Consistency auditing of databases

J. J. Florentin

Computer Science Department, Birkbeck College, Malet Street, London WC1E 7HX

Many databases need to have their contents kept at a high level of accuracy. New kinds of auditing procedures must be devised to ensure this. Inaccuracies arise from various causes; in this paper one particular fundamental aspect of accuracy is isolated—that of consistency of the stored data. Inconsistencies occur when the stored data fails to reflect the rules of compatibility of the elements of the applications world. The expression of the rules governing the generation of the data in the application, and the resulting complexity of validation procedures are investigated through mathematical logic.

(Received October 1972)

1. Introduction

Many databases are extensions of the traditional files and records of business and government organisations. An important activity in these traditional information systems is that of auditing; the main reasons for auditing can be summarised briefly as:

(*a*) management has a legal obligation towards shareholders and government taxation agencies to see that accurate financial accounts are kept

(*b*) records, such as stock, personnel, and customers, are used as a basis for management decisions, and it is prudent to see that these records are kept properly

(*c*) errors and malpractices in administration are brought to light

Computerised databases need to be audited for the same reasons, but further issues also arise; two of these are:

(*d*) in an automated system the storage of erroneous data may initiate a chain of actions which can reach outside the organisation, and cause various kinds of losses. For example, an error in a customer billing system may eventually cause unfavourable publicity to a company

(*e*) legislatures in several countries are considering the regulation of databases dealing with private citizens. Should these considerations actually lead to legislation it seems likely that some new kinds of auditing will be instituted.

At the present time accountants have adapted manual auditing procedures to computerised accounts and stock records; however, in many cases the checks are oriented towards the faults which occurred in manual systems, for example errors in addition. Now, errors in addition are rare in machines. On the other hand, the complexity and flexibility of computerised information processing is such that entirely new errors arise, and work from several directions will be needed to devise procedures to ensure the accuracy of large volumes of stored data. This paper isolates one fundamental aspect of the problem by pointing out that the preservation of accuracy ultimately depends on a statement of logical constraints. All present data storage systems incorporate a variety of data validity checks, and this paper shows how some of these can be subject to detailed study in the light of the above logical statements. Even in scientific computing systems, users of shared databases will be forced to rely on a well-designed data management system to preserve the accuracy of their data; for, in any shared database one particular user will not have the comprehensive knowledge required to make a chain of several updates which might be needed following one specific update. Generally, any large data store will accumulate more and more errors as time passes, so that the reliability of retrieved information is continually degraded; auditing can be regarded as a means of keeping the reliability of the information at some required level.

There are many ways in which inaccuracies can occur in stored data. A number of authors (Fraser, 1969; Wilkes, 1972) have discussed database integrity problems in relation to hardware and software malfunctions; this paper discusses the inaccuracies which occur when the hardware and software functions correctly, but chains of insertions and deletions by users eventually cause inconsistencies to appear. These have been mentioned by Codd (1970). These inconsistencies are logical, semantic incompatibilities; as an example consider a company in which it is an organisational rule that every department has one manager but, on inspecting the personnel file it is found that two employees have the designation 'manager of data processing'. This error is a discrepancy between the known rules of the outside world, and the description of that world held in the database. Such an incompatibility could have occurred because of an erroneous input, or because of a time lag in deleting an entry. Time lags in entering changes are always likely in large databases, and there is no simple validation procedure which will catch them. More complex validation procedures are needed, and these must be founded on the known rules of the outside world which gives rise to the data.

In a large database it would not be practical to make periodic checks of the whole of the stored data; instead, every update must be checked to see whether it could cause an inconsistency to appear, and also to see whether the data management system should take some consequential action. Such action could be automatic data modification or signalling of an error. To make a system of entry validation effective it is essential for the user to have pre-formatted transactions. Of course, these pre-formatted transactions may be extracted by a translator from an apparently free format input. Also the number of formats could be large.

To design the validation procedures it is necessary for the systems analyst to determine the conditions under which updates to the database occur, and then to show how these give rise to consistency conditions on stored data. During the design of a database decisions have to be made as to which aspects of data accuracy are the responsibility of the user, and which are the responsibility of the database administrator. By considering the underlying logic of the validation procedures the issue can be discussed in a systematic way. An important objective of validation theory is to estimate the amount of effort needed to effect validation; this paper presents a method for making such estimates. The discussion in the paper is based on highly simplified examples, but for the discussion of

validation effort it is necessary to introduce mathematical logic.

