BASIC DATA STRUCTURE MODELS EXPLAINED WITH A COMMON EXAMPLE.

Gordon C Everest
University of Minnesota Twin Cities

Reprinted from Computing Systems 1976,
Proceedings Fifth Texas Conference on Computing Systems,
Austin, TX, 1976 October 18-19, pages 39-46.
(Long Beach, CA: IEEE Computer Society Publications Office).

BASIC DATA STRUCTURE MODELS EXPLAINED WITH A COMMON EXAMPLE

Gordon C. Everest
University of Minnesota, Minneapolis, Minnesota

ABSTRACT: Several discussions of data structure models have appeared in the literature in the past few years. This paper presents a taxonomy of basic data structures which cuts through and omits the secondary differences to highlight the most important logical differences. First, all data structure models are divided into single file models and multifile models. A file is a collection of data about a set or class of entities which possess some common characteristics. The notions of entity, attribute, domain, value, and identification relate to important characteristics of a file. Single file models further divide into flat file data structures and hierarchical data structure models. Multifile data structure models are divided into coordinated files, network, relational, and binary data structure models. The essential characteristics of each of these six basic data structure models are described and each is exemplified using a common example containing personnel and organizational data.

KEYWORDS AND PHRASES: Data structures, data models, flat file, hierarchical data structure, coordinate files, multifile data model, network data structure, relational data model, binary data structures.

This paper relates solely to the structural aspects of data models. Discussions of data models in the literature often do not clearly define the data model before launching into a discussion of operators and languages used to specify processes on those structures and the resulting behavior of those structures. The reader is then lead to infer things about the data model and form value judgments sometimes incorrectly.

For example, a discussion of a low-level, record-at-a-time language on a hierarchical structure set beside a discussion of a powerful high-level language on a relational structure leads the reader to favor the relational data model. Unfortunately, the choice of data model is based on the language and not the data model. When evaluating data models, several factors must ultimately be considered -- language, behavior, stability, ease of user visualization, storage structures, accessibility, etc. But first it is necessary to understand the various basic data models, strictly in terms of their structure or shape.

This paper gets down to basics. It cuts through historical accidents and pragmatic dodges, and it cuts through the specific variants of particular data structure models. By getting down to basics, the paper may seem idealized. For example, the CODASYL-DBTG model does not typify the network structure per se because it has variants and restrictions which are not essential to the network data model. Also, it is incomplete in not explicitly providing for something that is assumed to be part of the network model.

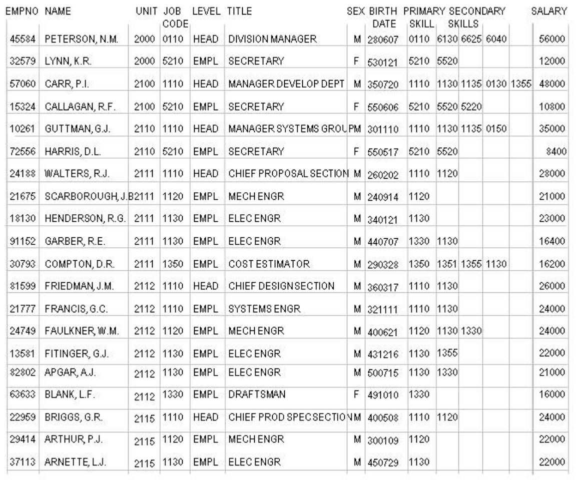
The basic data structure models discussed in this paper are first divided into single file models and multifile models. Even though a user may be able to deal with multiple files in a database system, if the system does not know the interfile relationships and cannot process the files together, it uses a single file data model.

SINGLE FILE DATA STRUCTURE MODELS

Considering a single file, the basic data models are the flat file and the hierarchical file. This classification is extended by considering a data model that is a collection of interrelated files, either flat files or hierarchical files. The relational data model is a collection of a special type of flat files. The network data model can be either a collection of flat files or a collection of hierarchical files. The last data model in the classification is a collection of binary relations.

Flat File Data Model

An example of a single flat file is shown in Figure 1. A file is a collection of data about a set of entities which possess some common characteristics. By possessing some common characteristics the entities constitute an entity class. The proper interpretation of the file centers on the type of entity it describes. In the example, the entity is an Employee. In general, an entity can be any concrete or abstract object or event in the real world about which we want to store data.



EMPNO	NAME	UNIT	JOB CODE	LEVEL	TITLE	SEX	BIRTH DATE	PRIMARY SKILL	SECONDARY SKILLS				SALARY
45584	PETERSON, N.M.	2000	0110	HEAD	DIVISION MANAGER	M	280607	0110	6130	6625	6040		56000
32579	LYNN, K.R.	2000	5210	EMPL	SECRETARY	F	530121	5210	5520				12000
57060	CARR, P.I.	2100	1110	HEAD	MANAGER DEVELOP DEPT	M	350720	1110	1130	1135	0130	1355	48000
15324	CALLAGAN, R.F.	2100	5210	EMPL	SECRETARY	F	550606	5210	5520	5220			10800
10261	GUTTMAN, G.J.	2110	1110	HEAD	MANAGER SYSTEMS GROUP	M	301110	1110	1130	1135	0150		35000
72556	HARRIS, D.L.	2110	5210	EMPL	SECRETARY	F	550517	5210	5520				8400
24188	WALTERS, R.J.	2111	1110	HEAD	CHIEF PROPOSAL SECTION	M	260202	1110	1120				28000
21675	SCARBOROUGH, J.B	2111	1120	EMPL	MECH ENGR	M	240914	1120					21000
18130	HENDERSON, R.G.	2111	1130	EMPL	ELEC ENGR	M	340121	1130					23000
91152	GARBER, R.E.	2111	1130	EMPL	ELEC ENGR	M	440707	1330	1130				16400
30793	COMPTON, D.R.	2111	1350	EMPL	COST ESTIMATOR	M	290328	1350	1351	1355	1130		16200
81599	FRIEDMAN, J.M.	2112	1110	HEAD	CHIEF DESIGN SECTION	M	360317	1110	1130				26000
21777	FRANCIS, G.C.	2112	1110	EMPL	SYSTEMS ENGR	M	321111	1110	1130				24000
24749	FAULKNER, W.M.	2112	1120	EMPL	MECH ENGR	M	400621	1120	1130	1330			24000
13581	FITINGER, G.J.	2112	1130	EMPL	ELEC ENGR	M	431216	1130	1355				22000
82802	APGAR, A.J.	2112	1130	EMPL	ELEC ENGR	M	500715	1130	1330				21000
63633	BLANK, L.F.	2112	1330	EMPL	DRAFTSMAN	F	491010	1330					16000
22959	BRIGGS, G.R.	2115	1110	HEAD	CHIEF PROD SPEC SECTION	M	400508	1110	1120				24000
29414	ARTHUR, P.J.	2115	1120	EMPL	MECH ENGR	M	300109	1120					22000
37113	ARNETTE, L.J.	2115	1130	EMPL	ELEC ENGR	M	450729	1130					22000

Figure 1. Example of a Flat File

The data in a single flat file describes a set of entities belonging to the same class. Each entity is described in terms of the values of the set of attributes listed across the top of the columns.

