NORMALIZED DATA BASE STRUCTURE: A BRIEF TUTORIAL

by

E. F. Codd
IBM Research Laboratory
San Jose, California

ABSTRACT:

Casual and other users of large formatted data bases need a simple tabular (relational) view of the data rather than a network or tree-structured view. This paper illustrates the removal of repeating groups, hierarchic and plex structures, and cross-referencing structures. Finally, the simplification of data base relations by normalization is discussed.

1. Introduction

In file management systems for formatted data there has been a distinction between the users' view of the data and its stored representation for many years. Its beginnings can be traced to early input-output systems which provided the service of combining or splitting physical records as they appeared on magnetic tape into larger or smaller record combinations, called logical records, which were made available to the application program. From these humble beginnings, the conceptual separation of logical and physical representations (sometimes called data structure and storage structure) has been given tremendous impetus by the heavy expense of reprogramming incurred by user installations whenever changes became necessary in the physical representation of their data (insertion of new fields, physical re-organization due to changes in traffic). Another important factor in this separation is the growth in demand for data base systems with interactive capabilities.

Several existing systems permit a variety of physical representations for a given logical structure. With such systems, if the logical structure is kept fixed, the physical representation may be changed without logically impairing the application programs - so long as these programs avoid direct reference to or direct modification of the physical representation. The complexity of the physical representations which these systems support is, perhaps, understandable, because these representations are selected in order to obtain high performance in access and update. What is less understandable is the trend toward more and more complexity in the data structures with which application programmers and terminal users directly interact. Surely, in the choice of logical data structures that a system is to support, there is one consideration of absolutely paramount importance - and that is the convenience of the majority of users.

It appears timely to investigate this question of user's convenience and to give special consideration to casual interactive users of data bases, since these users have been either ignored or treated as afterthoughts by the various committees developing standards.

2. Tables and Relations

To make formatted data bases readily accessible to users (especially casual users) who have little or no training in programming we must provide the simplest possible data structures and almost natural language. Remarkable progress has been made recently toward achieving efficient interpretation of almost natural language by Professor F. B. Thompson of the California Institute of Technology and his co-workers [1,2,3].

The casual user at a terminal often has occasion to require tables to be printed out or displayed. What could be a simpler, more universally needed, and more universally understood data structure than a table? Why not permit such users to view all the data in a data base in a tabular way?

It may be argued that in some applications the problems have an immediate natural formulation in terms of networks. This is true of some applications, such as studies of transportation networks, power-line networks, computer design, and the like. We shall call these network applications and consider their special needs later. The numerous data bases which reflect the daily operations and transactions of commercial and industrial enterprises are, for the most part, concerned with non-network applications. To impose a network structure on such data bases and force all users to view the data in network terms is to burden the majority of these users with unnecessary complexity.

Now, it may be contended that tables are inadequate data structures to support facile manipulation of relationships between and within the tables. Application of elementary operations on relations (operations such as projection, join, and division) shows that this contention is false. First, however, let us be more specific about the kind of tables we propose for the logical data base structure (remember we are not discussing the physical representation).

A table as normally understood is a rectangular array with the following properties:

P_1: it is column-homogeneous - in other words, in any selected column the items are all of the same kind, whereas items in different columns need not be of the same kind;

P_2: each item is a simple number or a character string (thus, for example, if we look at the item in any specified row and any specified column, we do not find a set of numbers or a repeating group).

For data base tables we add 3 more properties:

P_3: all rows of a table must be distinct (duplicate rows are not allowed);

P_4: the ordering of rows within a table is immaterial;

P_5: the columns of a table are assigned distinct names and the ordering of columns within a table is immaterial.

As a result of P_3, each row can be uniquely identified (or addressed) by its content. As a result of P_3 and P_4 together, this kind of table is what mathematicians call a relation. If the table has n columns, then it is a relation of degree n. We may apply the powerful operations of relational algebra or the expressions of relational calculus to derive many other relations from those contained in the data base. Property P_5 protects the user from having to memorize the column ordering in that table when selecting one or more columns of a data base table (a particularly important consideration for tables with many columns i.e., relations of high degree).

Fig. 1 exhibits a relation of degree 3 called COMPONENT. The triple (x,y,z) belongs to this relation if the part with part number x is a component of the part with part number y, and z units of x are needed to construct one unit of y. The headings SUB_P#, SUP_P#, Q stand for subordinate part number, superior part number and quantity respectively.