2. Consistency

The manifestation of accuracy considered here is the consistency of the stored data. Data solely within a machine cannot be consistent, or inconsistent; it can only take on this quality when considered as a description of some world outside. If the stored data describes a permitted state of the outside world then it is consistent, if some feature of the application is misrepresented then the data is inconsistent. Thus, to test consistency it is first necessary to have a well-documented account of the applications world. Systems analysts, of course, already collect such information during their normal work. Naively, all that would be necessary to design a database with consistency checking would be to extract from the systems documentation all the logical constraints on the data. Taking these logical constraints the permitted updates can be determined, together with the circumstances in which they occur, and then validation procedures can be devised. Needless to say, such a completely general and undirected approach is impracticable. To make progress it is essential to set up a limited framework within which the problems can be analysed. Here, a specific framework is established for describing the applications world. This makes it possible to describe the consistency problem in an effective manner, more particularly it enables the sources of changes in the data to be described easily. It should be noted that these issues appear when using any database system; for instance, in most systems a data item can be accessed by several paths, and it is necessary to give rules to determine when an item can be completely deleted. Such rules must be found from the systems analysis.

There are two aspects of consistency: when at one time instant the database must reflect a possible state of the applications world, it is *static* consistency. At successive time instants the world may change in certain, limited, ways and the database will be correspondingly updated. Any database state must have been arrived at by a sequence of permitted changes from the initial state; this is *dynamic* consistency.

2.1. *The world description method*

The world description method is based on the notion of an entity, like an employee, a product, etc. which has properties and can enter into relationships with other entities. An entity is of a single, specified, type, and entities of the same type are collected into sets. The description is to be considered as a symbolism for reasons explained later. The following symbols occur.

Entity set symbols There is an entity set symbol for each type of entity. Sets of entities may have a finite or denumerably infinite number of elements. It is convenient to allow an infinite number of elements when the systems analyst does not want to place an upper limit on the numbers involved. Of course the number of items actually stored in the database is always finite.

Attribute set symbols Attributes are properties of entities, and also of relations over entities. Attribute sets are also finite or denumerably infinite, and may contain a special symbol meaning 'temporarily undefined'.

Property function symbols An entity set or relation is associated with its attributes by a property function. A property function may be permanently undefined for certain arguments values; for example, in a personnel file, if factory workers have a works number but office workers do not, the property function 'works number' taking employees as arguments is undefined for office workers. This situation is different to the 'temporarily undefined' attribute above.

Relation symbols Relations between entities only are considered here. Some extension is possible, but will not be carried out here.

It would be possible to elaborate this model, and introduce further symbols, for instance, for the world state. However, such symbols are not used in the present examples. More advanced work will need a more elaborate model, especially where time synchronisation effects are more fully taken into account.

2.2. *Definition of consistency*

It is now clear that consistency means that a detailed correspondence exists between the world state and the contents of the database. But in the theoretical treatment a direct correspondence between the world and the stored data is not made; instead, the relevant properties of the world are first abstracted into logical axioms, and then both the world and the stored data are taken to be models of the same axioms. Two interpretations, in the sense of mathematical logic, are made; this is illustrated in **Fig. 1.** This approach allows the database and world to differ, and yet retain the same logical rules.

A detailed correspondence must now be set up between the symbols in the axioms and the data items, and also between the symbols and the entities in the world. To describe the database it is convenient to imagine a specific implementation, but it is stressed that this is hypothetical, and does not describe any real implementation. In the database each entity set and each relation is kept as a separate file. The records in these files contain a key, and attribute fields for that entity, or relation, entry. A key for an entity entry contains a portion identifying the type of the entity, and a unique identifying portion for that particular entity. A key for a relation entry contains a portion identifying the particular relation, and portions describing the particular entities involved in the relation, as shown in the example of **Fig. 2.** Although this structure is hypothetical every real implementation must still preserve the key structure, although keys may be realised in many ways. A key could be an

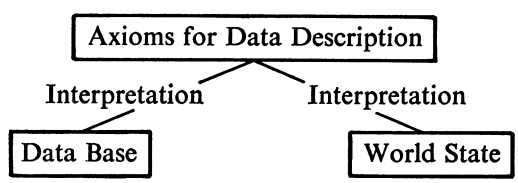

Fig. 1 Links between the data axioms, world and database

Student entity set

key	status
s1	part-time
s2	full-time
s3	—

Course entity set

key	faculty
c1	arts
c2	science
c3	science

Students_taking_courses relation

key	time
stc_s1_c1	evening
stc_s1_c2	day
stc_s2_c3	evening
stc_s3_c3	—

Fig. 2 An example of a database with keys

Table 1 Interpretation of the symbols of the data axioms into the world, and into the database

AXIOMS	WORLD	DATABASE
Individual entity symbol	Entity of appropriate type	Keyed data item with key showing type
Entity set symbol	Entity sets	A convention exists for constructing unique keys with this type designation
Individual attributes and attribute set symbols	Attributes of appropriate type	Attribute data can be stored, updated and read only by basic operations supplied by the database management system
Relations and elements of relations	Associations between entities	Relation entries with keys formed from a portion identifying the relation, and portions identifying the constituent entities
Property function symbols	Entities and relations have properties	Basic operations with entity arguments exist to update and read properties of entities. Operations supplied in fixed forms by the data management system