SOURCE: Thanks to Richard G. Canning who prepared this sample database for The Second Symposium on Computer-Centered Data Base Systems proceedings edited by C. Baum and L. Gorsuch, Santa Monica, CA: Systems Development Corp, TM-2624 (NTIS AD 625 417), 1965 Dec. pages 3-3 to 3-29.

A flat file is visually two-dimensional. Referring to Figure 1, each column corresponds to an attribute of the class of entities, and each row corresponds to one entity in the class of entities, that is, an entity instance. An attribute is a named characteristic of an entity class. The file describes an entity by the values of the attributes in its row. A single row of attribute values is also called an entry of the file. An entry in the file describes an entity instance in the real world.

Five characteristics serve to define a file. Each characteristic relates to a particular term or concept.

Concept	File Characteristic

ENTITY: EVERY FILE HAS A NAME WHICH SHOULD REFLECT THE CLASS OF ENTITIES.

ATTRIBUTE: A SET OF NAMED ATTRIBUTES DESCRIBE EACH ENTITY.

Each column in the two dimensional representation of a file corresponds to an attribute. The attributes are uniquely named within **a** file. The term data item is often used to denote the representation of an attribute in a defined file.

DOMAIN: ASSOCIATED WITH EACH ATTRIBUTE IS A DOMAIN OF VALUES.

In our example, all the values in a given column look to be of the same type. The domain for the attribute 'SEX' includes the two values of 'M' and 'F'. The domain of the attribute 'EMPNO' appears to include any five digit number. In general, each domain contains the values of 'UNKNOWN' and 'IRRELEVANT'.

For example, Maiden Name would only be relevant if SEX equals 'F'. All domains will have a collating sequence on the set of values. For some domains, the values will have arithmetic significance, such as with 'SALARY'.

VALUE: EACH ENTITY HAS ASSIGNED TO IT ONE AND ONLY ONE VALUE FROM EACH ATTRIBUTE DOMAIN.

Looking at the example, there is only one value for each row in each column.

IDENTIFICATION: ALL ROWS ARE DISTINCT, THAT IS, EACH ENTITY IS UNIQUELY IDENTIFIED BY THE VALUE OF ONE OR MORE OF ITS ATTRIBUTES.

COROLLARY: AN ENTITY INSTANCE IS NOT NAMED PER SE.

Stated another *way,* no two rows or entries in the file will have exactly the same set of values. If, in the development of a file, this characteristic is violated, it means that an insufficient number of attributes have been defined to properly describe the entities. One result of this characteristic is that, in general, *any* attribute can be used to identify or select a subset of entities in the file. In some files the relative position of the entity in the file may be necessary for unique identification. This situation can arise when short-lived entities are continually being added to and deleted from the file on either a first-in-first-out basis, as with a stack of entities. In such cases, it can be argued that rank is an attribute of each entity and this information is encoded in the relative position of the entry within the file.

The important concepts and terminology relating to a single flat file are shown in the schematic of Fig 2.

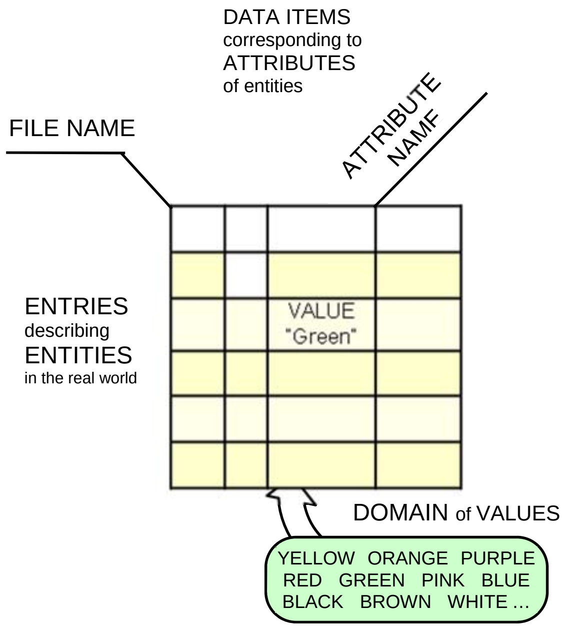
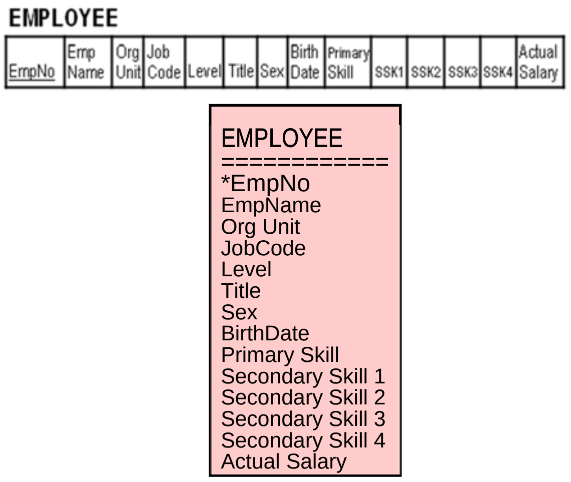

DATA ITEMS corresponding to ATTRIBUTES of entities
FILE NAME
ATTRIBUTE NAMF
ENTRIES describing ENTITIES in the real world
VALUE "Green"
DOMAIN of VALUES
YELLOW ORANGE PURPLE RED GREEN PINK BLUE BLACK BROWN WHITE ...

Figure 2. Important Data Concepts and Terminology

Sometimes it is more convenient to represent only the type of data in a particular flat file without showing the actual data. A schema representation does this as shown in Figure 3. At a minimum, the schema for a flat file is essentially just a list of the attributes used to describe each entity in the file. This minimum schema can be augmented with additional information about the flat file such as a description of the types of values for each attribute, the size of each value, the value domain for each attribute either as a range declaration or an enumeration, and a description of the attribute. In general, schema information is any information about the file which can be stated in terms of all the entries in the file.



