Database—the ideas behind the ideas

K. A. Robinson

Data Base Controller, Management Services, Surrey County Council, County Hall, Penrhyn Road, Kingston-upon-Thames, Surrey

A data base is a model of the real world. The programs which update and retrieve the data are themselves models of events in the real world. Maintenance of programs and systems is much easier if that part of the world being modelled by the programs and data is intuitively obvious from the data descriptions and the application procedure descriptions.

Three approaches to database management (CODASYL DBTG, GUIDE-SHARE and Relational Databases) are considered with regard to the facilities they provide for modelling the real world and the mechanisms by which they physically support these facilities.

(Received July 1973)

1. Introduction

A data base is a store of information which provides a variety of users with the facility to enter or retrieve data relating to their environment. The content and context of the information stored essentially provides each user with a model of that part of the world with which he deals. Inasmuch as a variety of users interact with the data, they are all interacting with the same real world entities which that data models.

It is no less true that a file in a conventional system is a model of part of the world. Systems analysis training in the past has been output oriented; the attention paid to stored data has been biased towards efficient processing of that data and has almost completely neglected the investigation and definition of what the data is about. Unless the real world structure is obvious from the data definition maintenance of the system is very difficult. This is even more true of applications operating in a database environment.

This paper considers three apparently different approaches to database management: the Joint GUIDE-SHARE Database management system requirements report, the relational database concepts exemplified by the work of E. F. Codd of IBM, and the 1971 CODASYL Data Base Task Group report. These approaches are compared thus:

(*a*) The way in which data being described models the world
(*b*) The view of data given to the applications
(*c*) The accessing of the data
(*d*) The physical storage of data

I believe that it is essential to the success of a database project that these four functional areas be clearly separated at the specification stages. I am thus particularly concerned with the merits and demerits of the three approaches in these areas.

2. Data models the world

2.1. *Entity data (GUIDE-SHARE)*

A central concept in the GUIDE-SHARE report (1970) is that of 'entity' data. An entity is defined as 'A person, place, thing or event of interest to the enterprise', how abstract or concrete a 'thing' may be is left undefined. The concepts built round entities are fairly daunting to the first-time reader and seem needlessly complex. There is something called an 'entity construct' (which appears to be an association of entities, e.g. PARENTAGE = MAN, WOMAN, CHILD). Research in the glossary of the report reveals the 'entity construct' as just another (if somewhat more abstract) entity in that, if named, it defines an 'entity record type' (an 'entity record type' represents the attributes of a particular 'entity type' (defined as 'entity' type)).

Since an entity (more or less by definition) is something which has existence whether its associated 'entity record type' is

EMPLOYEE = PAY-NUMBER
START-DATE
DEPT-NUMBER
MGR-NUMBER
COURSE-NUMBER } Occurs n
COURSE-TITLE }

Fig. 1(a) Unnormalised relation (record)

EMPLOYEE = PAY-NUMBER
START-DATE
DEPT-NUMBER
MGR-NUMBER

TRAINING = PAY-NUMBER
COURSE-NUMBER
COURSE-TITLE

Fig. 1(b) First normalisation step—elimination of repeating groups (Notice TRAINING inherits key PAY-NUMBER as well as establishing another attribute (COURSE-NUMBER) as part of its primary key)

EMPLOYEE = PAY-NUMBER
START-DATE
DEPT-NUMBER
MGR-NUMBER

TRAINING = PAY-NUMBER
COURSE-NUMBER

COURSE = COURSE-NUMBER
COURSE-TITLE

Fig. 1(c) Second normalisation step—separation of attributes not fully dependent on key

EMPLOYEE = PAY-NUMBER
START-DATE
DEPT-NUMBER

DEPARTMENT = DEPT-NUMBER
MGR-NUMBER

TRAINING = PAY-NUMBER
COURSE-NUMBER

COURSE = COURSE-NUMBER
COURSE-TITLE

Fig. 1(d) Third normalisation step—separation of attributes dependent on other attributes

named or not, it would seem that an 'entity construct' defines an entity pure and simple.