Table 2 Updating rules for the example database when there are no data constraints

WORLD CHANGE	DATABASE UPDATE
An attribute value is changed, e.g. 'part-time' becomes 'half-time'	Can be done freely, attribute value changed everywhere it occurs.
A new entity of existing type is introduced	A new unique key of correct type is constructed, and a new entity entry created.
Entity disappears	Entity entry erased. Attributes no longer accessible through that entity. Any relation entry containing that entity is also erased. The relation entries to be deleted can be found via their key constituent.
A new association occurs between entities	A new relation entry is made, but only if the constituent entities are already in the database.
An association between entities is broken	Delete the relation entry.
An entity gains or loses a property	The attribute entry is altered accordingly. The entity must already be in the database

alphanumeric string, or an access path, or could be contained in a record as an apparent content.

Static consistency is now defined by showing the correspondence between the axioms, the world, and the database. This is shown in **Table 1.** Dynamic consistency is checked by invoking logical constraints, as described later.

2.3. *An example*

A highly simplified example of a database is shown in Fig. 2. The entities are students and courses, and the relation 'students_taking_courses' and various attributes are shown. The status of student s3 is 'temporarily undefined'. In this example there are, so far, no data constraints, so the axioms merely assert the existence of the entities, attributes and relations as listed.

2.4. *Updating without data constraints*

Table 2 shows what updates occur in the example database as a result of changes in the world. The consequential deletion of relation entries following an entity deletion and the need to validate the entity entries before accepting a relation entry are pointed out.

2.5. *Updating with a constraint*

Constraints may be of many different kinds. As an illustration consider the world rule:

'No part-time student can take more than one science course'

Entries to the 'students_taking_courses' relation now have to be validated. First the student entity file has to be checked for the 'part-time' attribute, then, if necessary, the course file has to be checked for 'science'. Then, again if necessary, the 'students_taking_courses' file has to be checked to see if that student has already enrolled for a science course. Several variations of this procedure are possible.

The entry for student s3, who has no recorded status, presents further problems. His enrolments for science courses have to be accepted until his status is known; at this time some of his entries may become invalid. Some implementation technique could be devised to mark these potentially invalid entries, but the important point is to recognise that the situation exists, and allow for it at the database design stage.

This simple constraint shows two typical features of the resulting validation:

1. Validity can depend on the current state of the database, and the database itself must be referenced.
2. It can be necessary to have a sequence of searches, where one search must be completed before another is begun.

2.6. *An alternative treatment of the constraint*

Another approach to handling the above constraint is to re-structure the world picture, and hence the database One way of re-structuring is shown in **Fig. 3.** Use has also been made of the data constraint:

'No student can be both part-time and full-time'.

New keys have been constructed to take account of the new entities, but the numbering has been retained so that comparisons can be made. Now, validation of entries for the constraint consists of checking that entries in the 'part-time_students_taking_science_courses' do not duplicate a student key. This simplification of validation has been won at the expense of increasing the number of entity types. Student s3, whose status

is temporarily unknown, now appears twice, and it is imperative for the Data Management System to recognise this and update accordingly. It is known that grouping of data in a file system can have a profound effect on processing efficiency, and this is also true for validation processing.

2.7. *Constraints over sets*

A common kind of data constraint is that certain sub-total quantities must add up to a total held elsewhere in the database, for instance, the sum of wage fields in a personnel file may have to equal a wage bill total held in a departmental file. Here the constraint is over a whole set of entities, and over a whole file in the database. In the example of Fig. 2 this effect is shown by the constraint.

'No course can have more than 30 students'

One way of validating this condition would be to scan the 'student_taking_courses' relation once for each course, and during each scan to total the number of entries for that course. Some parallel operations are possible here, and they will be discussed later.

3. Measuring validation effort

It is desirable to have some estimate of the effort involved in validation whilst at an early stage of system design and before any details of hardware and software have been considered. For this purpose it is necessary to derive the estimate of effort directly from an analysis of the logical constraints and validation procedures. Such an estimate is possible if enough simplifying assumptions about the nature of the effort are made. This paper explores one simple approach where all the effort is associated with searching files. All other effort, such as central processor activity, is ignored. The effort is measured by the number of essentially sequential file searches; two searches which, in principle, could be done in parallel are counted as only one search. This measure is quite rough, but has the advantage of being independent of hardware and software implementations. Later, it will be shown that constraints which could be tested by file searching could also be tested by joining; the proposed measure is the same for this alternative basic technique.

Since the effort measure is to be derived from the constraint conditions it is first necessary to express these constraints in a precise way.

