2. The archaeological database — new relations?

P.N. Cheetham
Department of Archaeological Sciences, University of Bradford, Bradford, West Yorkshire BD7 1DP, U.K.
J.G.B. Haigh
Department of Mathematics, University of Bradford, Bradford, West Yorkshire BD7 1DP, U.K.

2.1 Introduction

Over two decades have passed since the foundations of the relational data model were formalised (Codd 1970) and today a large number of Database Management Systems (DBMS) based on its principles are readily available. The better of these have attained a high degree of sophistication, running in a variety of environments — micros, workstations, minis and mainframes — and have achieved some standardisation through the adoption of Standard (or Structured) Query Language (SQL). As such, the user who invests much time in learning to use a DBMS and its development tools, for example INGRES, will have little problem when the present micro is dumped and a workstation appears on the desk. More importantly for archaeological information, the data, its structure, and application programs will also transfer with minimal upheaval. This is a salutary warning to those investing a great deal of resources in non-upwardly mobile micro-based DBMS and they are urged to consider employing either ORACLE or INGRES (the current flagships of the 4th generation language multi-environment relational DBMS) if they wish to ensure the longevity of their *work*. The reference to *work* rather than just to *data* is deliberate and the cornerstone of this paper, for information is not just data values; it is the context and meaning of those values that ultimately determine the usefulness of the data. Data structure, user interfaces, validation procedures, help systems and applications are inextricably linked with the raw data, giving it context and providing a crude but non-trivial 'knowledge base' without which data files may be useless, or even a negative resource, if misunderstood.

Although high-quality relational DBMS did not come into general use as commercial products until the late 1980s, deficiencies in the relational model had already been noted in the previous decade. Important new products are likely to become generally available soon. Many of the major research areas of general DBMS have direct application in the management of archaeological data. The aim of this paper is to discuss some of the limitations and deficiencies of currently available relational DBMS, to review informally the most relevant areas of development (and one area which has yet to be developed), and to consider the implications for mainstream archaeology.

2.2 The problem

Current DBMS are in reality crude, if effective, information handling systems. Their main thrust and success has been in the management of large data sets requiring extensive cross-referencing — i.e. 'information crunching', the IT equivalent of mathematical 'number crunching'. The need for this emanated from the commercial sector, where money was available to pay for developed products, and it was in meeting commercial requirements that most effort was expended. The relational model was seen as a satisfactory basis for commercial databases and was augmented with multi-user, security and recovery facilities. At a lower level of the system, query optimisation, file structures and indexing methods were all improved. The main trend in current development is towards distributed database systems which incorporate the same facilities.

The relational data model has been adopted for the majority of these products since the underlying model is simple — one of its strengths — and leads to DBMS of general application. Such DBMS have very little to offer over manual systems except in the scale of the information they can handle. As soon as one strays from structures which look like the SUPPLIER-PARTS-ORDERS or EMPLOYEE-DEPARTMENT-PROJECT type of examples used to illustrate relational database principles, trouble may ensue, as anyone who has attempted to implement a bibliographic database on a relational system may well appreciate. Current commercial relational DBMS are notoriously bad at handling unpredictable data and data structures, since they were intended to be used in a situation which is well understood and well structured. Archaeological data hardly fits these two descriptors. Archaeologists often start with the best intentions of producing a database of archaeological information, but generally end up with a database structured on subjective recording parameters with little or no real archaeological content. In terms of content, relational tables such as CONTEXTS or FINDS are just arbitrary collections of archaeological recording units whereas a commercial DEPARTMENT table is at least a meaningful relation which reflects the underlying organisational structure of a company. In point of fact, we have no idea at present, of how to structure archaeological information in a meaningful way — for it is the underlying organisational structure we are seeking to determine! (see Evans (1984) for a listing of possible imposed structures). This may seem trivial, but arbitrarily imposed structures can impose the same arbitrariness on the data and render it of little value in the future. Perhaps the most honest and natural structure is the PARTIPLE (*ibid.*) in which each entity is an 'owner' or attribute of another producing a 'possessive' hierarchy (partial ordering) but without the strictures of a true hierarchy. The relational model is perhaps not the best for this and entity relationship or object orientated approaches may prove more suitable, but this is beyond the scope of this paper and only noted to make the reader aware of the importance of developing meaningful data structures. One major stumbling block to any such structure is the presence of the fourth physical dimension in archaeological data — time.

