# A PRACTITIONER'S VIEW OF RELATIONAL DATA BASE THEORY

T. William Olle
27 Blackwood Close
West Byfleet
Surrey KT 14 6PP
England

Author's note (16 January 1976)

The following paper represents a lightly edited version of Chapter 23 of my tutorial text "The CODASYL Approach to Data Base Management" to be published by John Wiley & Sons. This explains the tutorial style of the prose. The main aim of the chapter, however, is to analyze the differences and similarities not only between the CODASYL approach and the relational approach, but also between the latter approach, and IMS and TOTAL. These two systems are also each given a chapter in the textbook.

Analysis shows that CODASYL and relationalism are in many ways closer to each other than either is to TOTAL or IMS. The acceptability of the constraints implied in the relational approach is accessed from a practitioner's perspective.

Any comments on this analysis, either in the open pages of FDT or any private to the author, will be welcomed and addressed through the same media.

## 1. Background

There are three well known approaches to data base management which are commercially viable and in wide use among commercial installations all over the world. These are CODASYL, IMS and TOTAL. In this paper, we turn attention to an approach to data base management which is being widely studied among the academic fraternity and among the more research oriented in government and commercial organizations.

The relational approach evolved in a series of papers by Codd (1,2) in 1971, although the number of papers on this topic which have been published is quite vast. For a good comprehensive overview of what relationalism is all about, a recent book by Date (6) is recommended.

It must be emphasized that the relational approach is essentially a theory on which a number of experimental systems have been based. Rather than select one of these systems for presentation, we shall in this paper present the elements of the theory in such a way that explicit comparison with the CODASYL approach is facilitated, as well as a lesser degree of comparison with IMS and TOTAL.

In the relational approach unlike the other three approaches, there is no data definition language per se, but simply a way of looking at data. There is, however, a data manipulation language which Codd calls a "data sub-language". In fact, this term is felt to be a better term than "data manipulation language". It conveys the implication of an extension to an existing programming language so much more effectively. Codd's data sub-language is called ALPHA (2).

## 2. Relational Terminology

The first task is to equate the terminology used in discussion of the relational approach with that of the CODASYL approach. It says something about the state of comparative understanding of these two approaches when one considers that even a terminology equivalence table can be contentious. Bearing this in mind, we present in Figure 1 the equivalences to be used herein, and then discuss each in turn.

| Relational | CODASYL |
|---|---|
| attribute | data item |
| value | value |
| domain | - |
| relation | record type |
| tuple | record (occurrence) |
| table | file |
| inter-relation dependency | set type |
| degree of a relation | number of items in record type |

Figure 1 Terminology equivalences

### 2.1 Attribute, Domain and Data Item

The word domain comes from classical set theory. It implies that in a mathematical set there may be a number of elements and in a given situation some of these may be collectively referred to as a domain. It is also useful to think of a domain as a value set, such as M and F in the domain SEX. This idea of a predefined value set is not one which is implied in the term "data item" (specifically an elementary item) widely used in all commercial data processing.

Before leaving this topic, it is useful to repeat what Codd (3) writes on the topic of domains and fields (i.e., items):

> "It is the attribute concept rather than domain which is in close correspondence with field. The domain concept is missing from traditional data processing including the CODASYL DBTG approach.

> When a relational data base is defined, one of the first steps is to identify the underlying domains: that is, semantically distinct pools of non-decomposable data items from which tuples may be dynamically created. ... If attributes A, B ... take their values from a common domain, then it is semantically permissible to compare values of A, B for equality. ..."

As an example, we could have a domain called NAME-OF-PERSON and two attributes, CUSTOMER-NAME and EMPLOYEE-NAME. These two attributes are probably distinct and separate, but if they are not, it would be possible to test whether a given customer is also an employee.

This kind of precise thinking is not inherent in use of the term "data item" and the concept of a domain as distinct from an attribute is one which should be promulgated. The solution to the unfortunately loose thinking which surrounds the term "data item" is not to change the term, but rather to encourage recognition of the importance of defining the domain of values for an item prior to specifying the record types to which the item belongs.