COMPONENT	(SUB_P#,	SUP_P#,	Q)
	1	5	3
	2	5	1
	13	5	20
	2	6	4
	5	6	2
	5	7	1
	7	4	2

Fig. 1: A Relation of Degree 3

When a formatted data base is viewed as a collection of time-varying relations of assorted degrees, we shall speak of this view as the relational model (see [6] for more details). This phrase is used rather than relational data structure, because to many computer-oriented people the latter phrase connotes a structure containing explicit links.

3. Additional Data Structure

We shall now consider three of the more important structural concepts supported by current data base systems. Our purpose is to suggest that these concepts provide no additional logical capability beyond that of the relational model and to question their inclusion in the data models of future data base systems. It is important to remember that we are not making a case for or against any physical storage structures in this paper. The examples below are designed to illustrate normalization principles in section 4 as well as structure simplification in this section.

3.1 Repeating Groups

Consider data concerning parts and the projects which use those parts. For each distinct kind of part, the part number (P#), part description (PD), and quantity on hand (QOH) are recorded together with project data for each project J using that kind of part. The project data consists of project number (J#), project description (JD), project manager number (JM#), and quantity (QC) of this part committed to this project. Fig. 2 illustrates in CODASYL fashion a compound group schema (for parts) which contains a repeating group schema (for projects) --- see McGee's Chapter 2 in [4].

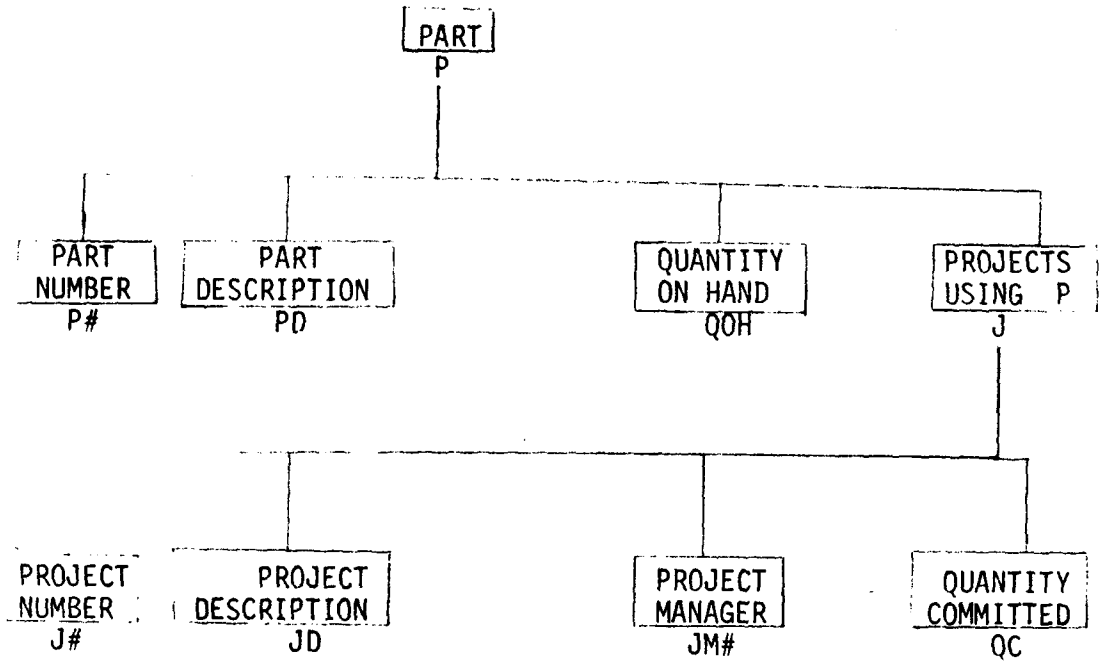


Fig. 2: Projects as a Repeating Group within Parts

The simplest way to eliminate the repeating group structure without losing information is to split the schema into two separate schemas as in Fig. 3. This split exploits the fact that P# uniquely identifies parts. We shall see later why a three-way split would be preferable.