EMPLOYEE

EmpNo	Emp Name	Org Unit	Job Code	Level	Title	Sex	Birth Date	Primary Skill	SSK1	SSK2	SSK3	SSK4	Actual Salary

EMPLOYEE
============
*EmpNo
EmpName
Org Unit
JobCode
Level
Title
Sex
BirthDate
Primary Skill
Secondary Skill 1
Secondary Skill 2
Secondary Skill 3
Secondary Skill 4
Actual Salary

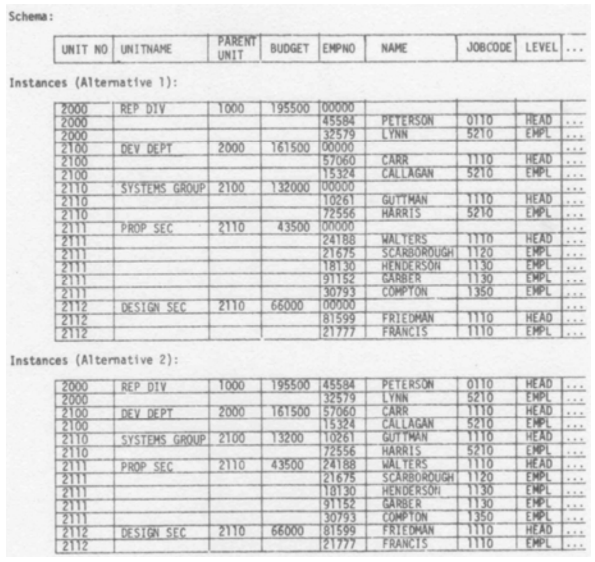
Figure 3. Two Representations of a Flat File Schema

The schema representation of a single flat file is essentially a list of the attributes used to describe each entity in the file. It can also contain additional information which is common across all entries in the file.

With more complex data structure models, the schema representation is needed to focus attention on the types of data and the relationships in a defined structure. The schema representation of a flat file serves as a building block to describing more complex structures.

A homogeneous flat file is a special case of the flat file in which every entry contains the same set of data items and each value appears in a fixed position in each entry instance. A homogeneous flat file is often easier for a user to comprehend and easier for the system to process.

A composite flat file results when a user attempts to define a flat file to contain data pertaining to multiple types of entities. The main symptom of a composite flat file is the existence of many IRRELEVANT item values in the entries. An example is shown in Figure 4 wherein an entry schema contains both organizational and personnel data.



Schema:

UNIT NO	UNITNAME	PARENT UNIT	BUDGET	EMPNO	NAME	JOBCODE	LEVEL	...

Instances (Alternative 1):

2000	REP DIV	1000	195500	00000				
2000				45584	PETERSON	0110	HEAD	...
2000				32579	LYNN	5210	EMPL	...
2100	DEV DEPT	2000	161500	00000				...
2100				57060	CARR	1110	HEAD	...
2100				15324	CALLAGAN	5210	EMPL	...
2110	SYSTEMS GROUP	2100	132000	00000				...
2110				10261	GUTTMAN	1110	HEAD	...
2110				72556	HARRIS	5210	EMPL	...
2111	PROP SEC	2110	43500	00000				...
2111				24188	WALTERS	1110	HEAD	...
2111				21675	SCARBOROUGH	1120	EMPL	...
2111				18130	HENDERSON	1130	EMPL	...
2111				91152	GARBER	1130	EMPL	...
2111				30793	COMPTON	1350	EMPL	...
2112	DESIGN SEC	2110	66000	00000				...
2112				81599	FRIEDMAN	1110	HEAD	...
2112				21777	FRANCIS	1110	EMPL	...

Instances (Alternative 2):

2000	REP DIV	1000	195500	45584	PETERSON	0110	HEAD	...
2000				32579	LYNN	5210	EMPL	...
2100	DEV DEPT	2000	161500	57060	CARR	1110	HEAD	...
2100				15324	CALLAGAN	5210	EMPL	...
2110	SYSTEMS GROUP	2100	13200	10261	GUTTMAN	1110	HEAD	...
2110				72556	HARRIS	5210	EMPL	...
2111	PROP SEC	2110	43500	24188	WALTERS	1110	HEAD	...
2111				21675	SCARBOROUGH	1120	EMPL	...
2111				18130	HENDERSON	1130	EMPL	...
2111				91152	GARBER	1130	EMPL	...
2111				30793	COMPTON	1350	EMPL	...
2112	DESIGN SEC	2110	66000	81599	FRIEDMAN	1110	HEAD	...
2112				21777	FRANCIS	1110	EMPL	...

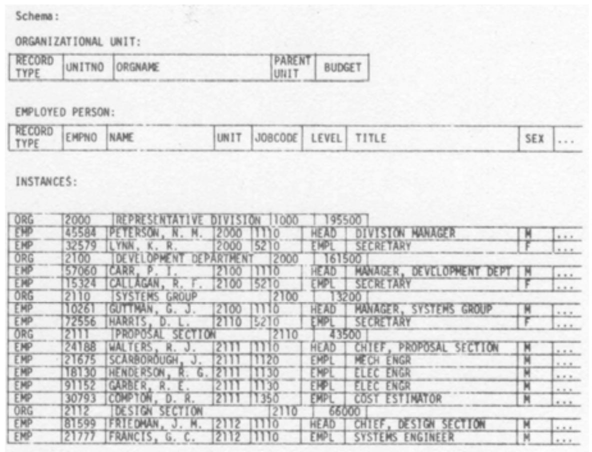
Figure 4. Example of a Composite Flat File.

A composite entry is defined to contain data about both organizational units and employees. In alternative 1, the data item called EMPNO indicates which type of data is stored in the entry instance: a value of 00000 in EMPNO for organizational data in which case the employee attributes are irrelevant, a value greater than 00000 in EMPNO for employee data in which case the organizational data is irrelevant except for the UNITNO showing where the employee works. Under alternative 2, the organizational data is stored in the same entry as the person who is the 'HEAD' of the organizational unit. In reality a composite flat file attempts to contain information about multiple entity classes.

In alternative 1, the EMPNO data item is used (instead of a separate RECORD TYPE data item) to indicate whether the entry describes an employee or an organizational unit. For an organizational unit entry, EMPNO would be set to a special value (say 00000) which could not be a valid employee number. The first item is relevant for all entry instances; the next three items are only relevant for organizational entities (EMPNO = 00000); the remaining items are only relevant for person entities (EMPNO > 00000). Such a composite flat file can result in substantial wasted space.

Rather than associate each entry exclusively with one specific entity type, each entry could describe a person along with the data pertaining to the organization in which that person works. This results in redundant data since the same organizational data is repeated for each person working in the same organizational unit. This can also represent some problems on retrieval. For example, it is no longer possible to sum the BUDGET data item to obtain the total budget for the organization. One alternative is to store the budget data only with the HEAD person of each organizational unit, as shown in alternative 2 of Figure 4. Note that the schema representation of these two variants is the same, indicating that additional information is necessary to properly define the semantics of the flat file data structure.

One might think of another variant which is to set up a different entry format for each of the two types of entities, as shown in Figure 5. Actually, this representation is no longer a single flat file because it cannot be described with a single schema. The particular schema to be used in interpreting an entry depends upon the value in the RECORD TYPE data item. This alternative gives rise to any one of the other data structure models — hierarchic, network, or relational.



Schema:

ORGANIZATIONAL UNIT:

RECORD TYPE	UNITNO	ORGNAME	PARENT UNIT	BUDGET

EMPLOYED PERSON:

RECORD TYPE	EMPNO	NAME	UNIT	JOBCODE	LEVEL	TITLE	SEX	...