## 2.2 Relations and Record Types

One major problem in conventional data processing terminology stems from the imprecise use of "record type" and "record". Nine people out of ten will use "record" when they mean "record type". In conventional data processing, this did not matter so much. In a data base oriented environment, it can cause communication and comprehension problems.

If the problem in the use of conventional data processing terminology is avoided, then "relation" and "record type" can be seen to be equivalent.

## 2.3 Tuples and Records

The word "record" in early FORTRAN carried an implication of a contiguous string of stored data - a very physical meaning. In COBOL, it has a far more logical meaning and there seems to be no inherent problem with equating record to tuple. It should be noted that tuple is an abbreviation for "n-tuple".

## 2.4 Tables and Files

A collection of tuples of a given relation is called a tuple. The word file in COBOL has a broader meaning in that it embraces the concept of multiple record type files and also it implies something which is stored on secondary storage, even though the CODASYL COBOL Journal of Development glossary defines a file quite simply as "a collection of records".

The CODASYL data base specifications, rightly or wrongly, avoid the word file. Perhaps if the specifications had been built on the terminology base of a file instead of a set occurrence, some of its concepts might have been easier to grasp by those with experience in conventional techniques.

The word "table" (which has an intra-record type meaning in COBOL) is used in relational theory to refer to a collection of tuples of a given relation. There are rules which these tuples must adhere to, but these are some of the facets of the theory.

## 2.5 Set Type and Inter-relation Dependency

Were are now entering into some of the more tricky aspects of a comparison between relational theory and the CODASYL approach, and it is appropriate to postpone discussion of this particular equivalence until the aspects of the theory have been presented.

## 2.6 Degree of a Relation

If a relation consists of six domains, it is said to have a degree of six. If, for the sake of argument, one of these domains was represented by two attributes, it is not clear whether the degree of the relation would be six or seven. In fact, it is probably seven because the subtle relationship between domains and attributes does not appear to have been thought through. If it is indeed seven, then we can safely say that the degree of a relation is the same as the number of elementary data items in a record type.

# 3. Relational Theory Rules

Although relational theory is based on pure mathematical set theory, nowhere in the vast literature is the theory presented as a set of lemmas. (A lemma is an assumed or demonstrated proposition used in argument or proof). This style of argument is inherent in many facets of pure mathematics and we will therefore present relational theory here as a series of lemmas using both sets of terminology. For clarity, these are presented in a form which facilitates lemma by lemma comparison.

| | Relational terminology | COBOL plus DBTG terminology |
|---|---|---|
| 1. | A non-decomposable element of named data is called an attribute. | A non-decomposable element named data is called an elementary data item (or data item or item). |
| 2. | Each attribute has a collection of predefinable values which are called its domain. | Each data item has a collection of values which may be predefined. |

| | |
|---|---|
| 3. A relation consists of a number of attributes. | A record type consists of a number of data items. |
| 4. The number of attributes in a relation is called the degree of the relation. | The number of data items in a record type could be called the degree of the record type. |
| 5. For a given data base there may be several relations defined. | A data base schema contains several record types. |
| 6. A number of tuples may exist in the data base for each relation. | Any number of records of each record type may exist in the data base. |
| 7. The collection of all the tuples of a given relation is called a table. | The collection of all the records of a given record type is called a file. |
| 8. The ordering of the tuples in a table is immaterial. | The ordering of the records in a file is immaterial. |
| 9. The ordering of the attribute values in a tuple must correspond to the ordering of the attributes in the relation. | The ordering of the data item values in a record must correspond to the ordering of the items in the record type. |
| 10. Each relation must contain an attribute which serves as a unique primary key. | Each record type must contain an item which serves as a unique primary key. |
| 11. An inter-relation dependency may exist between any two relations in a data base. | A relationship (called a set type relationship) may exist between any two record types in a data base. |
| 12. When an inter-relation dependency exists between two relations, one relation is the parent and the other is the subordinate. | In a set type relationship, one record type is the owner and the other is a member. |
| 13. For an inter-relation dependency to exist, the attribute which serves as the primary key in the parent must also be an attribute in the member. | In a set type relationship, the item which serves as the primary key of the owner record type must be an item in the member record type. |

14. If a relation is the subordinate in one inter-relation dependency and the parent in another, then the primary key attribute of the parent of the first dependency must be propagated through to the subordinate of the second dependency.