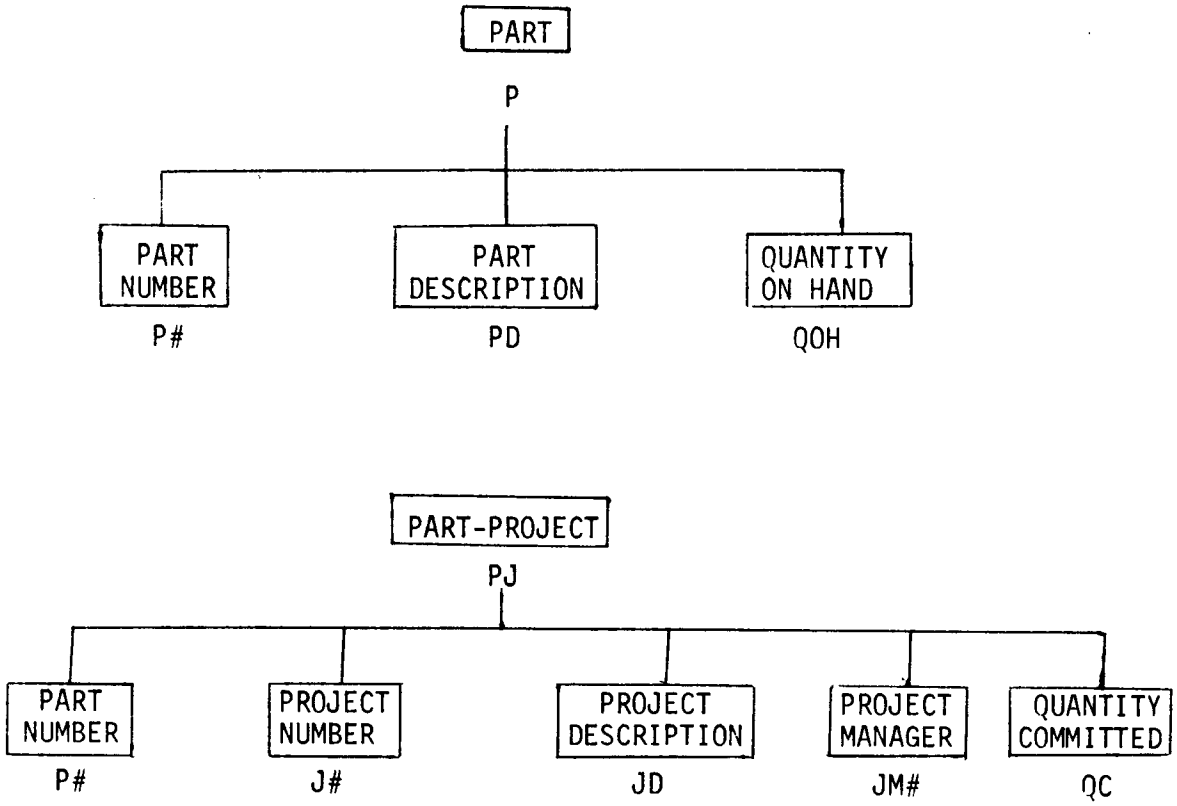


Fig. 3: Elimination of Repeating Group in Fig. 2

The schemas of Fig. 3 can be represented in concise relational notation as follows:

P(P#,PD,QOH)
PJ(P#,J#,JD,JM#,QC)

The more complicated cases in which two or more repeating groups are subordinate to a common group, or a repeating group is subordinate to another repeating group (see Fig. 2-24 of [4] for example), can be handled in much the same way as the simple case. At least one split is needed for each repeating group.

3.2 Hierarchic and Plex Structures

The hierarchic structure illustrated in Fig. 2-25 of [4] can be eliminated by the same splitting procedure used for repeating groups. The major operational difference between the repeating group structure (Fig. 2-24) and the hierarchic structure (Fig. 2-25) appears to be one of addressability: an instance of the group entry schema has to be retrieved as a complete unit, whereas certain portions of an instance of the tree entry schema can be retrieved without retrieving the whole instance. This distinction is an example of the intrusion in existing systems of performance considerations into the user's conception of the data.

The procedure for eliminating plex structures is only slightly more complicated than that for trees. Consider data concerning suppliers and parts. Two relationships between suppliers and parts are reflected in the data base: the CANDIDATE relationship holds between supplier x and part y if x is capable of supplying y; the ACTUAL relation holds between supplier x and part y if x is actually supplying part y (i.e., part y is on order from supplier x). This data is represented as a plex entry schema in Fig. 4. Note the use of directed lines to represent the relationships CANDIDATE and ACTUAL. This practice is prevalent in much of the contemporary literature in this field (including [4], Fig. 2-26 etc.). This is an example of yet another intrusion of physical representation concepts (in this case, pointers) into the user's view of the data.