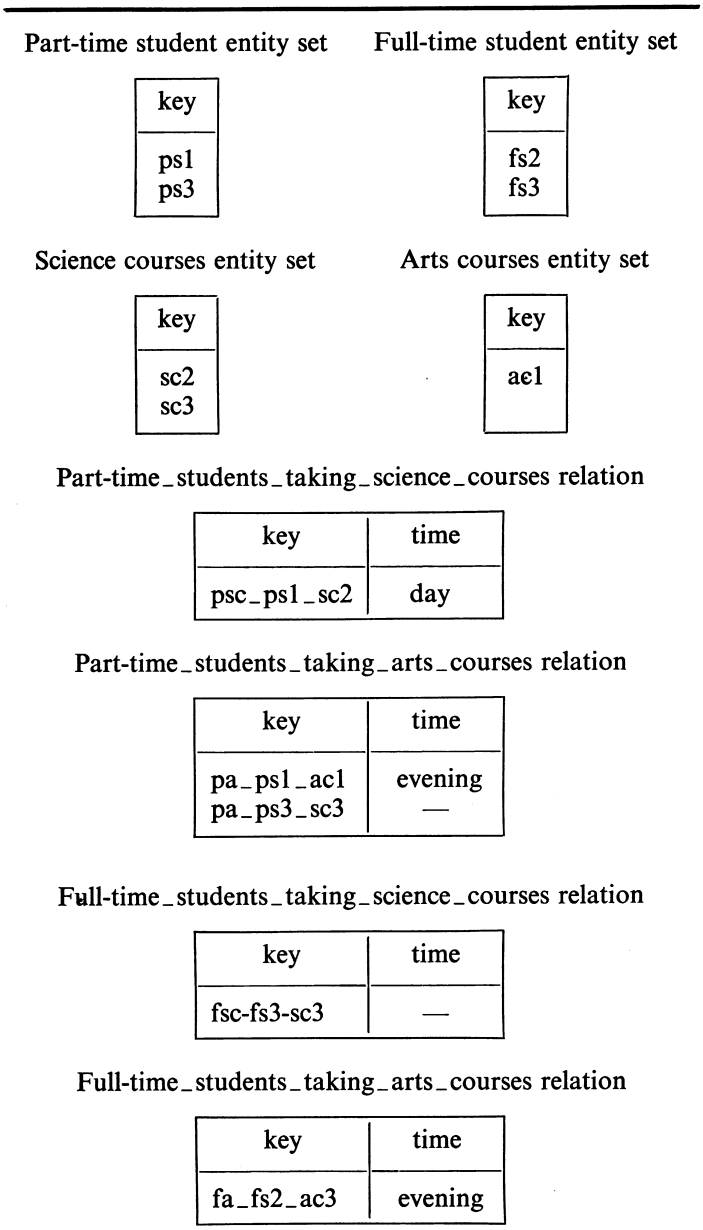

Fig. 3 Re-structuring of the database to take account of constraints

3.1. *Precise formulation of constraints and validation*

The precise expression of static consistency in the world description will be through a predicate 'is_compatible' which takes as arguments all the entity sets, relations, attribute sets and special individuals of the world. To see how this is done take the example of Fig. 2, and abbreviate the Student entity set to S, the Course entity set to C, and the Students_taking_courses relation to STC. In the situation without explicit constraints it was still required that before a student could enrol for a course, the student had to be in the student set, and the course had to be in the course set. Leaving out the attribute sets for brevity, this constraint is expressed.

$$\text{is\_compatible}(S, C, STC) \Leftrightarrow (\forall(s, c))\ [(s, c) \in STC \Rightarrow (s \in S) \wedge (c \in C)]$$

Suppose now that the world changes by the addition of (s', c') to STC, it is still required that

$$\text{is\_compatible}(S, C, STC \cup \{(s', c')\}) \Leftrightarrow (\forall(s, c))\ [(s, c) \in STC \cup \{(s', c')\} \Rightarrow (s \in S) \wedge (c \in C)]$$

In this simple example it is intuitively obvious that the validation test could be written

$$\text{is\_valid}(S, C, STC, (s', c')) \Leftrightarrow (s' \in S) \wedge (c' \in C)$$

However, by the mechanical application of the rules of some standard formulation of predicate calculus (for example in Mendelson, 1964), it would be possible to derive from the two above statements of 'is_compatible' that

$$\text{is\_compatible}(S, C, STC) \wedge (s' \in S) \wedge (c' \in C) \Rightarrow \text{is\_compatible}(S, C, STC \cup \{(s', c')\})$$

Thus, by taking the 'is_compatible' predicate before and after some change is made, it is possible to derive the validation test by logical manipulation. In practice, however, logical manipulation is so tedious that it is not possible to derive the validation test for any useful situation. In spite of this it is useful to recognise the underlying principles. In fact, the validation test so derived might not be sufficient because sometimes extra testing is needed, for instance, it might be necessary to have permission from a senior manager before a certain change is permitted. A more complete expression of the situation would be

$$\text{is\_compatible}(\text{Initial world}) \wedge \text{is\_permitted}(\text{change}) \wedge \text{is\_valid}(\text{change}) \Rightarrow \text{is\_compatible}(\text{New world})$$

Notice that the implication goes only one way and, as expected, a non-permitted, non-valid, change could still lead to a compatible world by a fortunate juxtaposition of errors.