INSTANCES:

ORG	2000	REPRESENTATIVE DIVISION	1000	195500				
EMP	45584	PETERSON, N. M.	2000	1110	HEAD	DIVISION MANAGER	M	...
EMP	32579	LYNN, K. R.	2000	5210	EMPL	SECRETARY	F	...
ORG	2100	DEVELOPMENT DEPARTMENT	2000	161500				
EMP	57060	CARR, P. T.	2100	1110	HEAD	MANAGER, DEVELOPMENT DEPT	M	...
EMP	15324	CALLAGAN, R. F.	2100	5210	EMPL	SECRETARY	F	...
ORG	2110	SYSTEMS GROUP	2100	13200				
EMP	10261	GUTTMAN, G. J.	2100	1110	HEAD	MANAGER, SYSTEMS GROUP	M	...
EMP	72556	HARRIS, D. L.	2110	5210	EMPL	SECRETARY	F	...
ORG	2111	PROPOSAL SECTION	2110	43500				
EMP	24188	WALTERS, R. J.	2111	1110	HEAD	CHIEF, PROPOSAL SECTION	M	...
EMP	21675	SCARBOROUGH, J.	2111	1120	EMPL	MECH ENGR	M	...
EMP	18130	HENDERSON, R. G.	2111	1130	EMPL	ELEC ENGR	M	...
EMP	91152	GARBER, R. E.	2111	1130	EMPL	ELEC ENGR	M	...
EMP	30793	COMPTON, D. R.	2111	1350	EMPL	COST ESTIMATOR	M	...
ORG	2112	DESIGN SECTION	2110	66000				
EMP	81599	FRIEDMAN, J. M.	2112	1110	HEAD	CHIEF, DESIGN SECTION	M	...
EMP	21777	FRANCIS, G. C.	2112	1110	EMPL	SYSTEMS ENGINEER	M	...

Figure 5. A "Single" File with Two Entry Types.

With multiple entry types, the entries in a file are not described using a single entry schema. While the system may treat this as a single file (by ignoring the contents of an entry), it is no longer a single flat file. Such a structural representation gives rise to any one of the other data structure models -- hierarchic, network, or relational.

Hierarchical Data Structure Model

The limitations of the single flat file data model become evident when trying to design a file to contain data that pertains not only to persons in an organization, but to organizational units and to skills. The basic problem is the need to represent more than one entity in the same data structure model. First, the multiple entities are not described according to the same set of attributes. Secondly, there may be some specific relationships between the various entities. In our example, a person may possess multiple skills, and multiple persons may work in an organizational unit.

Breaking out the three types of entities in our example yields the schema and instance representations of a hierarchical file as shown in Figure 6. The data can be stored in a "single file" in the form of three "record" types along with their hierarchical relationships. The hierarchical relationships are usually represented implicitly in the sequence of the records. Each ORGANIZATIONAL UNIT record is followed by (logically, at least) all the EMPLOYEE records for those persons working in that organizational unit, and each EMPLOYEE record is followed by a list of their SKILLS.

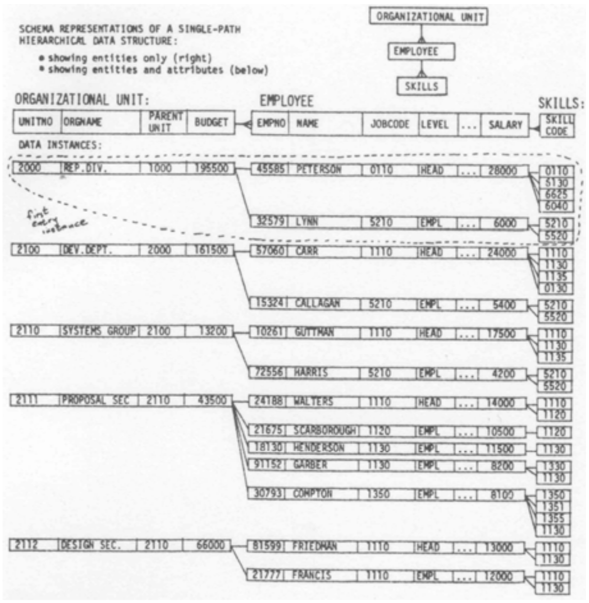

SCHEMA REPRESENTATIONS OF A SINGLE-PATH HIERARCHICAL DATA STRUCTURE:
• showing entities only (right)
• showing entities and attributes (below)
ORGANIZATIONAL UNIT
EMPLOYEE
SKILLS
ORGANIZATIONAL UNIT:
UNITNO | ORGNAME | PARENT UNIT | BUDGET
EMPLOYEE
EMPNO | NAME | JOBCODE | LEVEL | ... | SALARY
SKILLS:
SKILL CODE
DATA INSTANCES:
first entry instance
2000 | REP.DIV. | 1000 | 195500
45585 | PETERSON | 0110 | HEAD | ... | 28000
0110 5130 6625 6040
32579 | LYNN | 5210 | EMPL | ... | 6000
5210 5520
2100 | DEV.DEPT. | 2000 | 161500
57060 | CARR | 1110 | HEAD | ... | 24000
1110 1130 1135 0130
15324 | CALLAGAN | 5210 | EMPL | ... | 5400
5210 5520
2110 | SYSTEMS GROUP | 2100 | 13200
10261 | GUTTMAN | 1110 | HEAD | ... | 17500
1110 1130 1135
72556 | HARRIS | 5210 | EMPL | ... | 4200
5210 5520
2111 | PROPOSAL SEC | 2110 | 43500
24188 | WALTERS | 1110 | HEAD | ... | 14000
1110 1120
21675 | SCARBOROUGH | 1120 | EMPL | ... | 10500
1120
18130 | HENDERSON | 1130 | EMPL | ... | 11500
1130
91152 | GARBER | 1130 | EMPL | ... | 8200
1330 1130
30793 | COMPTON | 1350 | EMPL | ... | 8100
1350 1351 1355 1130
2112 | DESIGN SEC. | 2110 | 66000
81599 | FRIEDMAN | 1110 | HEAD | ... | 13000
1110 1130
21777 | FRANCIS | 1110 | EMPL | ... | 12000
1110 1130

Figure 6.

Example of a Single-Path Hierarchical Data Structure.

Two schema representations are shown at the top with some instances shown below. Notice the one-to-many, mutually exclusive and collectively exhaustive (i.e., total function), parent-dependent relationships between ORGANIZATIONAL UNIT and EMPLOYEEs and between and EMPLOYEE and SKILLS. This is the central characteristic of hierarchical data structures and is represented by the "inverted arrows" between the entity boxes of the schema representation.

This example is a single hierarchical file since, at the top or entry level, the entire data structure has a single entity connotation — that of ORGANIZATIONAL UNIT. An entry in a single hierarchical file consists of one instance of the top record type (ORGANIZATIONAL UNIT), plus all of its dependent (EMPLOYEE) records, plus all of their dependent records (SKILLS), and so on. The first entry is outlined in Figure 6.