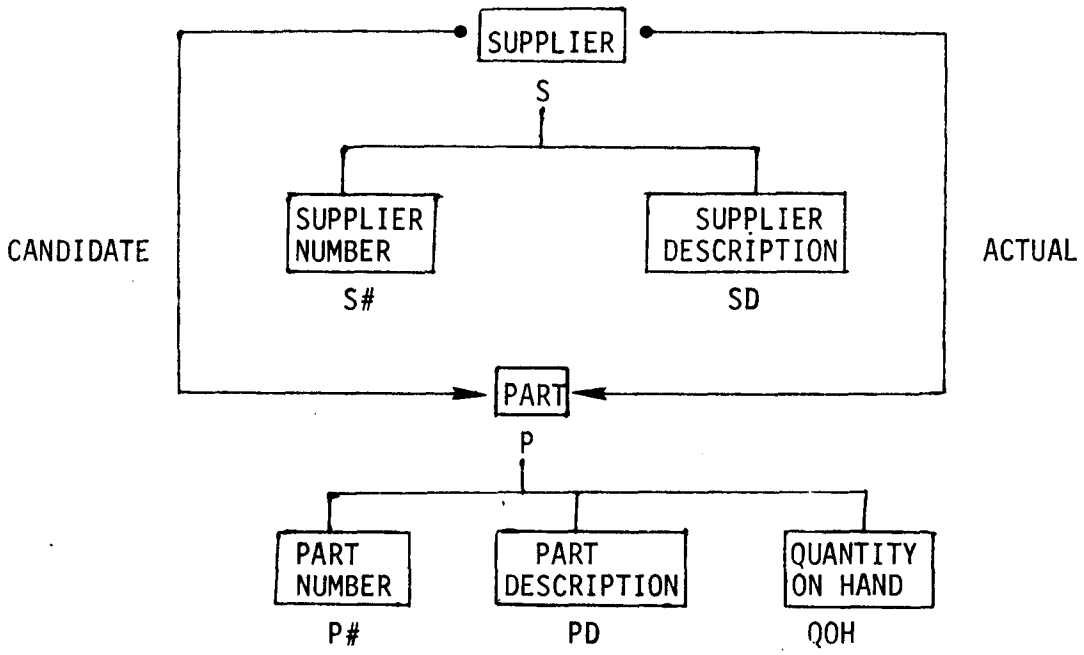


Fig. 4: A Plex Entry Schema

Each directional relationship* in the plex structure of Fig. 4 can be split off and represented as a binary relation in tabular fashion. Taking advantage of the fact that S# uniquely identifies suppliers and P# uniquely identifies parts, we obtain the four separate schemas of Fig. 5.

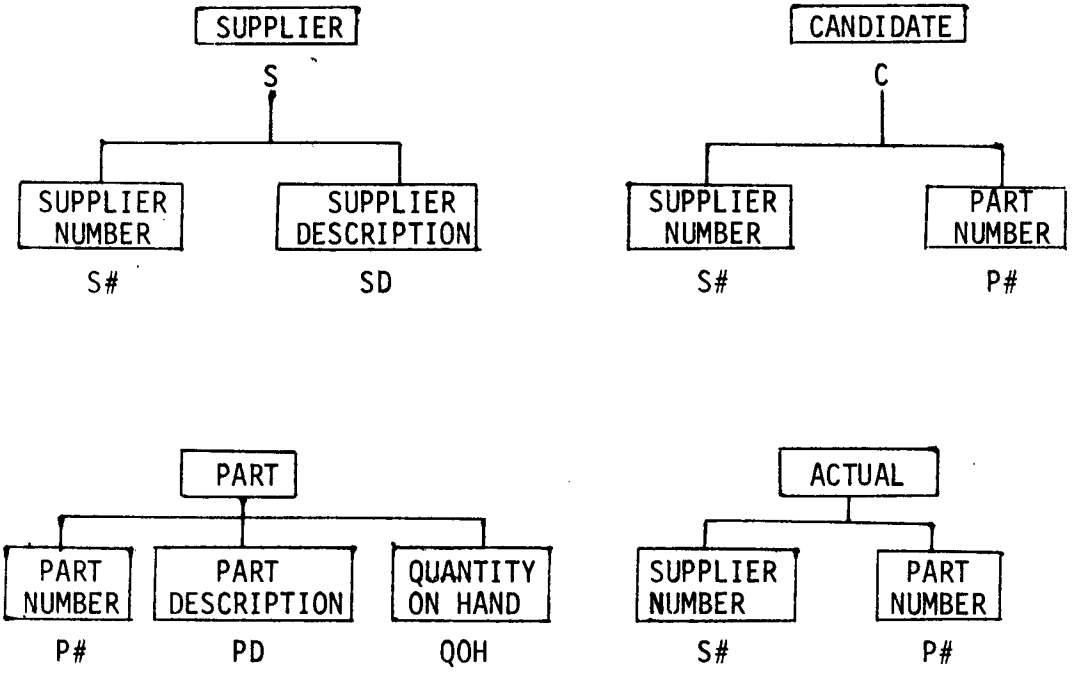


Fig. 5: Elimination of Plex Structure in Fig. 4

* The term group relation is used in [4].