In the practical situation the static consistency conditions will

be known, and the systems analyst, and database designer will devise tests by intuition. For the data constraint

'No part-time student may take more than one science course'

the static consistency condition is expressed

$$\text{is\_compatible}(S, C, STC) \Leftrightarrow$$
$$(\forall(s, c))\ [(s, c) \in STC \wedge (\text{status}(s) = \text{'part-time'}) \wedge (\text{faculty}(c) = \text{'science}) \Rightarrow (\forall c')\ [(s, c') \in STC \Rightarrow (\text{faculty}(c') \neq \text{'science'}) \vee (c' = c)]\ ]$$

where the attribute arguments have been omitted, and the property 'status' is assumed to be always defined. The validation is expressed as

$$\text{is\_valid}\ (S, C, STC, (s, c)) \Leftrightarrow$$
$$(\text{status}(s) = \text{'part-time'}) \wedge (\text{faculty}(c) = \text{'science'}) \Rightarrow (\forall c')\ [(s, c') \in STC \Rightarrow \text{faculty}(c') \neq \text{'science'}]$$

with the same conventions.

The set constraint

'No course may have more than 30 students'

is expressed with the aid of an auxiliary predicate, 'is_less_equal' which is expressed as

$$\text{is\_less\_equal}(STC, c, n) \Leftrightarrow [(n \geqslant o) \wedge (STC = \emptyset)] \vee [(STC \neq \emptyset) \wedge (\forall s)\ [(s, c) \in STC \Rightarrow (n \geqslant 1) \wedge \text{is\_less\_equal}(STC - \{(s, c)\}, c, n - 1)]\ ]$$

where it is understood that 'c' is a course, and 'n' is a positive integer or zero. The static consistency is now expressed

$$\text{is\_compatible}(S, C, STC) \Leftrightarrow (\forall c)\ \text{is\_less-equal}(STC, c, 30)$$

A recursive definition is used to check the properties of a set, this device can also be used to define a function over a set. The validity is

$$\text{is\_valid}(S, C, STC, (s, c)) \Leftrightarrow \text{is\_less\_equal}(STC, c, 29)$$

3.2. *Validation test procedures*

To discuss validation test procedures the hypothetical database with files corresponding to entity sets and relations will be assumed; the measures derived will still apply to any implementation. Logical validation conditions on the world model can be translated systematically into operations on the database: for example, '$(s, c) \in STC$' translates to saying that a certain keyed record is in file STC, the formula 'faculty(c) = "science"' translates to a test on a field of a keyed record. Formulae with quantifiers translate into file searches, in the formula

$$(\forall c')[(c' \in C) \wedge (s, c') \in STC \Rightarrow \text{faculty}(c') \neq \text{'science'}]$$

the quantification $(\forall c')[(c' \in C)\ldots$ translates to a full scan of the C file, selecting each record in turn, and carrying out the tests in the remainder of the formula; the tests indicated by $(s, c') \in STC \Rightarrow$ faculty(c') $\neq$ 'science' are the looking up of a record with a fully known key in the STC file, followed by testing the contents of the 'faculty' field.

If the above formula were re-written as

$$(\forall(s', c'))[(s', c') \in STC \wedge (s' = s) \wedge (c' \in C) \Rightarrow \text{faculty}(c') \neq \text{'science'}]$$

the translation would be different, although the overall effect would be the same. Here, the quantification $(\forall(s', c'))[(s', c') \in STC$ would lead to a scan of the STC file, on finding an entry with $(s' = s)$ the key of an entry in the C file would be retrieved. The C file would then be consulted with a known key.

3.3. *Validation effort*

To measure validation effort it is necessary to make some assumptions about the effort involved in basic operations. Here, the assumptions are:

1. Tests on record fields such as 'faculty(c) $\neq$ "science"', do not involve any effort once the record has been accessed.
2. Accessing a record with a fully known key does not involve effort. This gross assumption avoids entanglement with details of possible implementations of indexes, and is essential to avoid discussing implementation.
3. Looking for a record with an unknown, or partly known, key does involve effort. Here a record is specified by attribute, or by being related to another record.

