec. 1978, Nat Bur Stand. Special Publ. 500-49, July 1979

EDEL76 EDELMAN, J. A , LIAW, Y S , NAZIF, Z. A , AND SCHEIDT, D. L *Reorganization system user's reference manual*, USE Program Library Interchange, Sperry Univac, Oct 1976.

FRY74 FRY, J P., AND JERIS, D. W. "Towards a formulation and definition of data reorganization," in *Proc. ACM SIGMOD Workshop on Data Description, Access and Control*, May 1974, pp. 83-100.

FRY76 FRY, J P , AND SIBLEY, E H. "Evolution of data-base management systems," in *Comput. Surv.* (special issue on database management systems) **8,** 1 (March 1976), 7-42.

FRY78 FRY, J. P , TEOREY, T. J., DESMITH, D. A , AND OBERLANDER, L. B. *Survey of state-of-the-art database administration tools survey results and evaluation*, Database Systems Research Group Tech Rep. DSRG 78 DE 14.2, Graduate School of Business Administration, U. of Michigan, Ann Arbor, Mich., Aug. 1978.

GERR76 GERRITSEN, R , AND MORGAN, H. L. "Dynamic restructuring of databases with generation data structures," in *Proc. ACM Ann. Conf.*, Oct. 1976, pp. 281-286

GRIF76 GRIFFITHS, P. P., AND WADE, B. W. "An authorization mechanism for a relational database system," *ACM Trans. Database Syst.* **1,** 3 (Sept. 1976), 242-255.

HOUS77 HOUSEL, B. C. "A unified approach to program and data conversion," in *Proc. 3rd Int Conf. on Very Large Data Bases*, ACM, New York, Oct. 1977, pp. 327-335.

HSIA78 HSIAO, D. K., KERR, D. S., AND MADNICK, S E "Privacy and security of data communications and data bases," in *Proc. 4th Int. Conf. on Very Large Data Bases*, ACM, New York, Sept. 1978, pp. 55-67.

HULT79 HULTÉN, C., AND SODERLUND, L. *A framework for concurrent physical reorganization of large data bases*, 1979 (available from authors at Dep of Information Processing and Computer Science, U of Stockholm, Stockholm, Sweden).

IBM73 IBM CORP *OS/360 data management services guide*, Form GC26-3746-2, July 1973

IBM76 IBM CORP. *OS/VS virtual storage access method (VSAM) programmer's guide*, Form GC20-3838-2, April 1976.

IBM77a *Information management system/virtual storage (IMS/VS) general information manual*, Form GH20-1260-6, July 1977.

IBM77b IBM CORP. *Information management system/virtual storage (IMS/VS) utilities reference manual*, Form SH20-9029-4, July 1977.

KERS76 KERSCHBERG, L , KLUG, A. C., AND TSICHRITZIS, D. C. "A taxonomy of data models," in P. C. Lockemann, and E J Neuhold, (Eds.), *Syst. for Large Data Bases (Proc 2nd Int Conf on Very Large Data Bases*, Sept. 1976), North-Holland, Amsterdam, 1977, pp. 43-64.

LEON77 LEONG-HONG, B W., AND MARRON, B. *Technical profile of seven data element dictionary/directory systems*, Nat. Bur. Stand. Special Publ. 500-3, Feb 1977.

LEON78 LEONG-HONG, B W , AND MARRON, B. *Database administration concepts, tools, experiences, and problems*, Nat. Bur Stand. Special Publ. 500-28, March 1978.

LEWI75 LEWIS, K., DRIVER, B , AND DEPPE, M. E *A translation definition language for the version II translator*, Data Translation Project working paper 809, Graduate School of Business Administration, U of Michigan, Ann Arbor, Mich April 1975.

LYON76 LYON, J. K. *The database administrator*, Wiley, New York, 1976.

MART77 MARTIN, J. *Computer data-base orga-*

nization, 2nd ed., Prentice-Hall, Englewood Cliffs, N J., 1977.

MARU76 MARUYAMA, K., AND SMITH, S. E. "Optimal reorganization of distributed space disk files," *Commun. ACM* **19,** 11 (Nov 1976), 634-642.

MELT75 MELTZER, H. S. "An overview of the administration of data bases," in *Proc 2nd USA-Japan Comput. Conf*, AFIPS Press, Arlington, Va , Aug. 1975, pp. 365-370.

MRI78 MRI SYSTEMS CORP. *System 2000 general information manual,* 1978.

NATI78 NATIONS, J., AND SU, S. Y W "Some DML instruction sequences for application program analysis and conversion," in *Proc. ACM SIGMOD Int. Conf. Management of Data*, May 1978, pp. 120-131.

NAVA76 NAVATHE, S B., AND FRY, J. P. "Restructuring for large databases: three levels of abstraction," *ACM Trans. Database Syst.* **1,** 2 (June 1976), 138-158.

RAMI74 RAMIREZ, J A., RIN, N. A , AND PRYWES, N S. "Automatic generation of data conversion programs using a data description language," in *Proc. ACM SIGMOD Workshop Data Description, Access, and Control*, May 1974, pp. 207-225.

SCHN76 SCHNEIDER, L. S. "A relational view of the data independent accessing model," in *Proc ACM SIGMOD Int. Conf. Management of Data*, June 1976, pp 75-90.

SENK73 SENKO, M. E., ALTMAN, E. B., ASTRAHAN, M. M., AND FEHDER, P L "Data structures and accessing in data-base systems," *IBM Syst. J.* **12,** 1 (1973), 30-93.

SENK75 SENKO, M. E. "Specification of stored data structures and desired output results in DIAM II with FORAL," in *Proc. 1st Int. Conf Very Large Data Bases*, ACM, New York, Sept. 1975, pp. 557-571.

