An Access Path Model for Physical Database Design

R. H. Katz and E. Wong
Department of Electrical Engineering and Computer Science
University of California, Berkeley
Berkeley, CA. 94720

ABSTRACT: Design and Access Path Data Models are presented to form an integrated framework for logical and physical database design in a heterogeneous database environment. This paper focuses on the physical design process. First, a physical design is specified in terms of general properties of access paths, independent of implementation details. Then, a design is realized by mapping the specification into the storage structures of a particular database system. Algorithms for assigning the properties to logical access paths and for realizing a CODASYL 78 DBTG schema are given.

1. Introduction

As the trend towards distributed database systems continues to gain in momentum, the problem of database design in a heterogeneous environment is becoming crucial. We view a distributed database system as being built on top of existing systems available at the local sites of a computer network. If a distributed database is to evolve naturally, there must be support for extending it to the underlying heterogeneous systems.

Database design is complicated by the difficulty in designing physical databases for a variety of storage structures supported by the underlying systems. We follow [CARD75] in partitioning the physical design process into its implementation oriented (access path selection) and implementation dependent (storage structure choice) aspects. A physical design is specified in terms of basic properties of storage structure without making a commitment to an actual implementation. A design is "realized" by mapping the system independent specification into the storage structures available in a particular database system. Analytic methods, such as [CARD73, GOTL74, CARD75, YAO75, SILE76, DUHN78, SCHK78], can be used in the latter step. Our contribution is to provide an integrated framework for logical and physical design, and to provide design tools with a high degree of independence from the underlying data models and systems.

In this paper, we propose the concept of an access path data model for physical design. The access path model has grown out of the attempts to extend our work with the Design Model [WONG79] to problems of physical database design. The term "data model" is used in a generic sense to mean a collection of data object types, such as attributes and relations in the relational model. "Schema" is used to mean a specific choice of data objects to represent a database, such as a choice of specific relations and associated attributes.

The access path model can be viewed as an interface between the logical view of data and the access methods and storage structures chosen to support that view. In terms of the language of the ANSI/X3/SPARC report [TSIC77], it mediates between the conceptual and internal schemas.

We are not the first to exploit the usefulness of an access path model. The DIAM (Data Independent Access Model) framework [SENK73] is structured into four levels consisting of entity set, string, encoding, and physical device models. The string model is most closely associated with our notion of access path model. Although the DIAM model is a significant contribution, we believe that our formulation of access path is more natural and easy to understand. In addition, the access path schema is oriented towards the problem of physical design, rather than a general model of data management systems.

The paper is organized as follows. A semantic data model is presented which is the basis for our approach to database design. Logical access paths are represented by functional interrelationships between objects. The access path model is defined to capture those functions which can be used to efficiently access objects in the physical realization of the database. A methodology for specifying an implementation oriented physical design is given which is based on assigning the highest level support for the most frequently traversed access paths. A simple-minded approach for mapping

Research supported by the U.S. Army Research Office Grant DAAG-76-6-0245, the U.S. Air Force Office of Scientific Research Grant 78-3596, the Honeywell Corporation, and an I.B.M. Fellowship for the first author.

a design specification into the storage structures of CODASYL DBTG systems is included. We conclude the paper with a discussion on future directions.

2. The Design Model

The design model is the starting point for our approach to database design. It has been formulated to capture the kinds of integrity constraints supported by the relational and DBTG models, yet remains independent of them. The model is based on a variation of the entity-relationship model [CHEN76] and has been influenced by the semantic data model of [SCHM75]. A more complete discussion of the design model and its application to logical design and schema conversion can be found in [WONG79].