This example is a single-path hierarchical file because there is only one entity type represented at each level in the schema representation of the data structure. Consequently, the schema representation only has a single path extending down from the entry-level entity at the top of the structure. Adding entities for AUTHORIZED POSITIONS and PROJECTS assigned to an organizational unit creates a multipath hierarchical data structure (see Figure 7).

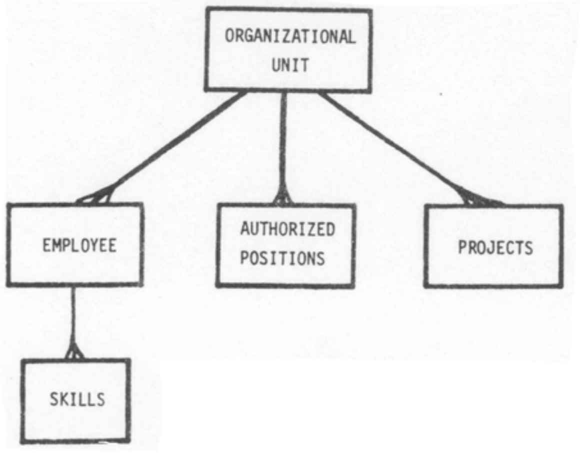


Figure 7. Schema for a Multipath Hierarchical Data Structure.

The parent-dependent relationship is represented in the schema picture using an "inverted arrow." In a downward direction the arrow represents a "contains" relationship -- an organizational unit contains employees. In an upward direction, the arrow represents a single-valued function, that is, given an EMPLOYEE (the independent variable or domain-of the function), a particular ORGANIZATIONAL UNIT is uniquely identified (the dependent variable or range of the function).* Stated in more general terms, there is a one-to-many relationship between ORGANIZATIONAL UNITs and EMPLOYEE such that each employee belongs to one and never more than one organizational unit. (This relationship is also characterized as "mutually exclusive and collectively exhaustive.") The inverted arrow is preferable to other notations because it ***inherently*** and ***visually*** depicts the one-to-many relationship.

In a hierarchical data structure, a dependent entity (record) is assumed to have no independent existence apart from its parent. For example, an EMPLOYEE could not exist in the file without being a dependent to some ORGANIZATIONAL UNIT (unless, of course, a "dummy" organizational unit were created to provide a home for an employee not belonging to an organizational unit). Deleting an EMPLOYEE from the file is assumed to imply that all the SKILL records should also be deleted. Similarly, deleting an ORGANIZATIONAL UNIT would imply deleting all its dependent EMPLOYEES — not always a reasonable assumption. This points up an anomaly which can result from a hierarchical data structure model.

MULTIFILE DATA STRUCTURE MODELS

One solution to the dilemma presented by the single hierarchical data model is to define multiple, independent files. Such a strategy is the basis for both the network data model and the relational data model. The coordinated file model is a rudimentary multifile data structure model.

Multiple, Coordinated Files

In the coordinated file model, a system can process multiple files if they are all ordered according to the same key criteria. For example, if several files in an organization contained employee identification numbers, then each file can be sorted by employee ID and a coordinated file system could read entries from each of the files in parallel. By finding a match on employee ID across multiple entries from the parallel files, the system can logically create a composite entry for each employee identifier. In effect, the system can read entries from multiple coordinated files as though there was only one big file. This is the basic mode of operation of several database retrieval systems such as Data Analyzer, Mark IV, ASI-ST, and Culprit.

In our example, it would be possible to retain the EMPLOYEE flat file as shown in Figure 1 (which includes the employee skills), and construct a separate flat file to ***contain*** the organizational data, the top level record in Figure 6. They could be processed as coordinate files by sorting each file on **organizational** unit number since that attribute is common to both files (see Figure 8). Of course, some convention must

*The "contains" or one-to-many relationship is the central idea behind the data structure diagrams described by Charles W. Bachman [1]. Nijssen [8,9] has discussed the relationship between Bachman's data structure diagrams and the notion of a single-valued function.

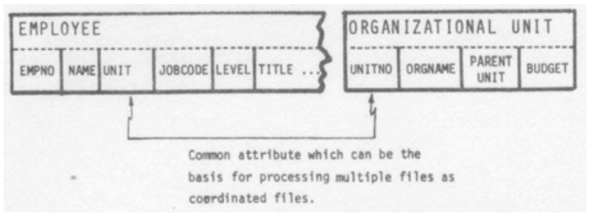

EMPLOYEE
EMPNO
NAME
UNIT
JOBCODE
LEVEL
TITLE ...
ORGANIZATIONAL UNIT
UNITNO
ORGNAME
PARENT UNIT
BUDGET
Common attribute which can be the basis for processing multiple files as coordinated files.

Figure 8. Coordinated Processing of Multiple Files.

be established for handling the multiple employees per organizational unit. One alternative is to focus on the employee adding on the organizational data as each employee record is processed. The other alternative is to focus on the organizational unit and add data for all employees when processing each organizational unit. The basic decision relates to the real world entity which is to be processed -- employee or organizational unit.

Network Data Structure Model

The chief characteristic of the network data model is the explicit and separate declaration of relationships between multiple entry types. The relationship is normally defined with one entry type as the parent and one (or more) entry types as dependents. An instance of the relationship involves one instance of the parent entry type and multiple instances of the dependent entry type(s). This defines a one-to-many relationship and can be graphically represented as was done with hierarchical data model schemas. A directed arc (drawn as an inverted arrow) can represent one relationship.

In general, more than one relationship can be defined between any two entry types, and, therefore, unique names must be assigned to each defined relationship. The schema representations of several network structures are shown in Figure 9.

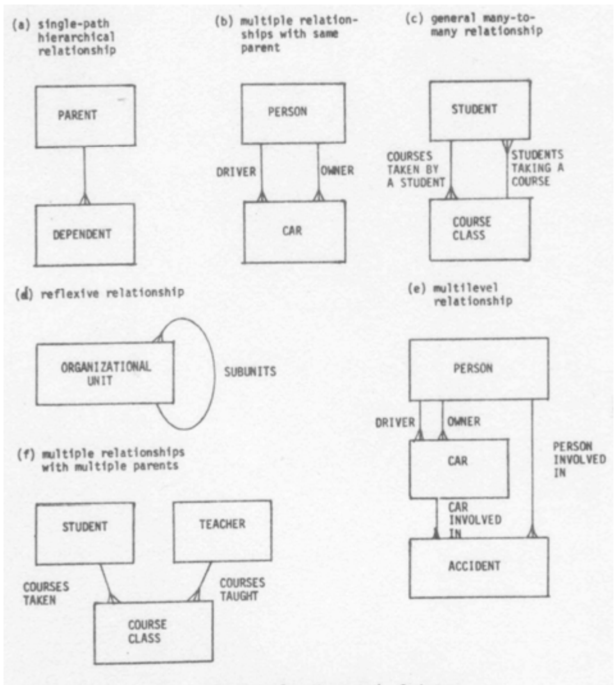

(a) single-path hierarchical relationship
PARENT
DEPENDENT
(b) multiple relationships with same parent
PERSON
DRIVER
OWNER
CAR
(c) general many-to-many relationship
STUDENT
COURSES TAKEN BY A STUDENT
STUDENTS TAKING A COURSE
COURSE CLASS
(d) reflexive relationship
ORGANIZATIONAL UNIT
SUBUNITS
(e) multilevel relationship
PERSON
DRIVER
OWNER
CAR
PERSON INVOLVED IN
CAR INVOLVED IN
ACCIDENT
(f) multiple relationships with multiple parents
STUDENT
TEACHER
COURSES TAKEN
COURSES TAUGHT
COURSE CLASS

Figure 9. Several Sample Network Schemas.