    If a record type is the member in one set type and the owner in another, then the primary key item of the first set types owner must be propagated through its member to the second set type's member.

A careful study of these rules indicates that relational theory calls for a fairly rigid set of constraints in a relation data base. It is useful to summarize these constraints in CODASYL terms.

1. Each record type contains only elementary data items.

2. Each record type must have a unique prime key.

3. To be able to define a set type, the prime key of the owner must also be in the member.

It is further useful to note the CODASYL Schema DDL concepts which are explicitly prohibited in a relational data base.

1. Repeating groups.

2. Control over physical contiguity of records of the same or different types.

3. Predefined set order.

4. Definition of search key.

5. Multi-member set types.

6. Storage class and removal class.

7. Set selection algorithms.

Finally, with specific reference to IMS and TOTAL, it is very important to understand that relational theory does not call for either of the major restrictions noted for these two systems. There is no assertion in the theory that a record type may not have more than two owner record types, which is felt to be a major problem with IMS. Furthermore, relational theory does not dictate that a record type cannot serve as an owner in one set type and a member in another set type at the same time. This is the main problem with the structuring facilities in TOTAL.

## 4. Relational and CODASYL Network

There has been considerable debate on the relative merits of the relational approach and the CODASYL approach (4). Interestingly, the CODASYL approach is invariably identified as "the network approach" in the literature. A quote from one of Codd's early papers (5) may explain this.

> "It may be argued that in some applications, the problems have been 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. ... The numerous data bases which reflect the daily operation and transaction 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 burdern the majority of users with unnecessary complexity".

This kind of unfortunate thinking tends to convey the impression that there is something inherently undesirable about networks. It was Date (6), however, who identified the CODASYL approach as "the network approach" in the same paper comparing it with the "relational model".

Taking writings of Codd and Date together, one could be forgiven for concluding not only that networks were undesirable, but also that the relational approach does not cater for them. Nothing could be further from the truth.

It is unfortunate that the simple little examples used to illustrate the principles of relational theory tend to be based on two or three record types such as those depicted using Bachman diagrams in Figure 2.

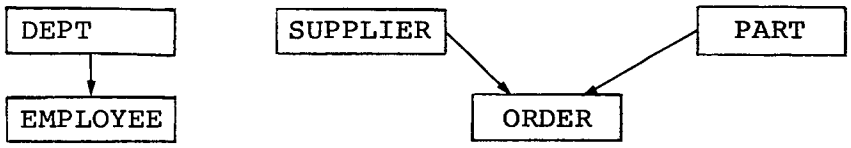


Figure 2 Simple data bases.

It is also unfortunate that the papers on relational theory have, for one reason or another, not used Bachman diagrams (or any other graphic formalism) for depicting the structure of a data base.

A study of the extensive writing on relational theory leads to the following assertion. The logical data structure which may be represented in a relational data base may be of the same degree of complexity as that allowed for in the CODASYL proposals for a data base management system - with the sole exception of the CODASYL facility to represent cyclic structures. This assertion cannot be made for IMS or TOTAL.

## 5. Use of Keys

Relational theory requires each record type in the data base to have a unique prime key. The CODASYL approach merely allows this as a possibility. It is achieved by giving the record type a location mode of CALC with "DUPLICATES NOT ALLOWED" option declared. It must be recalled that the latest thinking is to remove any connotation of randomizing from the use of the term "calc".

It must be noted that is not possible to assign a prime key to each record type in a TOTAL data base or in an IMS data base. Actually, one can achieve the objective of a prime key for every record type in both of these systems at the cost of not using set type relationships. Few users would consider this a worthwile price to pay.

CODASYL allows the definition of search keys or secondary keys. These are not catered for explicitly in relational theory.

## 6. Set Types and Their Properties

The set type capability is buried deep in the prose of the writings on relational theory. The term "inter-relation dependency" does not appear to be used, but Date (7) uses the term "intra-relation dependency" and also the term "inter-relation consistency".