For each instance of time t, let $E_1(t)$, $E_2(t)$, ... , $E_n(t)$ be n distinct sets, which are called entity sets. A property of an entity set E(t) is a one-parameter family of functions f_t, mapping at each t E(t) into a set V of values. Because f_t is defined for every element of the domain, it is a total function. As an example, consider the following entity sets and properties:

entity sets	properties
emp	ename,birthyr
dept	dname,location
job	title,status,salary

A relationship R_t among entity sets $E_1(t)$, $E_2(t)$, ... , $E_n(t)$ is a subset of the cartesian product $E_1(t) \times E_2(t) \times \ldots \times E_n(t)$ at each time t. Properties of relationships may be defined in an analogous way to properties of entity sets. Relationships are assumed to be independent, i.e. not derivable from other relationships, and indecomposible, i.e. not equal to the join of their projections into subrelationships. For example, the following two relationships specify the employees qualified to hold each job, and the jobs allocated to a given department. "Number allocated" may be specified as a property of "allocation":

relationships	properties
qualified(job,emp)	----
allocation(dept,job)	number

We further distinguish the types of relationships recognized by the design model. A binary relationship R_t on entity sets $E_1(t)$ and $E_2(t)$ is single-valued in $E_1(t)$ if each entity of $E_1(t)$ occurs in at most one instance of R_t. Intuitively, we may think of R_t as representing a function from $E_1(t)$ into $E_2(t)$, because each entity in $E_1(t)$ can be related to no more than one entity in $E_2(t)$. If each entity in $E_1(t)$ occurs in exactly one instance of R_t, R_t is called an association. We may think of R_t as representing a total function. Single-valued relationships which are not associations can be thought of as partial functions, because at a given point in time, the function need not be defined over all entities in $E_1(t)$. Associations are used to model the situation in which the domain object can exist only if it is related to some range object. If an object in the range of an association is deleted, then the objects in the domain no longer occur in an instance of R_t. Therefore, they must be deleted to maintain the totality of the function. Examples of associations include:

works-in (emp,dept)
assignment (emp,job)

which represent the facts that an employee must work-in some department at all times and must be assigned to some job at all times. An example of a single-valued relationship which is not an association is:

mgr (dept,emp)

which associates a managing employee with a department, although a department can exist without a manager.

Explicit provisions for value set definitions have been omitted in our model. A subsystem such as that proposed in [MCLE76] could be included, but existing systems do not support sophisticated domain definition. A simpler approach is to use the primitive data types supported by most systems for the domain definition (e.g., integer, char(10), etc.).

Our design model can be reformulated to represent logical access paths in terms of total and partial functions between objects. This is similar to the approach taken in the functional data models of [BUNE79] and [SHIP80]. The objects of the schema are the value sets, entity sets, and relationships. Single-valued relationships are partial functions, while associations and properties are total functions. In addition, total functions can be defined to map a relationship object into the entity set objects over which it is defined.

The above example is reproduced here in terms of the functional viewpoint (some abbreviation has taken place):

total functions
ename: emp --> char(20)
title: job --> char(15)
birthyr: emp --> integer
salary: job --> integer
dname: dept --> char(10)
works-in: emp --> dept
location: dept --> char(20)
assignment: emp --> job
qual-emp: qual --> emp
qual-job: qual --> job
alloc-dept: alloc --> dept
alloc-job: alloc --> job
number: alloc --> integer

partial functions
mgr: dept --> emp

A design schema can be represented graphically. Let I = (V,E) be a directed graph with set V of vertices and set E of edges. For each object in the schema, there is a vertex in V. For each function from $object_1$ to $object_2$, there is a directed edge from the vertex for $object_1$ to the vertex for $object_2$. Value objects are represented by black vertices, non-value objects by white. The graphical representation of the example schema is shown in figure 1.