2.2. *Third Normal Form (Codd)*

At the heart of the relational database approach is a mathematical theory of data. Data records represent relations between the attributes of entities. Every record type must have a primary key to uniquely identify each occurrence of the record type to users of the data; the occurrences must not be identified by external sequencing, e.g. instead of saying 'get me the third occurrence of record type REC' we must say 'get me the record type REC with primary key OCC—NO = 3'. Where two or more data items or groups of data items may serve as a primary key one such is arbitrarily selected.

Codd (1970; 1971a; 1971b) looks at the numerous ways in which data may be structured and suggests a way of reducing these to a 'normal' form. This is done in three stages:

(*a*) Separation of repeating groups into separate records (inheriting the primary key data from the original record).
(*b*) Separation of attributes not dependent on all fields of the primary key into separate records with primary key data being that part of the key on which the separated attribute(s) depend.
(*c*) Separation of attributes dependent on other attributes into records with primary key being those other attributes.

An example of the three normalisation steps is given in **Fig. 1.** Primary keys are underlined. (My naming conventions differ slightly from Codd's).

The principal objective of performing the three normalisation steps is the removal of update anomalies, e.g. in Fig. 1(*b*) the deletion of the last TRAINING record for a particular COURSE would mean that information was lost about the COURSE itself (COURSE-NO and COURSE-TITLE).

If the primary key is chosen differently in Fig. 1(*d*) this leads to a different set of records in third normal form as shown in **Fig. 2.** The implications of the choice of primary key to a great extent reflect the real world situation obtaining. Fig. 1(*d*) reflects the usual situation where when employees move departments they automatically change their departmental managers: Fig. 2 reflects a situation more akin to the feudal system where when a manager changes departments all employees move with him.

EMPLOYEE = PAY-NUMBER
START-DATE
MGR-NUMBER

DEPARTMENT = MGR-NUMBER
DEPT-NUMBER

Fig. 2 Alternative third normal form to 1(d)

The three normalisation steps are, in essence, an algorithm for discovering the underlying entity structure of the data.

The casting of records (or relations) in third normal form with a sensible choice of primary keys clearly provides an excellent basis for describing the underlying real world entities.

2.3. *Record descriptions in the schema (DBTG)*

The schema in the DBTG report (1971) holds the central descriptions of all records. No guidance is given in the report as to what collection of data should be put inside records but extrapolation from inferences in the report is generally confirmed in discussions with DBTG members giving advice such as 'Define fine entities'.

The records are fairly obviously entity descriptions. It is possible to bury one entity type in a repeating group inside a record representing another entity type but it is not intended that this should be done; the two entities should be defined as separate record types linked in a 'set'. Also, the absence of a facility for defining many-many record linkages forces the recognition of entities such as TRAINING in Fig. 1(*c*) linking EMPLOYEE and COURSE.

It is possible to include certain data items in a record type which originate in a different record type linked in a 'set' with the first one (these items are the ones defined with an ACTUAL or VIRTUAL SOURCE clause). There is an implicit recognition that some of these items do not really belong to the entity defined by the first record type; 'control data' items referenced in SET OCCURRENCE SELECTION clauses may be MODIFYed, the others may not. In fact, these 'control data' items are exactly parallel to those constituent data items of the primary keys common to pairs of records in Fig. 1(*d*); the other data items are those attibutes which could be removed by applying the various normalisation rules described in Section 2.2 and are there principally to afford time/space trade-offs.

A defect in the proposals (if one takes the view that all attributes of an entity should be made explicit) is that the inclusion in records of ACTUAL or VIRTUAL SOURCE data items derived from control data items referenced in SET OCCURRENCE SELECTION clauses is optional rather than mandatory. These items do, as shown in Section 2.2, really belong to the records since they are the key data used to identify occurrences of those records and, as such, should be explicitly defined. In fact, these derived items are not treated consistently throughout the report, e.g. no indication is given as to what happens when a record with a given value of a SOURCE data item is STORED.