### 6.1 Defining a Relationship

What happens in each of the commercially viable approaches, CODASYL, IMS and TOTAL, is the following. Two record types, A and B, are defined, and then in a separate piece of syntax a relationship between A and B is defined. In CODASYL and TOTAL there may be one or more relationships between A and B, and it is therefore necessary to name them. In the IMS, there may not be more than one relationship between A and B, and no naming of the relationship is allowed.

In relational theory, record types are defined, and any relationship between them is implicit in the attributes in each record type. For instance in the simple case of departments and employees, the record types may be as follows:

```
DEPT
  DEPT-NO
  DEPT-NAME
  MANAGER
  LOCATION


EMPLOYEE
  EMPLOYEE-NUMBER
  EMPLOYEE-NAME
  DEPT-NO
  SALARY
```

The repetition of the data item DEPT-NO in both record types coupled with the fact that DEPT-NO is the prime key of the record type has the effect of defining a set type relationship between the two record types. In other words, the definition of a set type relationship in a relational data base is implicit in the items defined in the record types as opposed to being explicit.

It is of great significance that TOTAL uses exactly this method to handle set type relationships with its control items whereas IMS does not even allow it as an option.

It is also noteworthy that the early CODASYL specifications did not cater for this kind of approach, and hence commercially available implementations tend not to support it. The most recent CODASYL specifications do not permit this as an alternative.

## 6.2 Set Order

One of the more contentious properties of a set type is set order. There is no concept of set order in relational theory. In fact, in his initial paper (1), Codd affirms:

> "Those application programs which take advantage to the stored ordering of a file are likely to fail to operate correctly if for some reason it becomes necessary to replace that ordering by a different one".

One cannot argue with this truism. However, the evil lies not in defining an order, but in taking advantage of it. Using reductio ad absurdum arguments, one could assert that an application program should not take advantage of any factor defined in the data definition because in theory it might change. Where does one draw the line?

Relational theory has had its impact, directly or indirectly, on the CODASYL proposals for set ordering. The set order clause was not removed, but the option for the data administrator to define a set order as immaterial was added. This option allows another

capability proposed in relational theory to be achieved in a CODASYL system.

### 6.3 Storage Class

Finally, the CODASYL concept of storage class must be examined. Relational theory does not admit of such a concept, which is another way of saying that the storage class is always automatic. A relational system would in effect behave in the same way as TOTAL. When a member record is stored, the value of the set item for any set type in which it participates must correspond to that in one of the owner records in the data base.

### 6.4 Set Mode

We have deliberately left set mode until last. The references to chains and pointer arrays in the early CODASYL papers gave rise to some of the early criticism that the proposals were unnecessarily involved with questions of how the data should be stored. The DDLC reacted well to such comments and the set mode clause was removed from the Schema DDL. The chain concept had tended to dominate the early image of the CODASYL proposals. It is now realized that the important capability is that of defining 1:M relationship between record types rather than the techniques for representing such relationships in direct access storage.

Relational theory steps back even from the explicit definition of the relationship, calling in effect for an implicit definition in terms of the data items in the record types. Nevertheless, supporters of the relational approach have been heard to admit verbally that some kind of set mode must be provided behind the scenes. As in the case of the set order, it should not be a factor of which the application programmer can take advantage.

## 7. Practically of Relational Theory

Three aspects of relational theory can be examined in terms of their impact on the practitioner. These are:

1. Unique prime key for every record type

2. Means of representing set type relationships

3. Propagation of prime keys down structure.

Before discussing these aspects, the rationale for them must be analyzed. It is purely and simply a question of data independence and the design of a data manipulation language which will maximize that data independence. The question to be asked is whether these rigid rules are in fact justified in order to achieve the data independence.

## 7.1 Unique Prime Key

Relational theory calls for every record type to have a unique prime key. In most cases this is justifiable and no one could argue against it. There are few instances where it appears contrived. One example is depicted in Figure 3.

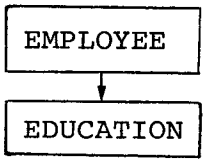


Figure 3 Discussion of unique prime key.