3. The Access Path Model

The functions of the design model represent logical access paths that can be used to navigate among the objects of the schema. For example, $WORKS\text{-}IN(ENAME^{-1}("fred"))$ gives us the department

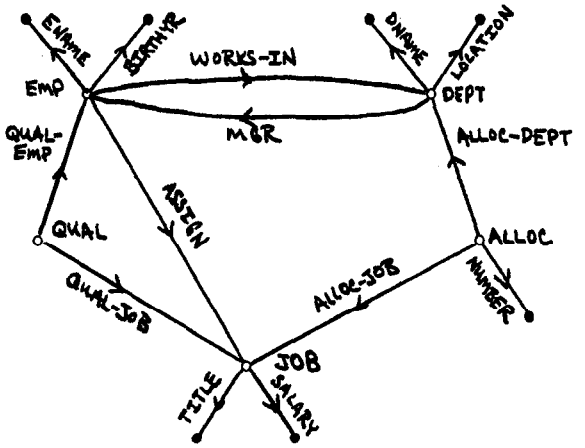


figure 1 - Graphical Representation of Schema

that Fred works in. The access path schema is concerned with those functions and inverses that are "supported" for efficient access by the underlying database system. "Support" is used in an operational sense to mean that the time to perform a supported access is less than the time to perform an unsupported one.

When used to access objects, logical access paths are called access mappings. An access mapping may be defined for either a function or its inverse. To make it possible to compose access mappings, we extend the definition to allow them to be applied to sets of domain objects. An access mapping is supported in the storage structure if the database system can efficiently perform the desired access, i.e., the time to access an object via a supported access map is less than the time to scan the object set exhaustively for the desired object(s). If an access mapping is not supported, it is an unsupported access mapping. Supported access mapping is our terminology for the usual notion of access path.

An access path schema consists of the objects of the design schema and the supported access mappings. A graphical representation similar to the one proposed in the previous section can be used to represent an access path schema. The schema must continue to represent all logical interrelationships, whether or not they are efficiently supported. For example, WORKS-IN associates with each employee a single department. If WORKS-IN is not supported, we must still be able to access the associated department, albeit not as efficiently as if the mapping had been supported. To accomplish this, we introduce the concept of identifier. An identifier is a 1-to-1 property of an entity set which is used to uniquely represent each entity in the set. An unsupported access mapping between employees and departments can be represented instead as an access mapping between employees and the identifier value set of department.

```
(supported)   WORKS-IN: emp --> dept
(unsupported) WORKS-IN:
                  emp --> identifiers of dept
```

The access path schema, together with the assigned storage structure properties (introduced in section 4.1), captures the effects of storage structure support without committing the schema to a particular implementation and without sacrificing any of the interrelationships of the design schema. WORKS-IN can be used to navigate directly between employees and departments only if the mapping is supported by the underlying system. It is immaterial whether this support is furnished by a physical pointer between employee records and department records, an index that maps employee identifiers into department records, or some other technique.

4. Physical Database Design

The access path schema provides a useful interface between the user's logical view of the data and its physical implementation. In this section, we will describe an implementation oriented physical design methodology which is largely independent of the specific database system and data model. The implementation dependent aspects will be discussed in section 5.

The approach is to generate designs which provide the best possible support for the most travelled access paths, without conflicting with the support for other paths. A specification of the user's expected access patterns is used to direct the design process. A system specific mapping is then invoked to implement the access path schema by choosing storage structures supported by the target system.

4.1 Algebraic Structure for Physical Design

For the purposes of implementation oriented design, we shall use the logical access paths of the design schema. An access path schema may be used to represent those paths actually chosen for support. Properties of an access mapping can be formulated to capture desirable characteristics of traversing the mapping in either the functional or inverse functional direction. Consider the schema function f: A --> B. The following properties of the mapping can be defined:

(1) Evaluated: given a in A, f(a) can be found without an exhaustive scan of B, i.e., the cost to access f(a) is less than the cost to access every element of B.

(2) Indexed: given b in B, $f^{-1}(b)$ can be found without an exhaustive scan of A.

(3) Clustered: the elements of $f^{-1}(b)$ are in close proximity, i.e., the cost to access the elements in the inverse is less than the cost to access an arbitrary subset of the same cardinality.

(4) Well Placed: a and f(a) are stored in close proximity, i.e., the cost to access both is less than the cost to access them separately.