Sometimes it is desirable to define a relationship where both the parent and dependent are the same entry type — called a reflexive relationship. For example, an organization unit consists of several subunits. In fact, this relationship was captured in the ORGANIZATION UNIT entry of Figure 6 with the inclusion of the PARENT UNIT data item. It is also possible for the same entry type to be the dependent in more than one relationship type with either one parent entry type or multiple parent entry types. It is this characteristic of multiple parents which gives rise to the name "network" data structure model.

The primitive relationships shown in Figure 9 can be used repeatedly to build up complex network data structures. Since an entry type can simultaneously be a dependent in one relationship and the parent in another relationship, it is possible to construct multi-level structures. For some large databases even this abbreviated schema representation yields a complex picture. Figure 10 shows about one-sixth of the database for a single application in a large business organization.

c

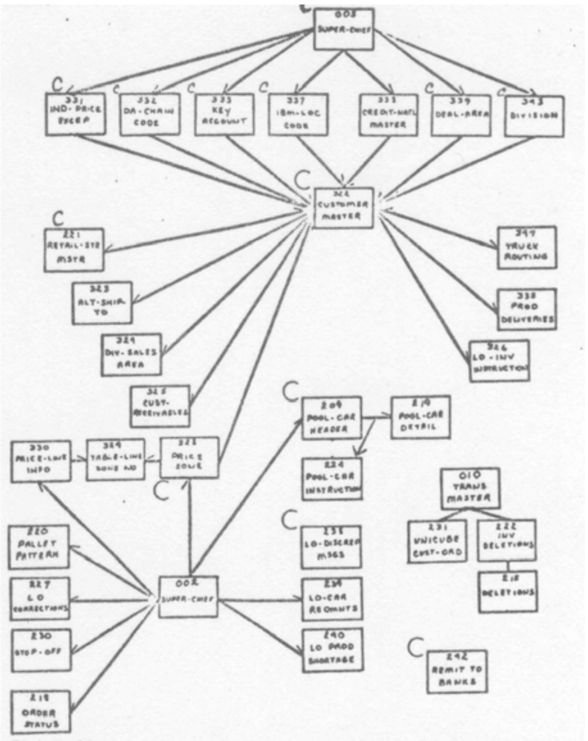


Figure 10.
Schema Representation of a Large Network-Structured Database.

Some network systems have restrictions on the types of relationships which can be defined. For example, it is not uncommon to prohibit reflexive relationships. Some systems also restrict entry types to be defined as a flat file, that is, containing no nested repeating data items or groups of items (that is, not a hierarchical entry structure).

In addition to representing a one-to-many relationship, the arc in the network schema diagrams may also represent the defined avenues of access. In fact, in some network systems, these are the only avenues of access to some entry types. The relationship may define two basic avenues of access as direct (using some form of hashing algorithm) or via an instance of the parent entry type. Such access restrictions are not inherent aspects of the network data model.

One characteristic which is common to network data structures is the lack of required stated criteria for the existence of the relationship between instances of entity types. For example, the relationship between ORGANIZATIONAL UNIT and EMPLOYEE would be defined but the UNIT data item would be omitted from the EMPLOYEE entry -- such information would be implicit in the defined relationship. In this sense the relationship is more than merely an access path; it carries logical information which is essential to the data structure. Without a UNIT data item in the EMPLOYEE entry, some mechanism must be established (such as a chain, or a set of pointers) to connect the dependent entry instances with their parent entry instance. An explicitly stored UNIT data item would constitute this connection mechanism without the separate declaration of a relationship. The criteria for the relationship would be a match on the values of UNIT in EMPLOYEE entries and UNITNO in an ORGANIZATIONAL UNIT entry.

In the network model, the multiple files are logically distinct; the entries of one are no longer thought of as being interleaved with the entries in another file as in the picture of a hierarchical file. Also, each file may be flat or hierarchical. Generally, the entries of one file have an existence independent of the entries in another file. In our example, a user may want to consider only EMPLOYEE entries without regard for ORGANIZATIONAL UNIT entries. Some network systems, however, permit the definition of a relationship in which the existence of dependent entries is dependent upon the existence of a parent entry, and access to the dependent entries is only via the parent entry.

Relational Data Structure Model

The relational data model as promulgated by Codd [4,5] and others is actually very similar to the network model, despite the many arguments, debates, and comparisons in the literature. The relational data model excludes access path information. All relationships are implicit in explicitly defined attributes in the entry types. In our example, UNIT would be included in the EMPLOYEE entry type and that alone would constitute the relationship between employees and organizational units. Additional relationships, which are separately defined in the network model, can be derived in the relational model using the language facilities.

Whereas the network data structure is defined in terms of multiple entry types (each constituting a file) and a set of separately defined relationships on those entry types, the relational data structure is defined simply as a set of files. The files are interrelated to the extent that they have common attributes.

Moreover, the multiple files in a relational data structure must be normalized flat files. For every flat file, a key must be defined consisting of a minimum set of attributes needed to uniquely identify the entries of the file. To be in normalized form, all attributes in a flat file must be functionally dependent only on the key and on the whole key.

Referring to our example in Figure 6, the hierarchical relationship between ORGANIZATIONAL UNIT and EMPLOYEE must be flattened by explicitly storing the UNIT data item in the EMPLOYEE ENTRY. Although most presentations of relational data structures do not show the schema in a graphical form, it is still meaningful to do so with the arc containing redundant information which is reflected in the common attribute.

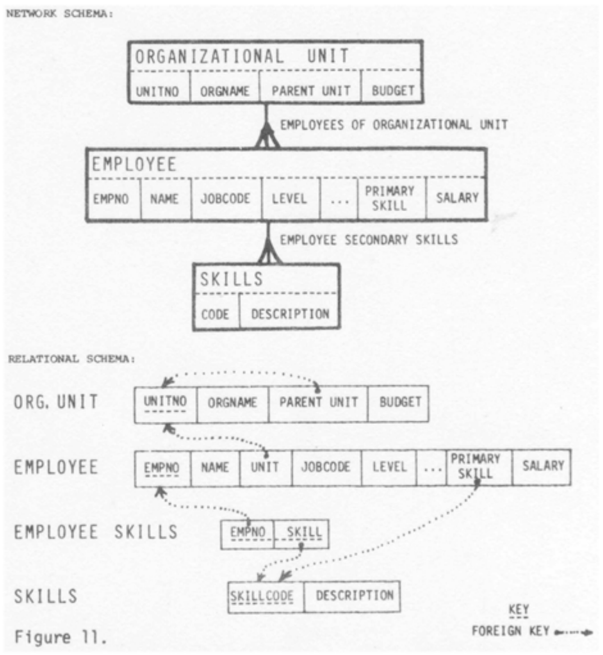


Figure 11.