3.3 Cross-Referencing Structure

Consider the schema of Fig. 6 for data concerning projects and employees. The manager of a project is an employee. Thus, his employee serial number appears in at least two places: in the E# column of the EMPLOYEE relation and in the JM# column of the PROJECT relation. The connection from JM# to E# (shown as a dotted line in Fig. 6) is intended to convey the possibility of associating with any given project manager his attributes when he is regarded as an employee. Similarly, the connection from DM# to E# is intended to convey the possibility of associating with any given department manager his attributes when he is regarded as an employee.

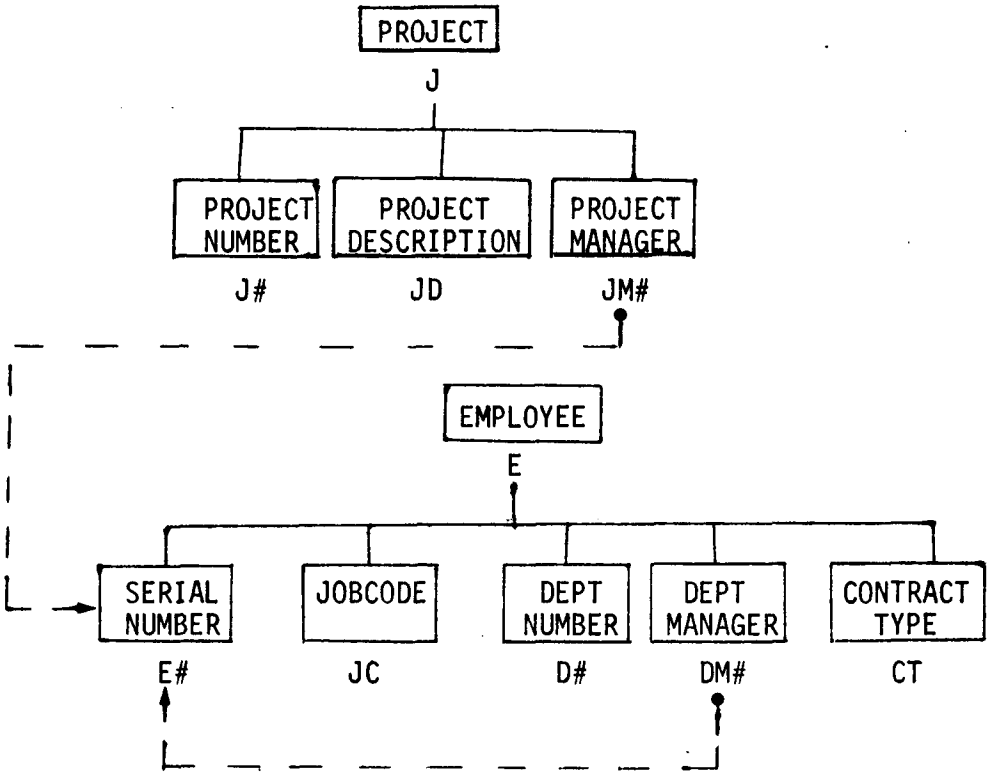


Fig. 6: Cross-Referencing Structure

The link between DM# and E# is one-way, indicating to the user that there is a search path he may follow from DM# to E# if he invokes appropriate path-following operations in his application program. Absence of a link from E# to DM# implies absence of the corresponding search path.

This kind of cross-referencing structure is both unnecessary and undesirable in the user's model. It is unnecessary because cross-referencing can be quite adequately handled by appropriate language (see sections 3.6 through 3.13 of [5]). It is undesirable because there are so many other cross referencing associations that can be exploited, and to depict them all in a structural way would result in presenting users with an impossibly tangled and complicated set of connections. For instance, if one considers employee serial numbers alone in this example, we would have to exhibit the six links in Fig. 7 to be complete.

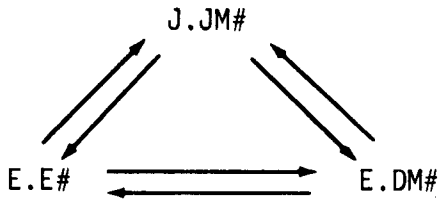


Fig. 7: Proliferation of Cross-Referencing Links

To detect nonsensical cross-references in user requests, the data base management system should store information concerning which pairs of attributes are comparable for cross-referencing purposes. The concept of system-convertible units fulfils this requirement in a very simple and general manner (see section 3.6 of [5]).