This measure is associated with unavoidable searches in the database. In a constraint without quantifiers, such as 'age(e) $\leqslant$ 65' there is no reference to the database, and thus the validation could be carried out off-line, or at a remote terminal. Several searches can be involved in one constraint, for example in:

$$\text{is\_valid}(E, E1, E2, e) \Leftrightarrow$$
$$(\exists e1)\ [(\exists e2)\ [e1 \in E1 \wedge \text{name}(e) = \text{name}(e1) \wedge e2 \in E2 \wedge \text{age}(e) = \text{age}(e2)]\ ]$$

where the convention is introduced that En, Rn are entity sets and relations, e is to be inserted into E, and 'name', etc. are properties. Notice that here 'e' can only be inserted if its properties are known at the time of insertion, this is a typical situation. Now, this constraint can be re-written as:

$$(\exists e1)[e1 \in E1 \wedge \text{name}(e) = \text{name}(e1)] \wedge (\exists e2)[e2 \in E2 \wedge \text{age}(e) = \text{age}(e2)]$$

This form shows that the two searches can be carried out in parallel. For the measure used here this counts as a single search; but, if the constraint were:

$$(\exists e1)[e1 \in E1 \wedge \text{name}(e) = \text{name}(e1) \wedge [(\exists e2)\ e2 \in E2 \wedge \text{age}(e1) = \text{age}(e2)]\ ]$$

it would be first necessary to search the $E1$ file on the attribute 'name', when a candidate '$e1$' had been found it would be possible to search the $E2$ file. Here the searches must be in sequence, so that the measure is two searches.

These remarks apply to two searches of a single file, where the validation

$$(\exists e1)\ [e1 \in E1 \wedge \text{name}(e1) = \text{name}(e) \wedge (\exists e2)[e2 \in E1 \wedge \text{age}(e2) = \text{age}(e)]\ ]$$

involves one essential search, whereas

$$(\exists e1)\ [e1 \in E1 \wedge \text{name}(e1) = \text{name}(e) \wedge (\exists e2)\ [e2 \in E1 \wedge \text{age}(e2) = \text{age}(e1)]\ ]$$

involves two essential searches.

Validation conditions over sets will usually involve recursive statements of the conditions, as was illustrated above. Checking a recursive condition, or evaluating a recursively stated function over a set (file) does not change the search situation, what is new is that auxiliary storage is required to record the changing goals of the search. This can be seen in the 'is_less_equal' predicate defined above. As another example consider the function 'Σ' which acts on a personnel file, P, to sum the salary fields. The function Σ can be defined by saying how it acts on an empty set and then on a non-empty set

$$[(P = \emptyset) \wedge (\Sigma(P) = 0)\ ] \vee [(P \neq \emptyset) \wedge (\forall p)\ [(p \in P \Rightarrow \Sigma(P) = \text{salary}(p) + \Sigma(P - \{p\})]\ ]$$

Only one scan of the file is needed, but after each record is read a sub-total has to be held.

Complexity can be found by inspection of the validation conditions. To make the inspection easier the validation condition can be written so as to show only quantifier and test

positions with the entity and relation symbols involved, for example

$$(Qe1)\ [\text{test}(E1, e1, e)\ (Qe2)\ [\text{test2}(E2, e1, e2)]\ ]$$

where Q indicates a quantifier symbol. The variables and tests can be checked for their depth of nesting within quantifiers; predicate—'test'—above encloses variables of depth one, whereas, 'test2' has argument variables at depth two.

3.4. *Join operations on files*

In the above discussion the basic operation on files has been a search. An alternative basic operation is a join of two files, when records with fields meeting certain conditions are placed in the output file, the records being suitably amalgamated. Codd (1971) has discussed the join operation on two files; suppose files $F1$ and $F2$ contain records with fields, say $(f11, f12, .., f1n)$ and $(f21, f22, .., f2m)$. Certain fields in the two records contain data of the same type, say $f11$ is of the same type as $f21$, and $f12$ is of the same type as $f22$, but the other fields are of different types. Then JOIN$(F1, F2, f11, f12)$ produces a file with records having fields $(f11, f12, f13, .., f1n, f23, .., f2m)$ made up from the records of $F1$ and $F2$ which have exactly equal contents in the $f11$, $f21$ and $f12$, $f22$ fields. A validation constraint

$$(\exists e1)\ [(e1 \in E1) \wedge \text{name}(e1) = \text{name}(e)]$$

can be treated as a join test which can be written

$$\text{JOIN}(E1, \{e\}, \text{name}) \text{ is non-empty}$$

The output file so obtained will contain all the elements, which satisfy the quantification condition.