Network and Relational Schemas of the Same Data Structure

NOTES:
- the reflexive relationship on organizational unit is represented the same way in each model.
- Declared relationships carry essential information in the network model.
- Foreign keys in the relational model are equivalent to the one-to-many binary relationships in the network model.
- Representation of direct, many-to-many relationships or greater than binary relationships requires the definition of a linkage entry in either model.

Figure 11 shows the same logical structure for our organizational and employee data in a network schema representation and in **a** relational schema representation. Of course, these schema representations do not capture all information about the structure. The schema definition would contain additional semantics of the data items such as type and value set. There are several important points of comparison between these two representations.

- The reflexive relationship on organizational unit is represented in the same way in each model.

- The one-to-many (mutually exclusive and collectively exhaustive) binary relationship between organizational units and employees is represented in the network model by an explicitly declared relationship and in a relational model as a foreign key which is the organizational unit number stored in the employee entry. In the network model, when the key to a parent entry is not stored, the declared binary relationship carries essential information—the arcs cannot be removed from the network schema representation. A foreign key in the relational model is logically equivalent to a declared binary relationship in the network model. However, there is an implicit difference in terms of accessibility. In the network model the implied direction of access is from an organizational unit to its multiple "contained" employees (from range to domain); whereas in the relational model, the implied direction pf access is from an employee to his/her uniquely identified organizational unit (from domain to range). To be sure in both models, access is (should be) possible in either direction. (In the relational model, there is no implied preferred direction of access if the relationship is 1:1 rather than 1:N.)

• As defined in the network model, the skills entry involves considerable redundancy. When the same secondary skill is possessed by more than one employee, a separate skills record (containing both skill code and its description) is required for each instance. (An alternative would be to define a value encoding table to associate each skill code with its description and only store skill codes for each employee.)

• If the relationship between two entities is direct and many-to-many or is more than binary, then in both the network and relational models it is necessary to define another entry type ("dummy" record or relation). This is exemplified in the relational model schema. The network model can either duplicate the same skill codes under the several employees who possess that skill (as in Figure 11) or set up a new linking record type, exactly equivalent to that shown in the relational model schema. A direct entity relationship uses a foreign key. An indirect entity relationship is established between entities which have attributes from the same domain. In our example, both employees and departments could have "location" attributes and then be related through a common location.

In discussing the relational data model, some different terminology is used in the literature:

relational data model -- a collection of normalized flat files
relation — a normalized flat file
tuple -- an entry in a flat file

Whereas the network structure employs the three basic constructs of attribute, entity (corresponding to record type), and (binary) relationship, the relational data structure model only employs the two basic constructs of attribute and relation. (The notion of domain represents additional definitional information about attributes and is not a basic construct.) The relation construct includes both the entity and the relationship constructs. For this reason some have concluded that the relational data model is simpler than the network data model. However, a normalized relational data structure can result in substantially more separate flat files which makes it more difficult for a user to comprehend the overall data structure. Furthermore, a graphical schema representation, so necessary, to human understanding, could still be extremely complex for a large database structure. It is interesting to observe the lack of graphical representations in articles pertaining to the relational data model.

Binary Relational Data Structure Model

This discussion is based upon the work of Michael E. Senko [12].

Rather than represent entity-attribute associations in an n-ary relation, as in the case of a flat file or normalized relation with n attributes, a binary relational model represents each meaningful pairwise association of two attributes. The familiar two-dimensional representation of a flat file is replaced with a graphical representation in which each node corresponds to a domain of values and each arc corresponds to a binary relation between two domains.

It is sometimes difficult to distinguish between entities and attributes. In the EMPLOYEE file the UNIT number is an attribute; in the ORGANIZATIONAL UNIT file UNIT is the entity being described. For another example, in a file of new cars, color may be an attribute; to the manager of the paint department, color may be the entity of a file with various attributes describing the characteristics of each color: name, how to mix it, what pigments to use, reflectivity coefficient, etc. In the binary data model, each domain is a class of "entities" and each value is a symbolic identifier of a member of the class. The phrase "entity domain" is used in the context of binary relations.

In the formation of a binary relational data structure, each flat file is essentially "chopped up" into its separate columns or entity domains. Then binary relations are defined between pairs of entity domains. In this way it is often possible to represent more structural information than in the simple flat file. Stated another way, there is less ambiguity in a binary representation of the data structure.

A binary relation defined on two entity domains is a two-way association which defines an attribute on each entity. Referring to Figure 12, one entity serves as a descriptor of the other via the binary association. Given the two entity domains EMPLOYEE NUMBER and JOBCODE, the two arcs of the binary relation are labelled "employee-number-of-jobcode" and "jobcode-of-employee-number." One attribute of EMPLOYEE NUMBER - 57060 is JOBCODE - 1110, similarly, one attribute of JOBCODE 1110 is EMPLOYEE NUMBER - 57060. The functional relationship is written between the two arcs of the binary relation. In this case, there is a one-to-many relationship between JOBCODE and EMPLOYEE NUMBER. Note that EMPLOYEE NUMBER = 81599, 10261, and 24188 are also attributes of JOBCODE = 1110.

Instances of a binary relation:

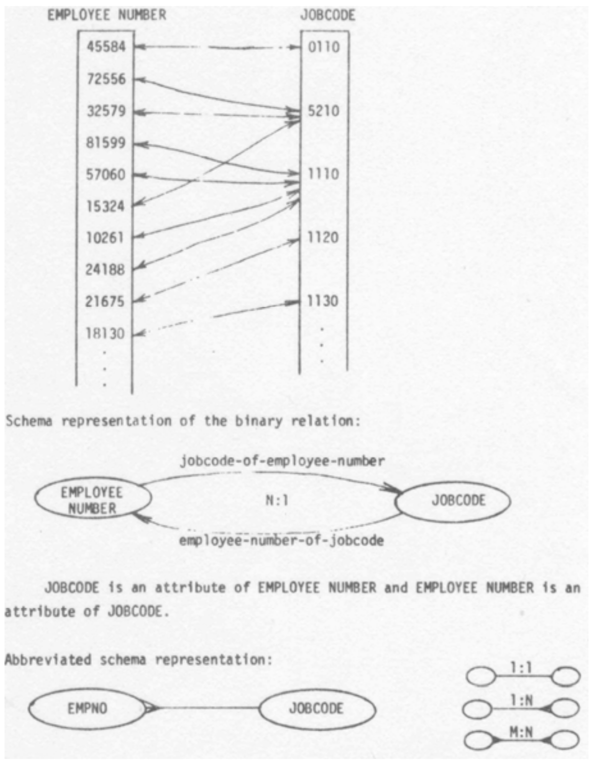

EMPLOYEE NUMBER
JOBCODE
45584
72556
32579
81599
57060
15324
10261
24188
21675
18130
0110
5210
1110
1120
1130
Schema representation of the binary relation:
jobcode-of-employee-number
EMPLOYEE NUMBER
N:1
JOBCODE
employee-number-of-jobcode
JOBCODE is an attribute of EMPLOYEE NUMBER and EMPLOYEE NUMBER is an attribute of JOBCODE.
Abbreviated schema representation:
EMPNO
JOBCODE
1:1
1:N
M:N

Figure 12. Representation of a Binary Relation.