3.4 Comments on Linked Structures

Except in network applications, links should not be employed in the user's data model for the following reasons:

1) a directed link from A to B implies that B is accessible from A, but not vice versa unless a second directed link goes from B to A (asymmetry in access);

2) links imply different treatment for relations of degree 2 from that accorded relations of greater degree (i.e., a different set of accessing operations, modifying operations, authorization options, etc.), thus, greatly increasing system complexity and complexity of use;

3) growth of a relation from degree 2 to degree 3 cannot be gracefully accommodated if the degree 2 version is represented by a link;

4) links offer the temptation to represent a ternary relation as two binary relations with consequences that, depending on circumstances, can be fatal (see connection trap in [6]).

Network applications can be supported by adding a specialized front end to a general purpose, relational data base management system.

4. Normalization of Relations

This discussion of normalization is highly informal and introductory. For a more thorough treatment, see [7,8].

Normalization is a step-by-step reversible process of replacing a given collection of relations by successive collections in which the relations have a progressively simpler and more regular structure. The simplifying process is based on non-statistical criteria. The reversibility guarantees that we can recover the original collection of relations and therefore have not lost any information. The objectives of normalization are:

1) To make it feasible to tabulate any relation in the data base (property P_2 in particular);

2) To obtain a powerful retrieval capability by means of a simpler collection of relational operations than would otherwise be necessary;

3) To free the collection of relations from undesirable insertion, update and deletion dependencies;

4) To reduce the need for restructuring the collection of relations as new types of data are introduced, and thus increase the life span of application programs;

5) To make the relational model more informative to users;

6) To make the collection of relations neutral to the query statistics, where these statistics are liable to change as time goes by.

The first two objectives apply only to the first step (conversion to first normal form). The last four objectives apply to all normalization steps.

4.1 Conversion to First Normal Form

To illustrate the first step of normalization, consider the relation P whose schema is exhibited in Fig. 8 along with a partial snapshot (some values occuring at some instant of time). The symbols in this figure have the same meaning as in Fig. 2. The primary key P# is underlined.

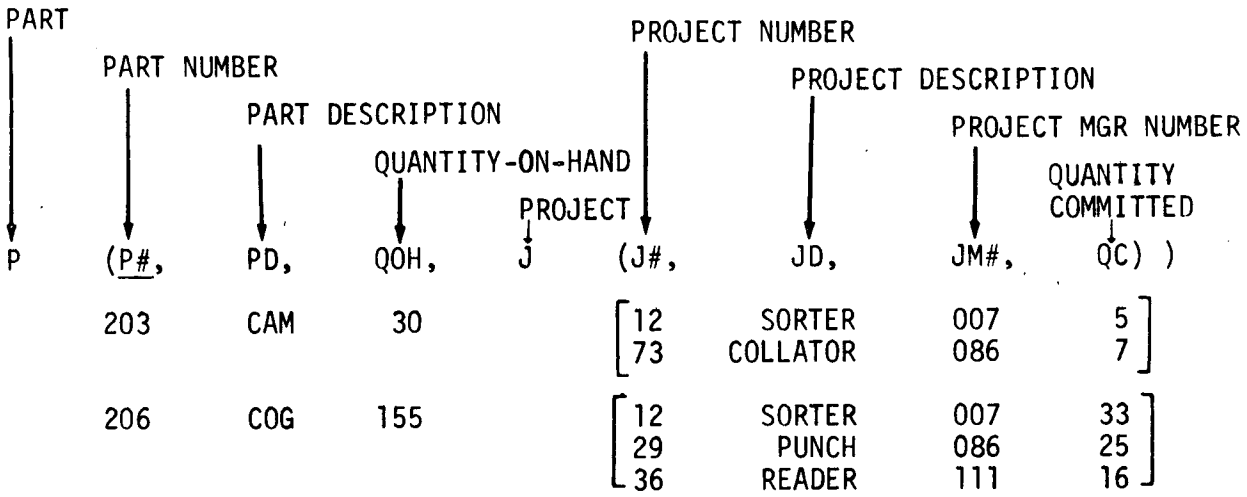


Fig. 8: An Unnormalized Relation

It will be observed that the attributes P#, PD, QOH all have simple values while the attribute J has relations as its values. Relation P is said to be unnormalized. The first step of normalization is to split P just as we split the structure in Fig. 2. Accordingly, we obtain the relations P_1, PJ_1 as in Fig. 9. Note that the primary keys are P# and (P#,J#) respectively.