The JOIN operation is taken to act on actual files, and not on sets in the world model, through axioms. It would be possible to introduce algebraic operations in the data axioms, like union, intersection, set product, etc. and totally re-phrase the constraint conditions in algebraic terms; however, this has been found to be inconvenient in practice. It is, therefore, preferable to regard the JOIN operation as an alternative implementation of conditions expressed logically. As a further example the validation condition

$$(\exists e1)\ [e1 \in E1 \wedge \text{name}(e) = \text{name}(e1) \wedge (\exists e2)[e2 \in E2 \wedge \text{age}(e1) = \text{age}(e2)]\ ]$$

would be implemented by

$$\text{JOIN}(E2, \text{JOIN}(E1, \{e\}, \text{name}), \text{age}) \text{ must be non-empty}$$

This needs two join, and so the resulting join measure is the same as the previous search measure. Taking searches over into joins in this way will always produce the same number of joins as searches.

3.5. *Remarks on the validation effort measure*

The measure of validation effort has to be deduced from the form of the validation condition, quantifiers have to be recognised as giving rise to searches, or joins. Nested quantifiers give rise to essential sequencing of searches. This measure is entirely dependent on the written form of the validation condition, now logical formulae may be transformed into many truth-value equivalent forms, and some forms will be more economical in searches than others. An economical form will have a minimum nesting of quantifiers. The extent to which transformations may be carried out will depend on all the axioms which are available, which will include axioms on the data, as well as the constraint conditions. In an axiom system of reasonable power there will not be a universal method of transforming validation formulae into minimal search form, and each formula will have to be considered within the special circumstances of the problem.

The measure is very coarse, and a number of possibilities of refinement can be considered, for instance differentiating between universal and existential quantifiers. However, it seems that such refinements need more detailed consideration of data organisation, indexing, and hardware performance.

4. Use of the entity model

In the approach to consistency checking outlined above the most critical design decision is made when the elements of the applications world are labelled as entities, attributes, and relations. By this labelling the acceptable world changes are circumscribed, and some basic compatibilities are established. Also, all the possible insertions and deletions are settled and pre-formatted. In a practical situation it is not easy to label the elements of the application in this way, and many systems analysts may question whether the effort is worthwhile. The argument in favour of making such an effort is that it is always necessary to examine how errors could arise in the database through user mistakes. If this examination is done in a haphazard fashion a great deal of effort can be expended without any assurance that nothing has been overlooked. Construction of an entity model provides a disciplined framework for an examination, and every element is properly considered. Also, when new data types are added to the database there is a means of testing how they interact with data already present, and unexpected interactions can be detected.

It has to be admitted that sometimes the evidence for deciding whether an element is an entity, attribute, or relation will be based on hypothesising the database transactions. Such reverse reasoning will often be necessary when existing files must be incorporated in a new database, without change. In this circumstance it may be necessary to allow anomalous transactions, but these can be clearly recognised. It must be made clear that the sole aim of the world model is to provide a working framework for the design of validation procedures. It is obviously useful to database designers, but its fault is that it might impose extra burdens on systems analysts. Other methods for organising data might be found, which will have to be studied when available.

4.1. *Constraints and validation responsibility*

Constraint conditions can be of many different kinds, and the world situations described can be very complex; for example, if details on students, classrooms, and professors were kept in a database then the validation criteria could state that the stored data must represent a solution to the classroom scheduling problem. Of course, this criterion does not demand the production of a solution, but only that a check is carried out on a proposed solution. However, in this situation it might be argued that the user is wrongly off-loading his problem on to the database administrator. The resolution of this argument depends on all the circumstances of the application, but the assignment of responsibilities should be explicit.

A more technical consideration is the acceptability of the constraint. To illustrate this consider:

> 'A student may not enrol for a science course unless he is also going to enrol for an arts course within the next year'

Such a condition is beyond any effective checking (waiting for a year would not be accepted as a check). A reasonable limitation is that the validation should depend on searching files which actually exist. The above constraint refers to a file entry which does not yet exist. A more specific criterion is that the formal logical statement should refer only to files actually in the database, and that the criterion should be stated in a form which does not involve the complement of any file (entity set or relation). Another kind of logical difficulty is in circular dependencies, as an example suppose that in recording a joint banking account, details of the husband cannot be entered without details of the wife being present, and also, details of

the wife cannot be entered without details of the husband being present. It is clear that the only way that an entry can be made is for the details of both husband and wife to be entered at the same time. Such circular dependencies can arise when separate constraints are being combined, and they have to be detected as such.