2.4. *Comparison of entity concepts*

We can see that all three approaches do share the 'entity' concept. This is explicitly (if not clearly) defined by GUIDE-SHARE; Codd's normalisation steps are an attempt to discover the entities to which the data relates; DBTG has an implicit idea of the entity and (if the inconsistencies, e.g. in the STORE command, are cleared up) can be used to define third normal form data although, unfortunately, it provides facilities for defining unnormalised records.

3. Data Seen by Application Programs

3.1. *Logical Record (GUIDE-SHARE)*

In the GUIDE-SHARE requirements, entities are never directly referenced by application programs. The application programs are interfaced to 'logical data'. Two properties of the logical data are that it need not contain all the attributes known for the entity data which the applications interface to, but only those attributes that are actually used, and also that unnormalised records of the type shown in Fig. 1(*a*) are constructed and deconstructed by the DBMS (or DBCS) from the entity data, for use by the application programs.

This accessing of unnormalised records by the application is deemed to be necessary to realise the GUIDE-SHARE objective of data independence. My inclination is to reject this view because I think that the proper way to achieve data independence is by making the applications aware of the entities which they are processing; an application program which 'knows' how the data models the world is much more easy to maintain than one whose functions depend on implicit descriptions of the world (e.g. a program which accesses unpaid invoices by selecting those with a null payment date is easier to understand than one which gets invoices from a file which is implicitly known to contain only unpaid ones). Tremendously cumbersome routines are needed to normalise and unnormalise the data and resolve the anomalies that occur when applications attempt to update unnormalised data. Such routines must be more expensive than the occasional rewriting of an

application program accessing entities whose initial bad definition forces change. The Codd normalisation rules provide an excellent basis for keeping bad definition of entities to a bare minimum.

The accessing of a subset of the entity data is useful and certainly reduces the amount of maintenance when new attributes are added to entities.

3.2. *The Target List* (*Codd*)

The data manipulation language, ALPHA, advocated by the relational database enthusiasts is primarily directed towards retrieving and processing entity data. The data items accessed by the application programs are subsets of the entities and may be combined into first normal form records; the subset of data items which is to be retrieved is called the 'target list'. Some of the items which may be retrieved are not data items from any of the entities but are the results of procedures (such as counting the number of occurrences of given values) applied to such items.

The updating of items is effectively confined to updating one entity type at a time thus avoiding update anomalies.

The definition of the target list is dynamic. It is defined by the access statement rather than the program in which that statement occurs.

3.3. *Record descriptions in the subschema* (*DBTG*)

The application programs do not interact directly with records defined in the schema but rather with records defined in a subset of the schema called a 'subschema'. A subschema is defined statically, before a program is defined, and gives the subset of the database available to a program or series of programs.

Insofar as the data descriptions in the schema define real world entities, the subschema record descriptions form a subset of the attributes of those entities. The only way in which attributes from other entities can be associated with the attributes of a given entity to form a logical record is by defining such derived attributes as data items with an ACTUAL or VIRTUAL SOURCE clause in the schema definition of the record. Items which are procedure results may also be specified in the schema and accessed in the subschema.

As in the Codd approach, derived attributes (except for those that are really implied key items) can only be modified via the entity to which they belong.

3.4. *Comparison of logical record concepts*

The GUIDE-SHARE approach whereby the DBMS builds up and fragments unnormalised records is unnecessary if the real world entities are properly defined in the first place. The approach of ALPHA and the DBTG approach are equivalent even though one system requires the logical record to be defined at access time and the other requires the logical record to be defined prior to compile time; neither approach allows the construction of completely unnormalised records and neither approach allows the update of items that do not belong to the entity being accessed.