P_1 (P#, PD, QOH)

P#	PD	QOH
203	CAM	30
206	COG	155

PJ_1 (P#, J#, JD, JM#, QC)

P#	J#	JD	JM#	QC
203	12	SORTER	007	5
203	73	COLLATOR	086	7
206	12	SORTER	007	33
206	29	PUNCH	086	25
206	36	READER	111	16

Fig. 9: Relations in First Normal Form

This normalization step is essentially the same procedure used to remove repeating groups and hierarchic structure in sections 3.1, 3.2.

4.2 Conversion to Second Normal Form

Now, we observe that in the relation PJ_1 there are some irregularities in the dependence of attributes upon the primary key (P#,J#). Informally, we observe that QC is an attribute of the entire key, while JD, JM#, are attributes of the J# component of the primary key. These irregularities give rise to the following anomalies:

1) unless fictitious part numbers are introduced, data concerning a new project cannot be recorded until the project uses some parts (an insertion anomaly);

2) if only one kind of part remains in use by a project, deletion of data concerning that part causes deletion of the last remaining information on that project, while previous deletions did not have this consequence (a deletion anomaly);

3) if a change is made to the value of an attribute of a project (e.g., the manager's serial number JM#), the number of copies of this information to be updated in the data model depends on the number of parts in use by that project at the instant the update is performed (an update anomaly).

The type of irregularity illustrated in this example can be removed by again splitting the PART-PROJECT relation PJ_1 to obtain the relations exhibited in Fig. 10. Note that the quantity QC of a part committed to a project is an attribute of the combination of part number P# and project number J#. Hence, it belongs in the relation PJ_2, not in J_2. As usual, the primary key of each relation is underlined.

P_2 (P#, PD, QOH)

P#	PD	QOH
203	CAM	30
206	COG	155

PJ_2 (P#, J#, QC)

P#	J#	QC
203	12	5
203	73	7
206	12	33
206	29	25
206	36	16

(Fig. 10 continued on next page)

J_2	(J#,	JD,	JM#)
	12	SORTER	007
	73	COLLATOR	086
	29	PUNCH	086
	36	READER	111

Fig. 10: Relations in Second Normal Form

The splitting needed to convert to second normal form is of the same kind that data base growth tends to force.

4.3 Conversion to Third Normal Form

To illustrate the final step of normalization, we consider the EMPLOYEE relation of Fig. 6. A sample snapshot of this relation is exhibited in Fig. 11 below.

EMPLOYEE

EMPLOYEE NUMBER

JOBCODE

DEPT NUMBER

DEPT MGR NUMBER

CONTRACT TYPE

E	(E#,	JC,	D#,	DM#,	CT)
	1	a	x	11	g
	2	c	x	11	g
	3	a	y	12	n
	4	b	x	11	g
	5	b	y	12	n
	6	c	y	12	n
	7	a	z	13	n
	8	c	z	13	n

Fig. 11: A Relation not in Third Normal Form

Although this relation does not possess the kind of irregularity discussed in section 4.2, some of its attributes are transitively dependent on others, and this gives rise to anomalies similar to those discussed above. Take the attribute CT for example. This is dependent on D# which, in turn, is dependent on E#.

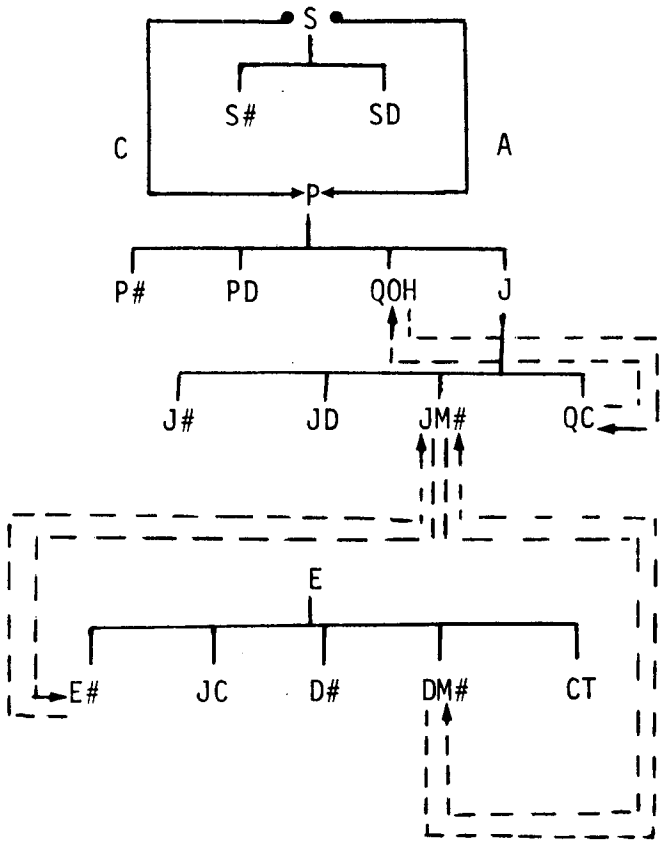
Once again, the irregularities and their associated anomalies (see [7]) are removable by splitting the EMPLOYEE relation into the two relations E_3, D_3 shown in Fig. 12.