SENK76 SENKO, M. E., AND ALTMAN, E. B. "DIAM II and levels of abstraction. The physical device level: a general model for access methods," in P. C. Lockemann and E. J. Neuhold (Eds.), *Syst. for Large Data Bases (Proc 2nd Int Conf. Very Large Data Bases*, Sept 1976), North-Holland, Amsterdam, 1977, pp. 79-94.

SEVE76 SEVERANCE, D G., AND LOHMAN, G. M. "Differential files: their application to the maintenance of large databases," *ACM Trans. Database Syst* **1,** 3 (Sept 1976), 256-267.

SHNE73 SHNEIDERMAN, B. "Optimum data base reorganization points," *Commun ACM* **16,** 6 (June 1973), 362-365.

SHOS75 SHOSHANI, A "A logical-level approach to data base conversion," in *Proc. ACM SIGMOD Int. Conf Management of Data*, May 1975, pp. 112-122

SHU75 SHU, N. C , HOUSEL, B. C., AND LUM, V. Y. "CONVERT a high level translation definition language for data conversion," *Commun ACM* **18,** 10 (Oct. 1975), 557-567

SHU77 SHU, N. C , HOUSEL, B. C., TAYLOR, R. W , GHOSH, S. P., AND LUM, V Y "EXPRESS: a data extraction, processing and restructuring system," *ACM Trans. Database Syst* **2,** 2 (June 1977), 134-174.

SIBL73 SIBLEY, E. H , AND TAYLOR, R. W. "A data definition and mapping language," *Commun. ACM* **16,** 12 (Dec. 1973), 750-759

SIBL76 SIBLEY, E. H. (Ed.) *Comput Surv.* (special issue on database management syst.) **8,** 1 (March 1976).

SIWI77 SIWIEC, J. E. "A high-performance DB/DC system," *IBM Syst. J.* **16,** 2 (1977), 169-195; and private communication, June 1977.

SMIT71 SMITH, D. C. P. "An approach to data description and conversion," Ph.D. Diss., Moore School of Elec. Eng., U. of Pennsylvania, Philadelphia, Pa., 1971.

SOCK76 SOCKUT, G H., AND GOLDBERG, R. P. "Motivation for data base reorganization performed concurrently with usage," Tech Rep. 16-76, Ctr. for Research in Computing Technology, Harvard U , Cambridge, Mass., Sept. 1976.

SOCK77 SOCKUT, G. H. "Data base performance under concurrent reorganization and usage," Ph.D Diss , Div. of Applied Sciences, Harvard U , Cambridge, Mass., Nov. 1977; Tech Rep. 12-77, Ctr for Research in Computing Technology, Harvard U., July 1977.

SOCK78 SOCKUT, G. H. "A performance model for computer data-base reorganization performed concurrently with usage," *Oper Res.* **26,** 5 (Sept.-Oct 1978), 789-804.

SOFT77 SOFTWARE AG OF NORTH AMERICA, INC *ADABAS introduction*, 1977.

SPER78 SPERRY UNIVAC *Data management system (DMS* 1100) *level 8R1 system support functions data administrator reference*, Univac Publ. UP 7909.1, 1978.

SU74 SU, S. Y. W., AND LAM, H "A semi-automatic data base translation system for achieving data sharing in a network environment," in *Proc ACM SIGMOD Workshop Data Description, Access, and Control*, May 1974, pp. 227-247.

SWAR77 SWARTWOUT, D. E., DEPPE, M. E., AND FRY, J. P. "Operational software for restructuring network databases," *Proc. 1977 AFIPS NCC*, Vol 46, AFIPS Press, Arlington, Va., June 1977, pp. 499-508

TAYL76 TAYLOR, R. W , AND FRANK, R L. "CODASYL data-base management systems," *Comput. Surv.* (special issue on database management systems) **8,** 1 (March 1976), 67-103

TAYL79 TAYLOR, R W., FRY, J. P., SHNEIDERMAN, B , SMITH, D. C P , AND SU, S. Y. W "Database program conversion a framework for research," in *Proc. 5th Int Conf Very Large Data Bases*, ACM, New York, Oct. 1979, pp. 299-312.

TSIC76 TSICHRITZIS, D C., AND LOCHOVSKY, F H "Hierarchical data-base management: a survey," *Comput Surv.* (special issue on database management systems) **8,** 1 (March 1976), 105-123.

TSIC77 TSICHRITZIS, D C , AND KLUG, A C. (Eds.) *ANSI/X3/SPARC DBMS framework report of the study group on data base management systems*, AFIPS Press, Arlington, Va., Nov 1977.

TUEL78 TUEL, W G , JR. "Optimum reorganization points for linearly growing files," *ACM Trans Database Syst.* **3,** 1 (March 1978), 32-40.

TUFT79 TUFTS, R. J. "Tools for the successful DBA," *Information Syst.—Effectiveness for the User (Proc. 18th Ann. ACM/NBS Tech. Symp.)*, ACM, Washington, D.C, Chapter, June 1979, pp. 131–144.

WEIS78 WEISE, R Private communication, Aug 1978.

WIED77 WIEDERHOLD, G. *Database design*, McGraw-Hill, New York, 1977.

WILS79 WILSON, T. B. *A general model for dynamic data base restructuring*, 1979 (available from author at Sperry Univac, Roseville, Minn.).

YAO76 YAO, S. B., DAS, K. S., AND TEOREY, T. J. "A dynamic database reorganization algorithm," *ACM Trans. Database Syst* 1, 2 (June 1976), 159–174.

RECEIVED MARCH 1979, FINAL REVISION ACCEPTED AUGUST 1979