I believe that DBTG mix up the functions of accessing entities and of storing the data in an optimal manner and that even though, for efficiency, derived attributes are stored in records defining other entities, the system should still appear to the application to be accessing data purely on the basis of the entities to which it applies. In the example of Fig. 1(*c*) (*d*) I would argue that the application program should issue calls to GET EMPLOYEE and GET DEPARTMENT if it requires to access MGR-NUMBER for an EMPLOYEE even though MGR-NUMBER may be stored in the physical record representing the entity EMPLOYEE; how this may be achieved is discussed in Section 5.4.

4. Access methods used by the programmer

4.1. *Database command language* (*GUIDE-SHARE*)

The GUIDE-SHARE requirements allow data to be organised into files. A file is 'a named collection of occurrences of logical records which may be of more than one logical record type'. Such a collection may have a sequence and structure defined by the data administrator. It is a logical file rather than the kind of file known to present day operating systems.

It is not entirely clear how the DBCS is to tell that a logical record is a member of a file. GUIDE-SHARE provide for the explicit addition of records to a logical file in an analogous manner to the DBTG INSERT statement (paragraph 4.3 below) as well as for the implicit membership of records in files based on data content; this implies that the system must keep some list of the records which are members of particular logical files. Thus, in a sense, the logical file must be physically implemented.

The DCBL is a language which acts on logical files. The application programmer can make requests for logical records from these files based on conditional expressions. Primitive functions for operating on this data are defined, e.g. retrieve, add, delete, replace, and the facility to combine the primitive functions into more powerful compound functions is required.

4.2. *ALPHA and relational algebra* (*Codd*)

Codd conceives of two different languages for operating on the data. One of these, ALPHA, is a language based on the predicate calculus and enables the statement of logical selection requirements in a non-procedural manner. The other possibility is relational algebra which allows the application of various primitive logical operations to build up procedures to handle the selection requirements; such an operation might be the joining together of the TRAINING and COURSE records of Fig. 1(*d*) on the item COURSE-NUMBER in order to find who had attended a particular course.

Codd strongly advocates the high level predicate calculus based language typified by ALPHA as a data handling language and, unfortunately, the whole relational approach now seems to be identified with the use of a predicate calculus based language. This identification is incorrect; the casting of data in third normal form, is independent of the choice of language for accessing and processing. The undoubted difficulty of implementing a language such as ALPHA is totally irrelevant to the use of third normal form data.

The nearest that ALPHA comes to supporting the concept of a file is the 'workspace'. The workspace is a storage area which holds the occurrences of the logical records defined by the target list which have been selected and ordered according to criteria specified in the access statement. A workspace is local to a program, being defined at access time; it could, e.g. contain all EMPLOYEE records arranged in ascending PAY-NUMBER order, or it could contain just that EMPLOYEE record with PAY-NUMBER = 12345.

4.3. *Sets and the data manipulation language* (DBTG)

The DBTG report makes extensive use of a concept called the 'set' which is defined as 'a named collection of record types'. It consists of an owner (which is either the system itself or a named record type) and a number of occurrences of one or more named member record types. There may be multiple occurrences of each set type—one such for each occurrence of the owner record type.

A set seems to be a single level logical file, or a collection of such files, one for each owner record occurrence. This is not strictly true if our definition of a logical file is something like '*any* named collection of record types' because the set concept requires that some method of physical implementation (such as chaining) be specified; a set is, in fact, a logical file for which some kind of physical support (extending to identifying file

members and their sequence) is provided.

The DML statements enabling access to the data do so largely in terms of these sets, e.g. access to records of a given type in a given sequence must be accomplished via a set which has been ordered either as records are STOREd (using a schema defined set order) or locally using the ORDER verb; access from the EMPLOYEE record of example 1d to the TRAINING records would be via a set with the first record type as owner.

The DML supports the facility to manually INSERT records into sets as required rather than automatically making them members at STORE time.