Employees normally have employee numbers and a prime key in the record type EMPLOYEE is perfectly acceptable. However, in the other record type, namely EDUCATION, a unique prime key seems to be rather contrived. It is unlikely that any use of this modest data base would call for an access to an EDUCATION record independent of the EMPLOYEE record to which it belongs.

Insisting on a unique prime key is a good idea most of the time. Certainly, it is important to be able to define such a key for any record type in the data base (not possible in TOTAL and IMS), but to have to do so seems unnecessary.

## 7.2 Representation of Set Type Relationships

The relational way of handling set type relationships is simple and effective. It is achieved at the expense of duplicating an item (and therefore its concomitant values) in two record types. The approach is easy to understand and to use. It is without question commercially viable, as it is the only alternative allowed in TOTAL.

By contrast, the CODASYL approaches to this problem are provided for in the plethora of set selection options, and only recently has one of these corresponded to that espoused by relational theory.

We have no hesitation in endorsing the relational approach in this respect as being preferable in almost every case. It is perhaps the second major disadvantage of IMS that this kind of facility is not provided.

## 7.3 Propagation of Prime Keys

The illustration of this practice is shown in an example borrowed from Codd's initial paper (1). In subsequent papers on relational theory, the examples chosen are all two level ones.

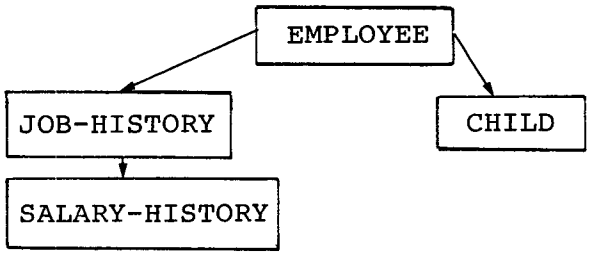


Figure 23.4 Three level structure.

If the four record types were to adhere to relational theory, they would be in the following form:

EMPLOYEE (EMPNO, NAME, BIRTHDATE)

JOB-HISTORY (EMPNO, JOBDATE, TITLE)

SALARY-HISTORY (EMPNO, JOBDATE, SALARYDATE, SALARY)

CHILDREN (EMPNO, CHILDNAME, BIRTHDATE)

This way of specifying the record names and the items belonging to each is widely used in papers on relational theory. The underlining of an item name is the recognized way of indicating that this item serves as the prime key in the record type, or as part of the prime key. These record types are collectively expressed in what relational theory refers to as third normal form - a term so far not introduced into this paper although its implications have been taken into full account.

This kind of representation is a very inherent aspect of relational theory. In essence, what is happening is that constraints are being placed on the items to be contained in the record types in a data base structure, with the aim of being able to manipulate this structure in such a way that the application programs are somewhat more independent of changes to the data structure than they otherwise would be. Again the relationships between record types are implied in the data items in those record types.

While endorsing wholeheartedly the practice of propagating prime keys down one level, it seems very contrived to do this down two levels or more. The requirement to include the item EMPNO in the SALARY-HISTORY record type merely to satisfy the semantic needs of a specific manipulation language is hard to support.

## 8. Data Sub-languages (DML) for a Retentional Data Base

There is no single DML defined for a relational data base. Instead there are two approaches which are based on relational algebra and relational calculus. Date (7) summarizes the difference between these rather succinctly.

"There are at least two ways in which we might allow the user to do this: (1) he could actually specify the sequence of relational algebra operations to be performed to produce the desired result; (2) he could simply state a definition of the desired result in terms of the relational calculus leaving the system to determine which operations are necessary. The difference between the two approaches is analogous to the difference between (1) actually constructing a set by performing a sequence of set operations (union intersection, etc.) and (2) simply stating the "defining property" of a set in the form of a predicate; in other words difference between procedurality and non-procedurality".

## 8.1 Relational Algebra

A relational algebra operation refers to one or more record types and names yet another record type as the result of performing the operation. The two main operations are referred to as projection and join.

A projection acts on one record type only and produces one with less items. To be precise, the operation does not act on a record type, but on all the records of the type named. The effect of executing a projection is to generate a collection of records, all of this new record type. These are regarded as being stored in what relational theory calls a workspace.