We make the critical assumption that each object of the schema, be it a value, an entity, or a relationship instance, is assigned to a single stored record. Replication, e.g., the replication

of data item values to record instances, will be made explicit by introducing new objects into the schema. The usual concept of "record" can be represented as a concatenation of the stored records of the values that make up the fields of the record. Our approach does not preclude the record segmentation and allocation techniques described in [SCHK78]. Given this assumption, certain implication rules can be formulated:

(i) well placed ==> evaluated

By placing f(a) near a, a fast way to get from the domain to the range is automatically provided. It is no longer necessary to scan the entire set of range objects to find the desired one.

(ii) clustered ==> indexed

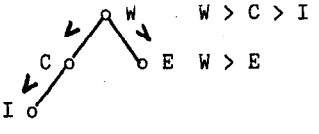
By placing elements of $f^{-1}(b)$ together, an exhaustive scan of all the domain objects of f is not necessary. The scan is considerably speeded up by placing the objects together.

(iii) well placed ==> clustered

Let b = f(a). Well placed means that a and b are stored together. Since there is one record for each b instance, all A objects with b in the range of f will be placed near b and hence near each other. Thus clustering is achieved.

For systems without index storage structures, it is possible to have a mapping which is evaluated but not indexed. For example, an employee's name may be stored in the record that represents the employee, with no storage structures available to access the record via an employee name. The opposite is possible as well. Some inverted file systems allow access to a record through a value that is not stored in that record. For example, an employee's name may not be stored with the record that represents the employee, but an index on employee name is available. Thus evaluated need not imply indexed and vice versa.

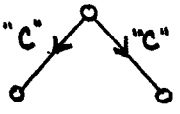
The implication rules can be used to impose a partial ordering among the properties:



A label is an assignment of properties to an edge of the integrity schema. There are six distinct labels: W, <C,E>, <I,E>, C, I, and E. Our algorithm will generate schemas with maximally supported access paths. We assume that all access paths are at least evaluated. Therefore we deal only with the first three labels, denoted as "W", "C", and "I". A labelling is an assignment of a label to each edge of the schema, denoted as an n-tuple $(l_1, l_2, \ldots, l_n)$ where n is the number of edges in the schema. The assignment is subject to constraints which are shown below. The partial ordering among properties induces a partial ordering among labels as well: "W" > "C" > "I". A partial ordering can be defined for labellings. Let L_1 and L_2 be two labellings over the same schema. We say that $L_1 = L_2$ if for each edge in the schema, L_1's assigned label is the same as L_2's assigned label. We say that $L_2 > L_1$ if for each edge in the schema, either L_1's assigned label is the same as L_2's or L_2's label > L_1's, and $L_1 \neq L_2$. Note that under this definition, some labellings are incomparable, e.g. L_1 = ("W","C") and L_2 = ("C","W").

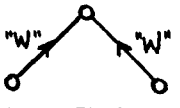
An obvious approach to achieving a maximal labelling is to assign "W", the label that represents the highest degree of support, to each edge. Unfortunately, certain labellings represent a choice of properties which can not be supported simultaneously within a schema. There are four constraints which conflict-free labellings must meet:

(i) cluster constraint: it is not possible to label more than one outedge of a node with a "C" or "W". Clustering places together all domain objects which are mapped into the same range object.

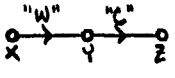


It is not possible to partition the domain on more than one function and still achieve this advantageous placement. Note that 1-to-1 properties do not cause a conflict because a 1-to-1 function partitions the domain objects into clusters of size one. This can always be supported regardless of additional clustering.

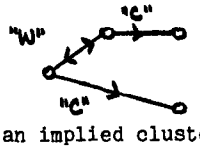
(ii) placement constraint: it is not possible to label more than one inedge of a node with "W". Well-placement places clusters of domain objects with a common range object near that range object. It is not possible to achieve this advantageous placement simultaneously for domain objects from more than one function.