The commercial need for correct up-to-date information initially discouraged any attempts to introduce time handling as a basic element of a DBMS. Extensive research effort has recently produced working systems to alleviate what is now an accepted deficiency. These new developments are highly relevant to archaeological databases. The need to record time as data also introduces the concept of abstract data types (ADTs) of which date and money are two of the few offered by commercial DBMS above the base types of integer, float and character. ADTs, such as date, provide their own operators, functions and return values e.g. in Date1 < Date2, the less-than operator equates with 'is before' and the system must recognise these as date types to produce a correct comparison result. For the expression date(Date2)-date(Date1), where date() is a function to allow the addition or subtraction of date types, the result is not an *absolute* date but a time *interval* expressed in years, months, days, hours, minutes and seconds. ADTs are important for defining format and domains for data values (see Ryan this volume), as well as meaningful operations performed upon that data. The definition of ADTs is highly relevant to archaeological information management because of their ability to bind meta data permanently within the DBMS system catalogues and not in individual application programs or support manuals. It is implemented on some experimental systems, but not openly available.

Finally the problem of imparting more meaning to the data is a wide ranging field including temporal and ADT considerations. It extends into the handling of null (missing) values — which in archaeology abound not only in quantity but type, procedural data types — which effectively hard-wire applications within the data, and expert/intelligent interfaces — which guide users through database operations while possessing additional abstract knowledge essential to the correct archaeological usage of a system. This leads on to the whole database/knowledge base overlap which the database and artificial intelligence (AI) camps have still not really come to terms with, although interfaces between the two systems are now available and are a highly topical area of research.

While all these considerations question the virtue of the relational model, in the medium term its simplicity, established worth, and availability have ensured some degree of stability. Many of the developments outlined below have been implemented as extensions to this basic model, often employing INGRES (Stonebraker *et al.* 1976) as the basis of the prototyping system. Much of this research emanates from the United States, often funded by the military and supported by large experienced teams. Since the research is highly complex and expensive to implement practically, archaeology may find it has little influence on the development of systems it may eventually inherit as one of its most fundamental tools.

2.3 Temporal databases

A fully fledged **temporal** database supports three classes of time representation and handling: *transaction* time, which emanates from the system clock (e.g. when a database record was last updated); *valid* time, the real world time when information is valid (e.g. the dates of an excavation or the date of publication of an analysis); and *user-defined* time (e.g. a conventional relation field holding say a radiocarbon date). Transaction time handling databases are termed **rollback** and are able to provide time varying snapshots of the database system itself. Valid time databases, termed **historical**, are concerned with the currency of the information content over time. A system which supports both rollback and historical, as well as facilities for user defined time is termed **temporal** (Snodgrass 1987). User defined time is an extension of data types as outlined in section 2.5, but it is pertinent to point out that the one usually provided as standard — the date type — is in itself woefully inadequate. The INGRES date type can handle absolute dates from 1 Jan 1582 (the year the Gregorian calendar was adopted in Europe) and time intervals of -800 to +800 years. Neither are particularly useful in archaeological or even historical work, suggesting that the designers are out of touch with many potential non-business users.

2.3a Rollback databases

Support of transaction time is extremely useful in archaeological situations. Current databases have the most annoying property that any updates overwrite and irrevocably lose former data values. Changes, intentional or unintentional, go unrecorded; although backup and journaling will allow a measure of transaction history support, the information is not readily available or in the correct form to support comprehensive rollback facilities. It is not unknown in archaeological recording to make occasional changes to paper records. It is normal practice, and in museum practice a stated guideline, that corrections should not obliterate former information in order to allow clarification of any ambiguities which may arise later. It is interesting to note a case where the computerisation of site context records has met resistance from experienced staff on precisely these grounds (Gordon 1991). Rollback facilities would therefore allow the user to view the whole database as it existed at any point in the past and, more usefully, to recover the transaction history of an individual record. It is in effect '...the history of database activities...', (Snodgrass 1987:250).

2.3b Historical databases

In many cases the difference between transaction time and valid time may be considered insignificant and some researchers choose to ignore it (Copeland & Maier 1984). For example, if we wished to study the degradation of monuments from *condition* information within sites and monuments records, then a transaction history query would return changes in the *condition* of a particular monument. It may return corrections as well, although there is some disagreement about the status of corrections as opposed to information updates in temporal systems. Snodgrass' definition of an historical database specifically excludes past states of the database. In the above example, *condition* information should be associated with a date at which the site was examined. This date becomes the *valid* time and is considered to remain valid until a new condition status and associated *valid* time are defined.