E_3(E#,	JC,	D#)	D_3(D#,	M#,	CT)
1	a	x	x	11	g
2	c	x	y	12	n
3	a	y	z	13	n
4	b	x			
5	b	y			
6	c	y			
7	a	z			
8	c	z			

Fig. 12: Relations in Third Normal Form

As with conversion to second normal form, the splitting needed to convert to third normal form is of the same kind that data base growth tends to force.

Finally, we compare in Figs. 13, 14 two schemas for a common assemblage of information: the first schema is an example of common practice today, the second is in third normal form. In the latter form all attributes not underlined are dependent in a simple and direct manner on the corresponding underlined attributes (primary keys). This clean structure can be more readily understood, interrogated, modified, and authorization-controlled.



C	CANDIDATE
A	ACTUAL
S	SUPPLIER
S#	SUPPLIER NUMBER
SD	SUPPLIER DESCRIPTION
P	PART
P#	PART NUMBER
PD	PART DESCRIPTION
QOH	QUANTITY-ON-HAND
J	PROJECT
J#	PROJECT NUMBER
JD	PROJECT DESCRIPTION
JM#	PROJECT MGR NUMBER
QC	QUANTITY COMMITTED
E	EMPLOYEE
E#	EMPLOYEE NUMBER
JC	JOBCODE
D#	DEPT NUMBER
DM#	DEPT MGR NUMBER
CT	CONTRACT TYPE

Key to Diagram

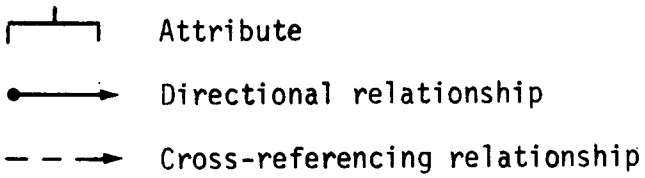


Fig. 13: Network Schema

PJ COMMITMENT	D DEPARTMENT
S(S#,SD)	A(S#,P#)
P(P#,PD,QOH)	C(S#,P#)
J(J#,JD,JM#)	D(D#,DM#,CT)
PJ(P#,J#,QC)	E(E#,JC,D#)

Fig. 14: Relational Schema in Third Normal Form

References

1. F. B. Thompson, P. C. Lockemann, B. H. Dostert, R. S. Deverill, "REL: A Rapidly Extensible Language System", Proc. Nat. ACM August 1969 San Francisco.

2. B. H. Dostert, F. B. Thompson, "How Features Resolve Syntactic Ambiguity", Proceedings of Symposium on Information Storage and Retrieval, University of Maryland, April 1-2, 1971.

3. B. H. Dostert, F. B. Thompson "Syntactic Analysis in REL English", 1971 International Meeting in Computational Linguistics, Debrecen, Hungary September 1971.

4. Codasyl Systems Committee, "Feature Analysis of Generalized Data Base Management Systems", May 1971, obtainable from ACM HQ, New York, New York.

5. E. F. Codd, "A Data Base Sublanguage founded on the Relational Calculus", Proc. 1971 ACM-SIGFIDET Workshop on Data Description, Access and Control, to be available from ACM HQ, 1972.

6. E. F. Codd, "A Relational Model of Data for Large Shared Data Banks", Comm. ACM 13 6, June 1970, 377-387.

7. E. F. Codd, "Further Normalization of the Data Base Relational Model", Courant Computer Science Symposia 6, "Data Base Systems", New York City, May 24-25, 1971, to be published by Prentice-Hall.

8. I. J. Heath, "Unacceptable File Operations in a Relational Data Base", Proc. 1971 ACM-SIGFIDET Workshop on Data Descirption, Access and Control, to be available from ACM HQ, 1972.