(iii) placement-cluster constraint: it is not possible to simultaneously label an inedge of a node "W" while labelling an outedge "C". The placement of X object clusters near their associated Y objects destroys the advantageous clustering of the Y objects. 1-to-1 functions do not cause the constraint to be violated.



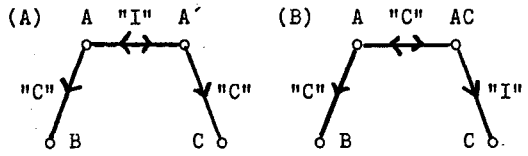
(iv) implied constraints: Certain compositions of functions and their properties result in the violation of one of the above constraints. For example, this schema would cause a violation of an implied cluster constraint.



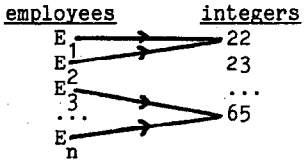
The above constraints are conservative in the sense that the desirable properties of placement and clustering can be achieved, even if the constraints are violated. However, this tends to be sensitive to the parameters of a particular system. For example, in constraint (ii), if clusters from both domain object sets can fit on the same page, then simultaneous well placement can be achieved.

The degree of a schema is the number of violations of placement or cluster constraints that may be made during the labelling process. Each of these violations can be resolved if we introduce the replication of objects. Assume that the schema is labelled as in (i). A conflict is a violation of a cluster or placement constraint. A

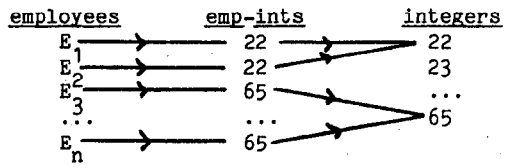
cluster conflict can be resolved by one of the following methods:



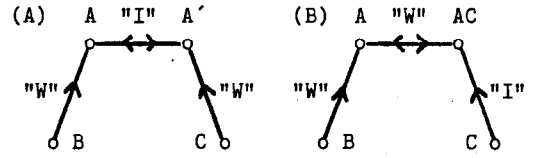
In (A), a copy of the domain object is made, and both the original and the copy are clustered on the appropriate ranges. In (B), a copy of the range is made and placed in one-to-one correspondence with the original domain object. To illustrate this, consider the entity set employees and the value set integers, interrelated by the property function age. Schematically, the following situation can arise:



The effect of type (B) cluster resolution is to replicate the age values so there is one age value per employee:



A placement conflict is resolved in an analogous way:



The degree of a schema is a measure of the amount of replication we are willing to tolerate during the labelling process. Replicated information introduces increased costs for storage and update while reducing retrieval costs. A degree of 0 insures that no replication will result, i.e., the cluster and placement constraints are never violated; a degree of n > 0 will allow up to n replicated objects to be created.

A maximal labelling is a labelling L for which there exists no labelling L' such that L' > L. Our objective is to generate maximal conflict-free labellings. Because not all labellings are comparable, it is possible to generate many such labellings for the same schema. Rather than generate all the possible labellings for a given schema, usage information can be use to restrict the enumeration to those that best support the expected usage patterns of the database.

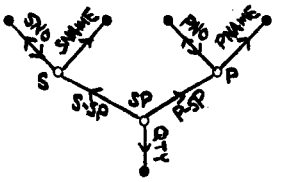
4.2 A Labelling Algorithm

In this subsection, we present an algorithm for generating a maximal labelling that specifies superior support for the access paths most heavily travelled. Assume that the degree of replication is n. This means that up to n placement or cluster conflicts will be tolerated while labelling the schema. These conflicts will later be resolved using the techniques of the previous subsection.