4.4. *Comparison of access concepts*

Of the three approaches GUIDE-SHARE and DBTG are very similar. Both purport to access some kind of logical file which actually turns out to be physically supported by some kind of list structure. The GUIDE-SHARE file is admittedly a more general hierarchical structure than the simple owner-member set of DBTG; the GUIDE-SHARE report also requires more complex access qualification than DBTG provides, but this merely changes the amount of work the programmer must do—it does not change the quality of interface to this quasi-logical file or set.

This may be unfair to GUIDE-SHARE, but the kind of system they seem to be thinking of is one where the system maintains its own list of the members of the files—how else can a system which allows manual membership of files typified by the DBTG INSERT verb operate? My view, already expressed in paragraph 3.1, is that all information should be explicit in the entity descriptions and not implicit by the membership of an entity in some file. If all information is explicit, then what is a file?

What does a conditional expression applied to select information from a file mean? My own view is that a conditional expression applied to select data from a file is merely a definition of a smaller file (or set of files) which should have been defined as such in the first place.

The ALPHA workspace idea seems to be the best hope for true data independence. Using this concept, a file is any collection of records whatsoever, identified by their content—or the content of some (possibly remotely) related records—which a program needs to access and which are ordered in the sequence in which they need to be accessed. ALPHA, however, in my view, is not implementable within our current level of technology.

What can be done? Suppose we define logical files of the DBTG set variety and make them purely logical concepts, i.e. the system may have no other way of deciding which records belong to a set other than scanning the entire database, selecting the required records and then sorting them into the defined set order. The system can maintain lists for the most popular sets and use them to access these quickly. The data administrator can write procedures to determine whether records are members of the defined sets. Some of these procedures will be system supplied like the key matching procedure provided on the SET OCCURRENCE SELECTION clause at present. These procedures can operate in one of two modes: at STORE and MODIFY time leading to INSERTion into a list occurrence (and similarly for DELETE) or at FIND time requiring a scan of the database. Where a list is maintained then at FIND time part of the INSERTion procedure would (as at present) have to be invoked to determine the list to be used. The procedures would also have to resolve differences between the list ordering and the set ordering.

For the installation with a DBTG based system which wishes to start building its database now, the solution described above may seem to provide no comfort. However the data administrator of the installation could define a logical file interface which the application program modules could use and build his database access modules on the other side of that interface using the set purely as a physical list. This is the approach many users are adopting.

5. Physical Mapping of Data

5.1. *Physical Data* (*GUIDE-SHARE*)

There is no implication in the GUIDE-SHARE report that any particular form of physical mapping must be used. Items may not actually be stored in the database. It is possible to invert the records when storing them. Data retrieved is always 'materialised' from the physical data.

5.2. (*Codd*)

Beyond passing references to some attributes being indexed no physical storage considerations are described.

5.3. *Sets and records in the schema* (*DBTG*)

Record descriptions in the schema are entity descriptions but they do also describe the physical records. Data items belonging to the entity can be specified as virtual items derived either by calculation or from a record describing a related entity. The physical record also may contain items which belong to other entities; this facility is provided to allow multiple copies of the same data item to be held for access optimisation.

5.4. *Comparison of physical record concepts*

GUIDE-SHARE clearly separate their entity, logical and physical record concepts. Codd does not deal with physical data at all. DBTG have, to some degree, mixed up the functions of defining entities, logical and physical data.

If we follow the principles outlined in Paragraph 3.4 of only allowing logical records to be subsets of the data items which are attributes of the entities, then we must not in a DBTG system allow records to have SOURCE derived data items which are not used as control data items in SET OCCURRENCE SELECTION clauses for sets containing those records. If we do follow these principles it may be desirable to provide some mechanism to allow the system to make use of redundant copies of data items for access time optimisation.