4.2. *Use of mathematical logic*

It is usually agreed that systems design must involve creative imagination, but there is hope that more and more of the design effort will be carried on by routine techniques, in the same way that engineering design involves routine calculations. It is, therefore, pertinent to ask whether mathematical logic as used above could become a routine design tool. At this early time some tentative observations are:

1. Mathematical logic is obviously a tool for specialised theoretical studies.
2. Logical notation is a concise shorthand for expressing certain kinds of design information. It is reasonably easy to learn, and can therefore be expected to find some general use. It will be reinforced in use if certain types of database programming languages come into use (see Codd, 1971).
3. The direct manipulation of logical formulae is so tedious as to be unusable as a practical design tool. Methods of manipulating logical formulae by computer have been demonstrated by several research workers (see Robinson, 1967); the performance achieved so far, falls far short of that needed for system design. The future of such developments remains speculative.
4. Progress could be made by studying specific efforts, where concrete insights could be combined with logic. An example might be in developing tests to recognise circular dependencies in constraints.

At the present time it has been found that systematic attempts to express constraint conditions in logical notation are a valuable aid to examining the nature of the constraints. In particular, it is found in practise that various alternative ways of expressing the constraint become apparent, and this is valuable in providing more insight.

5. Conclusions

An introduction to one general aspect of the preservation of accuracy of stored data has been given by isolating the notion of the consistency of stored data. Accuracy needs in commercial applications have already given rise to several methods of input data validation which have been subjected to basic theoretical research. On the other hand, auditing of stored data is usually the result of an ad hoc collaboration between accountants and systems designers, and does not appear to have been the subject of systematic research. Since computerised systems are much more flexible than manual ones, it seems that a slow evolution of auditing practises will not be adequate to cope with changing and expanding demands. It will be necessary to devise an overall design approach to auditing that will allow the designer to produce effective audit procedures for new, and untried, systems.

Acknowledgements

The author would like to acknowledge stimulating discussions on the fundamental theory of databases with P. Hopewell (IBM UK Laboratories Ltd.), Dr. M. G. Notley and Dr. T. W. Rogers (IBM Scientific Centre, Peterlee). The application of logic to the theory of computation has been discussed with C. D. Allen and C. B. Jones (IBM UK Laboratories Ltd.). The author is also grateful for encouragement and comments from his colleagues Professor P. J. H. King and J. Inglis.

References

CODD, E. F. (1970). 'A Relational Model of Data for Large Shared Data Banks', *Comm. ACM*, Vol. 13, No. 6, pp. 377-387

FRASER, A. G. (1969). 'Integrity of a Mass Storage Filing System', *The Computer Journal*, Vol. 12, No. 1, pp. 1-5.

MENDELSON, E. (1964). *Introduction to Mathematical Logic*, Van Nostrand, New York.

ROBINSON, J. A. (1967). 'A Review of Automatic Theorem Proving', *Proc Symposia in Applied Mathematics*, Vol. 19, American Mathematical Society, Providence, R.I., pp. 1-18.

WILKES, M. V. (1972). 'On Preserving the Integrity of Data Bases', *The Computer Journal*, Vol. 15, No. 3, pp. 191-194.

Book review

Pattern Recognition, Learning and Thought, by L. Uhr, 1973; 506 pages. (*Prentice-Hall Inc.*, Englewood Cliffs, N.J. £7·30)

Professor Uhr has written an unusually *personal* book. In many ways, his text reads as though it were an interesting, albeit onesided, discussion with the author. The style is informal, colloquial, and jargon-free.

Nevertheless, the subject matter is treated thoroughly and it is possible to support the author's implied claim that he has written a useful, self-contained textbook 'as an introduction to simulation models of cognitive processes and artificial intelligence'.

The first nine chapters of this rather long book deal exhaustively with the great variety of programs which have been designed to solve pattern recognition problems. In common with many authors, Professor Uhr has concentrated his attention on character recognition techniques, and, within this rather restricted area, has produced an admirable survey. He introduces a programming language (EASEy) which he describes as 'a general-purpose list-processing, pattern-matching language . . . closely modelled on SNOBOL', and uses his language to provide programmed illustrations for all the main features of pattern recognition methodology. In this way, some of the apparent differences between similar programs written in different language structures, are ironed out.

Descriptions of useful applied pattern recognition systems, where 'production' pattern processing is being achieved, are not given, nor are the problems concerned with grey-scale pictures and scene analysis discussed. Thus, these chapters are thorough in their treatment of pattern recognition only up to the point where the problems begin to look insuperable.

The second half of the book looks at problem solving, game playing, theorem proving and (computer) learning methods. Finally, these various concepts are combined with the pattern recognition ideas to suggest flexibly structured programs which learn to recognise patterns in fairly simple classes of input picture.

There is a good bibliography (25 pages of references) and a short and rather unhelpful glossary. Anyone wishing to get a good idea from scratch as to what pattern recognition is all about could do a lot worse than study this book. Those who are old hands in the field will find the summary discussions, following each chapter, a useful survey of what can be done in recognition of line patterns. Incidentally, it is possible to read this book adequately without bothering to learn EASEy. Programs are summarised in plain language as well as appearing in full detail.

M. J. B. DUFF (London)