The input to the algorithm is a schema to be labelled and a ranking of the edges (access mappings) according to frequency of traversal. The algorithm only enforces cluster and placement constraints. Initially all edges are labelled "I". We begin by assigning the next most favorable label ("C") to the heaviest used edge. We continue assigning labels in this manner until either n cluster conflicts have been detected or all edges have been examined. Then we assign the most favorable label ("W") to the most heavily used edge that is already labelled "C". We continue until a total of n cluster or placement conflicts are detected. The edge that causes the $n+1^{th}$ conflict is not relabelled. The algorithm to assign labels is given in figure 2. When all edges have been assigned a label, resolution is performed for each vertex which does not meet the placement and cluster constraints. Type (A) placement resolution is chosen for conflicts involving association and single-valued relationship edges and type (B) for conflicts involving property edges.

The algorithm can be illustrated with an example. Consider the sample schema:

S(SNO,SNAME)
P(PNO,PNAME)
SP(S,P,QTY)



Consider the following ranking of access mappings, from most to least heavily used.

1) S-SP
2) SNO
3) P-SP
4) QTY
5) PNO
6) PNAME
7) SNAME

```
Algorithm LabelSchema
#conflicts <- 0
for each edge do label edge "I"
for each edge
    (in frequency of access order) do
    label edge "C"
    if cluster conflict then
       if #conflicts = n
          then relabel edge "I"
       else #conflicts <- #conflicts + 1
for each edge labelled "C"
    (in frequency order) do
    label edge "W"
    if placement conflict then
       if #conflicts = n
          then relabel edge "C"
       else #conflicts <- #conflicts + 1
```

figure 2 - Labelling Algorithm

This ranking could have been derived from a set of user queries in conjunction with an indication of relative frequency, or simply specified by the designer. For a degree of replication = 1, the algorithm proceeds as follows:

initial labelling: all edges labelled "I"

C-labelling:
STEP 1: label S-SP with "C"

STEP 2: label SNO with "C"

STEP 3: P-SP can not be labelled "C" without a conflict. Label it "C". No additional conflicts are allowed.

STEP 4: QTY can not be labelled "C" without a conflict.

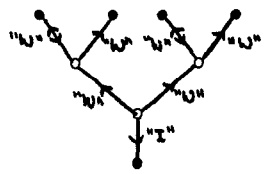
STEP 5: label PNO with "C"

STEP 6: label PNAME with "C" (does not conflict with PNO)

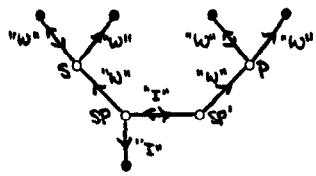
STEP 7: label SNAME with "C" (does not conflict with SNO)

W-labelling: all edges labelled "C" can be labelled "W" without conflict.

The resulting labelling is:



Placement resolution must be performed for SP. The more frequently used edge will eminate from the original SP while the less frequently used edge will eminate from the replicated SP'. Type (A) resolution is used because the conflicting edge, P-SP, involves non-value vertices:



We note that fully constrained labelling can be formulated in terms of an integer linear program with an objective function that maximizes the sum of the frequencies of the edges labelled "W" and "C". Further details can be found in [KATZ80].

5. Implementing a Schema

Up to this point, the design has been independent of the actual data model and system. In this section we briefly discuss the considerations involved in mapping a labelled schema into DBTG storage structures.

The quality of the mapping depends on the detail of usage information specified. In the following, we assume that information has been specified at the level of the previous section. All property mappings are "evaluated" supported by placing the range value in the record that represents the entity or relationship instance.

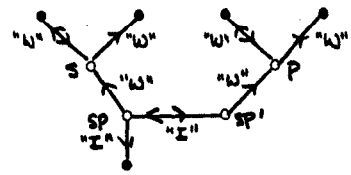
In the new CODASYL proposal [CODA78], many aspects of the physical database design have been removed from the schema DDL and localized in data storage definition. The DSDL provides facilities for the specification of the pagination of the storage media, schema to storage record mapping, record pointer implementation, set representation, and storage record placement. We do not deal with the specification of the storage media, and assume that all sets are represented by chains with direct pointers. Additional usage information could be used to make a more sophisticated choice for these parameters.