When a single binary relation is defined between two entity domains, the association names can be dropped from the schema diagram. From the context of a given entity domain, the name of an entity domain connected via a binary relation can uniquely identify the attribute relationship. The schema representation of our full personnel-organizational data using this abbreviated notion is shown in Figure 13 along with a partial instance picture.

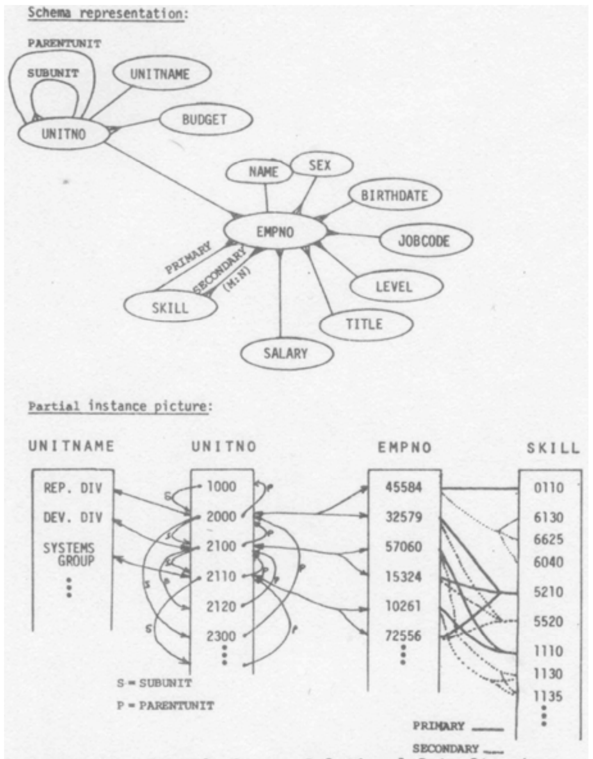


Figure 13. Example Binary Relational Data Structure.

SUMMARY

The taxonomy represented in this paper first divides basic data structure models into single file models and multifile models. The single file models are divided into flat file models and hierarchical models. A single flat file consists of a set of attributes which describe entities from a single entity class. Each attribute in a flat file structure can contain only one value instance. The hierarchical data structure model extends the flat file model by allowing multiple values for one attribute or for a group of attributes to be associated with a single entity instance. In our example, several employees can work in an organizational unit and each employee can possess multiple secondary skills.

The multifile data structure models consist of coordinated files, network, relational, and binary relational data structures. Multiple coordinated files are so called because they possess a common attribute (or set of attributes) and can therefore be sorted and processed in parallel as though they constituted a single logical file. The network and relational data models are very similar except that in some network models, access paths are defined coincidentally with logical relationships. This has led to considerable confusion. Furthermore, the criteria for the relationship is not usually explicit in the formal definition of a data structure. The relational data model requires that each file ("relation") be flat, have an explicitly defined key, and, if in third normal form, have no transitive or partial dependencies between the non-key attributes and the key. The binary relational model further breaks down the general n-ary relational model by explicitly and separately representing the relationship of each attribute with the key and any other attributes of other files.

This paper is excerpted from the author's textbook entitled Database Management: Objectives, System Functions, and Administration, Chapter 4, forthcoming from McGraw-Hill, (1986).

BIBLIOGRAPHY

[1] BACHMAN, Charles W., "Data Structure Diagrams," Data Base (1:2), 1969 summer, pages 4-10.

[2] CHEN, Peter Pin-Shan, "The Entity-Relationship Model -- Toward a Unified View of Data," ACM Transactions on Database Systems (1:1), 1976 March, pages 9-36.

[3] CODASYL Development Committee, "An Information Algebra," Communications of the ACM, 1962 April, pages 190-204.

[4] CODD, E. F., "A Relational Model of Data for Large Shared Data Banks," Communications of the ACM (13:6), 1970 June, pages 377-387.

[5] CODD, E. F., "Normalized Database Structure: A Brief Tutorial," Proceedings of the 1971 ACM-SIGFIDET Workshop "Data Description, Access, and Control," edited by E. F. Codd and A. L. Dean, New York: Association for Computing Machinery, 1971, pages 1-17.

[6] KERSCHBERG, L., A. KLUG, and D. TSICHRITZIS, "A Taxonomy of Data Models," Computer Systems Research Group, University of Toronto, Technical Report, CSRG-70, 1976 May, 75 pages.

[7] McGEE, William C., "The Property Classification Method of File Design and Processing," Communications of the ACM, 1962 August, pages 450-458.

[8] NIJSSEN, G. M., "Data Structuring in the DDL and Relational Data Model," Data Base Management, Proceedings of IFIP-TC2 Working Conference, Cargese, Corsica, 1974 April 1-5, edited by J. W. Klimbie and K. L. Koffeman, Amsterdam: North-Holland Publishing Co., (New York: American Elsevier), 1974, pages 363-379.

[9] NIJSSEN, G. M., "Two Major Flaws in the CODASYL DDL 1973 and Proposed Corrections," Information Systems (1:4), 1975, pages 115-132.

[10] OLLE, T. William, "A Practitioner's View of Relational Data Base Theory," FDT. Bulletin of ACM-SIGMOD (7:3-4), 1975, pages 29-43.

[11] RUSTIN, Randall, editor, Data Models: Data-Structure-Set Versus Relational, part of ACM-SIGMOD Workshop, 1974 May, New York: Association for Computing Machinery, 1975.

[12] SENKO, Michael E., "Data Description Language in the Context of a Multilevel Structured Description: DIAM II with FORAL," Data Base Description, Proceedings of IFIP-TC2 Working Conference, Belgium, 1975 January, edited by B. C. M. Douque and G. M. Nijssen, Amster-dam: North-Holland Publishing Co., (New York: American Elsevier), 1975, pages 239-258.

[13] SHARE Committee on Theory of Information Handling, "Report TIH-1," W. Orchard Hays, chairman, New York: SHARE, Inc., SHARE Secretary Distribution SSD-71, C-1663, 1959 July 15, 22 pages.

[14] SIBLEY, Edgar H., and Robert W. TAYLOR, "A Data Definition and Mapping Language," Communications of the ACM (16:12), 1973 December, pages 750-759.