A workspace differs from a CODASYL user working area in two major respects, the first being that it is essentially open ended and the second that there is no restriction which states that it accommodates only one record of each type.

In COBOL, records which are generated during the course of execution of a program would normally correspond to a record type defined in the Working Storage Section. In addition, the application programmer would have to delcare the space needed for storing the file produced as a result of executing the projection.

The other major statement is usually referred to as a "join". If two record types have an item which is in both and the values of the item are in both cases selected from the same domain of values, then it is possible to perform one of these join operations on the common item. The effect is to generate a collection of records of a new type which contains all items from both record types, but the common item only once. If a value in the common item appears in the collection for a record of one type and not for any record of the other type, then the record containing this value does not participate in the join.

In fact, the join is nothing very novel. It will be recognized by those with experience in the processing of tape files as a merge operation. Nevertheless, the projection and join together seem to provide a powerful relatively non-procedural way of minipulating files.

One could argue that it is necessarily heavy on storage space compared with the record at a time logic of the commercial systems in use today. Its advocates point to the much reduced number of statements required to specify any given transaction.

## 8.2 Relational Calculus

The relational calculus is different from the relational algebra in the way already indicated. The data sub-language ALPHA (2) is the best known example of this approach. Just as the FIND statement is the cornerstone of the CODASYL DML, so is a powerful GET statement the most important statement in ALPHA.

The general format of a GET statement might be as follows:

GET workspace target-list WHERE condition

The parameter workspace refers to the storage space in which the set of records generated as a result of the statement are stored. It is an important aspect of ALPHA that the programmer does not need to know whether the workspace is in direct access storage or not. The target list consist normally of a list of item names (possibly qualified by record names), and record names alone. Finally, the conditon could be of full Boolean complexity rather like compound conditionals in COBOL.

In addition to the GET, there are a number of other statement types such as PUT (i.e., STORE), UPDATE (MODIFY) and DELETE.

# 9. Summary of Conclusions

A number of conclusions can be made from the analysis of the relational approach. They are as follows:

1. The relational approach requires a data definition process which contains a number of restrictions.

2. Two of the restrictions - unique prime key and the way of defining set types - are felt to be quite acceptable. Propagating prime keys from the top to the bottom of a deep structure is questionable.

3. The relational approach permits the definition of network structures as complex as those proposed by CODASYL (with the exception of cyclic structures).

4. The CODASYL approach fits far more closely with relational theory than does either IMS or TOTAL.

5. The requirement to propagate prime keys through the structure is needed to support the semantics of the data sub-language suggested for a relational data base.

6. The CODASYL single record at a time DML could manipulate a relational data base.

Finally, it must be stated that much of the debate "relational versus network" seems to have been contrived to draw peoples attention away from the deficiencies in other systems such as IMS. In fact, the relational approach and the CODASYL approach have quite a lot in common. The DDLC decisions during the period 1972 to 1976 indicate that relational theory had indeed been having an impact on the CODASYL thinking, and rightly so. Relational theory on the other hand as yet had no impact whatever on IMS.

References


1. E.F. Codd. A relational model of data for large shared data banks, CACM, Vol. 13, No. 6, June 1970, pp. 377-387.

2. E.F. Codd. A data base sub-language founded on the relational calculus, Proceedings of ACM SIGFIDET Workshop on Data Description Access and Control, pp. 35-68.

3. E.F. Codd. Understanding relations, FDT, Vol. 6, No. 4 (1974), pp. 18-22.

4. ACM SIGMOD Workshop on Data Description, Access and Control, Vol. 2 "Data Models: Data-Structure-Set versus Relational", Ed. R. Rustin.

5. E.F. Codd. Normalized Data Base Structure: A brief tutorial. Proceedings of ACM SIGFIDET Workshop on Data Description Access and Control, pp. 1-16.

6. C.J. Date. Relational Data Base Systems: A Tutorial. Published in Information Systems COINS IV, 1974, Plenum Press, pp. 37-54.

7. C.J. Date. An Itroduction to Data Base Systems. Published 1975 by Addison-Wesley.