The DSDL provides three choices for the record placement strategy. A record may be calc'd (hashed) on a key specified in the DDL, clustered by set membership and optionally placed near the owner, or stored in sequential sorted order. Indexes can be specified separately for keys specified in the DDL.

At most one non-identifier outedge of a node can be labelled "W" or "C". This edge should be used to determine the primary structure of the record type if its traversal frequency exceeds that of the identifier outedge. Otherwise, the identifer outedge (which can always be labelled "W") should be used. In the latter case, the record type is calc'd on the related key data item. In the former, if the outedge is a property, then the record type is stored sequentially and sorted and indexed on the appropriate data item. Otherwise the outedge represents an association or single-valued relationship, and the record type is clustered on the associated set. If "W" is specified, the records are placed near their owners. Indexes are created for data items whose associated property mappings are labelled "I". The algorithm of figure 3 can be used to determine the record type's structure.

The DSDL also provides facilities to allow a single schema record to be represented by multiple stored records. This corresponds closely to our formulation of replication. Consider the following degree 1 labelling and its associated CODASYL schema:

record types	sets
S(SNO,SNAME)	S-SP, Owner S, Member SP
P(PNO,PNAME)	P-SP, Owner P, Member SP
SP(QTY)	



The DSDL specification for the schema would be:

```
MAPPING FOR S
   STORAGE RECORD IS S

MAPPING FOR P
   STORAGE RECORD IS P

MAPPING FOR SP
```

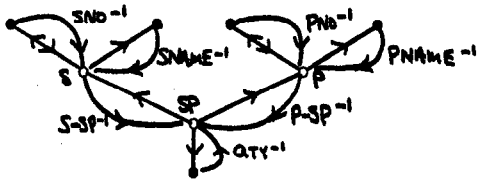
Algorithm CodasylPhysicalDesign

```
FOR EACH non-value node DO
  IF node is entity set THEN
     i <- identifier outedge
     j <- other outedge labelled "W" or "C"
     IF f_i > f_j

       THEN calc on key data item
       ELSE
         IF j is association or
            s.v. relationship
           THEN cluster on set membership
             IF label = "W"
                THEN place near owner
           ELSE /* property outedge */
             sort & index on data item
     FOR EACH property edge labelled "I" DO
       index on data item
  ELSE /*relationship */
    j <- outedge labelled "W" or "C"
    IF association

      THEN cluster on set membership
        IF label = "W"
          THEN place near owner
      ELSE sort & index on data item
     FOR EACH property edge labelled "I" DO
       index on data item
```

figure 3 - CODASYL Design

```
   STORAGE RECORDS ARE SP,SP'

STORAGE RECORD NAME IS S
   PLACEMENT IS SEQUENTIAL ASCENDING SNAME
   SET S-SP ALLOCATION IS STATIC
      POINTER FOR FIRST, LAST RECORD SP
         IS TO SP

STORAGE RECORD NAME IS P
   PLACEMENT IS SEQUENTIAL ASCENDING PNAME
   SET P-SP ALLOCATION IS STATIC
      POINTER FOR FIRST, LAST RECORD SP
         IS TO SP'

STORAGE RECORD NAME IS SP
   LINK TO SP'
   PLACEMENT IS CLUSTERED
      VIA SET S-SP NEAR OWNER S
   SET S-SP ALLOCATION IS STATIC
      POINTER FOR NEXT, PRIOR
      POINTER FOR OWNER

STORAGE RECORD NAME IS SP'
   LINK TO SP
   PLACEMENT IS CLUSTERED
      VIA SET P-SP NEAR OWNER P
   SET P-SP ALLOCATION IS STATIC
      POINTER FOR NEXT, PRIOR
      POINTER FOR OWNER
```

plus specification for INDEXES for each data item not covered in the above. The access schema for the above is:



All access mappings are maximally supported. Usage information may indicate that certain paths are not worth the overhead of supporting them.