Fig. 3(*a*) shows how the access requirements described in paragraph 3.4 are currently catered for if redundant copies of the item MGR-NUMBER are held. Fig. 3(*b*) shows an alternative method which allows logical records to contain only items which are attributes of the entities described by those records.

```
RECORD NAME IS EMPLOYEE
  02 PAY-NUMBER
  02 START-DATE
  02 DEPT-NUMBER; VIRTUAL, SOURCE IS DEPT-NUMBER
                  OF OWNER OF DEPT-EMPS SET
  02 MGR-NUMBER; ACTUAL, SOURCE IS MGR-NUMBER
                 OF OWNER OF DEPT-EMPS SET
```

Fig. 3(a)

```
RECORD NAME IS EMPLOYEE
  02 PAY-NUMBER
  02 START-DATE
  02 DEPT-NUMBER; VIRTUAL, SOURCE IS DEPT-NUMBER
                  OF OWNER OF DEPT-EMPS SET

RECORD NAME IS DEPARTMENT
  02 DEPT-NUMBER
  02 MGR-NUMBER; COPIED INTO MEMBERS OF
                 DEPT-EMPS SET
```

Fig. 3(b)

Using the data structure of Fig. 3(*b*) a request to FIND OWNER OF DEPT-EMPS SET followed by a GET MGR-NUMBER would be required to retrieve MGR-NUMBER for an EMPLOYEE's DEPARTMENT but would not result in any access other than that required to retrieve the EMPLOYEE record since the system would know that the MGR-NUMBER was held there.

6. Conclusions

All three approaches contain some basic idea of data describing entities in the real world; the work of E. F. Codd, in particular, is oriented towards discovering the entity structure of the data.

With regard to the way the applications see the data all three approaches let the programs see only the information actually used by them. GUIDE-SHARE require the facility to aggregate the subsetted data into elaborate structures; Codd and DBTG restrict aggregation to the formation of logical records without repeating groups in them and restrict updating to items which belong to the entity. My view is that the entity structure should be properly identified and that the applications should explicitly access all the entities with which they interact, logical records being restricted to subsets of the entity data. The DBTG system can be operated in this way and the elaborate GUIDE-SHARE logical data structuring facilities are not needed.

For access, both GUIDE-SHARE and DBTG make use of some kind of logical file concept which in the former appears to be, and in the latter has to be, physically supported using a list structure. The Codd language, ALPHA, does not process files as such but rather allows access commands to define the data to be processed and its sequence of processing, both based on values in the data. Although the ALPHA language does present severe implementation problems its concepts can be used. I believe that if all information about an entity is present in the record describing that entity and not denoted by its membership in some 'file', then a file does not really exist except as a description of a collection of records known to a particular program. With this concept it is possible to keep separate the application program modules accessing the data in the file, and the database access modules which have to access the data from the database and create the file.

The Codd system makes no real mention of physical data. DBTG and GUIDE-SHARE both provide for virtual storage of data and for holding redundant copies of data where desirable. DBTG does mix up its entity description functions and its mechanism for holding multiple copies of the same data item and if the DBTG system is to be operated on the basis of explicitly accessing the entities to which the data relates this mixup should be resolved.

The DBTG system is the only one of the three for which implementations are commercially available; in fact, the GUIDE-SHARE report describes not a system, but a set of requirements. Many of the most useful features for building a system based on entity data held in third normal form are not available in current implementations. Nevertheless, the DBTG system can be used to build a database which is a good model of the world although a lot of the work which would ideally be performed by the system is devolved on to the data administration staff.

References

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

CODD, E. F. Further Normalization of the Data Base Relational Model, Courant Computer Science Symposia **6** *Data Base Systems*, published by Prentice Hall, May 1971.

CODD, E. F. A Data Base Sublanguage founded on the Relational Calculus, *IBM Research Report RJ893*, San Jose, California, July, 1971.

CODASYL, *Data Base Task Group Report*, BCS HQ April 1971.

Joint Guide-Share, *Data Base Management System Requirements*, Guide or Share Distribution, November 1970.