6. Conclusions and Future Work

In this paper we have proposed an access path model for physical database design as an extension of our original work with a semantic model for logical database design and schema conversion. The properties of access paths were discussed and a methodology which generates maximally supported schemas was proposed and illustrated with examples. We believe that this approach to qualitative physical design is new and unique.

We have briefly discussed the applications of our methodology for designing CODASYL physical databases. More work is required on usage specification in order to improve the quality of the design.

The access path model also has applications to problems of program translation. A generalized query processing algorithm can be formulated to "compile" non-procedural queries, e.g., relational calculus, into the access paths supported in the access schema. Primitive operations on the access schema can be defined in a way that facilitates implementing these operations in terms of CODASYL DML. In addition, we have been investigating how to reverse the process, i.e., "decompiling" programs that access data at the level of DML into non-procedural queries, with the aid of the access schema. These problems are further explored in [KATZ80].

7. References

[BUNE79] Buneman, P., Frankel, R. E., "FQL -- A Functional Query Language," Proc. A.C.M. SIGMOD Conf., (May 79).

[CARD73] Cardenas, A. F., "Evaluation and Selection of File Organization - A Model and a System," Comm. A.C.M., V 16, N 9, (Sep 73).

[CARD75] Cardenas, A. F., "Analysis and Performance of Inverted Data Base Structures," Comm. A.C.M., V 18, N 5, (May 75).

[CHEN76] Chen, P. P., "The Entity-Relationship Model - Toward a Unified View of Data," A.C.M. Trans. on Data Base Sys., V 1, N 1, (Mar 76).

[CODA78] CODASYL Data Description Language Committee Journal of Development, 1978.

[DUHN78] Duhne, R. A., Severence, D. G., "Selection of an Efficient Combination of Data Files for a Multiuser Database," Proc. AFIPS Natl. Comp. Conf., 1978.

[GOTL74] Gotlieb, C. C., Tompa, F. W., "Choosing a Storage Schema," Acta Informatica, V 3, pp.

297 - 319, 1974.

[KATZ80] Katz, R. H., "Database Design and Translation for Multiple Data Models," Univ. of California, Berkeley, Ph.D. Thesis, in preparation.

[MCLE76] McLeod, D. J., "High Level Domain Definition in a Relational Data Base System," 1976 A.C.M. SIGMOD - SIGPLAN Conf. on Data, (Mar 76).

[SCHM75] Schmid, H. A., Swenson, J. R., "On the Semantics of the Relational Data Model," Proc. A.C.M. SIGMOD Conf., (May 75).

[SCHK78] Schkolnick, M., "A Survey of Physical Database Design Methodology and Techniques," Proc. of Conf. on Very Large Data Bases, 1978.

[SENK73] Senko, M. E., et. al., "Data Structures and Accessing in Data-Base Systems," IBM Systems Journal, V 12, N 1, 1973.

[SHIP80] Shipman, D. W., "The Functional Data Model and the Data Language DAPLEX," to appear in A.C.M. Trans. on Database Systems, 1980.

[SILE76] Siler, K. F., "A Stochastic Model for Database Organizations in Data Retrieval Systems," Comm. A.C.M., V 19, N 2, (Feb 76).

[TSIC77] Tsichritzis, D., Klug, A., eds., "The ANSI/X3/SPARC DBMS Framework - Report of the Study Group on Data Base Management Systems," 1977.

[WONG79] Wong, E., Katz, R. H., "Logical Design and Schema Conversion for Relational and DBTG Databases," Proc. Intl. Conf. on Entity-Relationship Approach to Systems Analysis and Design, (Dec 79).

[YAO 75] Yao, S. B., Merten, A., "Selection of File Organization Using an Analytic Model," Proc. Intl. Conf. on Very Large Data Bases, 1975.