Valid time and *transaction* time can vary markedly. If a user wished to obtain condition information about a group of sites at a particular date (e.g. What was the state of preservation of certain barrows in 1975?), then *transaction* time would be irrelevant if the information was only recently computerised, or the records appended and updated at varying times. We would like to be able to frame queries such as:

```
/* Pseudo SQL query */
SELECT monument_no,condition
FROM SMR_record
WHERE parish='some_parish' AND
      monument_class='BARROW'
AT_VALID_TIME=1975
```

Without the final clause, using the conceptual language extension AT_VALID_TIME, the system would default to NOW and appear as a conventional (up-to-date) database to the user, (note: the date 1975 is not a conventional field value of date or other type, but a *valid* time date stamp on a tuple). In a conventional 'snapshot' DBMS a separate relation for condition and date of condition would have to be created and the same query would require the following SQL code:

```
/* Join tables on same SMR number */
SELECT s.monument_no,c.condition
FROM SMR_record s,condition_rel c
WHERE s.monument_no=c.monument_no AND
      s.parish='some_parish' AND
      s.monument_class='BARROW' AND
      c.date <= 1975 /* in or before '75 */
/* Check if latest up to 1975 */
AND NOT EXISTS
      (SELECT *
      FROM condition_rel r
      WHERE c.monument_no=r.monument_no
            c.date < r.date AND
            r.date <= 1975)
```

which is more cumbersome, and a subselect similar to the one in the example would be required on a join to retrieve the current condition!

The first example query is a gross simplification of language extension requirements of a temporal DBMS. For detailed information on temporal databases the reader should consult Snodgrass (1987) which contains a most readable introduction to basic concepts, a detailed presentation of a temporal query language TQuel and a comprehensive bibliography on the topic.

In both query examples the valid time was assumed to be known, as the information is not archaeological but administrative in nature. If valid times could be assigned to archaeological data proper, then we could apply powerful temporal query language operators such as **overlap**, **precede**, **equal** and **extend**. TQuel considers time as intervals rather than continuous. It is difficult to represent instantaneous events and in practice a valid time granularity (e.g. seconds, months or years) must be imposed and consequently any interval smaller than the granularity is considered instantaneous; conversely, truly instantaneous events are considered to extend over the granularity interval (Snodgrass 1987). Archaeology needs multiple time granularity as, while landscapes and material cultures may change slowly, almost instantaneous intra-site activities can be detected. While absolute valid time is unlikely ever to be available for most archaeological data, relative dating is. Unfortunately current research assumes absolute dating is available. Since TQUEL only supports historical states per record rather than per field, all historical information on individual fields must be recovered from the sequence of valid times for entire records. What TQUEL and other systems fail to address is potential changes over time to the underlying structure of the database and individual records, another area of instability frequently encountered in connection with archaeological information.

In a temporal database it can be seen that time varying information can be handled in a cumulative way. Changes to subjective data such as site dating or classification can be become current without eradicating former views and we are therefore in a position to build on our information base without constantly replacing it.

2.4 Null values

The useful and predictable handling of null values within a database is highly desirable. These null values should not be identified too closely with missing values (see Smith 1984 for a discussion of missing values in an archaeological context) for there are instances when a database field may be blank but the data is certainly not missing! Some DBMS support a null value and it is recognised in SQL, but it is an all embracing and problematical concept as implemented in SQL (Date 1986). Date identifies two varieties of nulls:

1. A tuple is incomplete because field data values are unknown.

2. A field is inapplicable (e.g. an area for a linear feature or volume of a surface).

Date's analysis underestimates the variety of null 'values' possible, and certainly encountered, in archaeological situations. He recommends an alternative default-value approach to nulls in which fields are initialised to a default value defined in a field specification; he regards this as a more intuitive approach than the one used in SQL. An example of default-value specification is:

```
DECLARE weight(float) .. DEFAULT (-1.0)
/* A variation on Date's example syntax */
```

This is the way most archaeologists approach the problem but in an undeclared way and relying on the user and application programmer to know the null default value and to exclude it where appropriate. For example in SQL:

```
SELECT AVG(weight)
FROM artefact
WHERE weight != -1.0
```

which would return the average weight excluding any nulls from the calculation. The advantage of declared default values is that: a) The default value is explicitly stated in the table declaration and therefore internally documented and b) a certain amount of automation could be achieved in that the WHERE clause condition could be replaced as follows:

```
SELECT AVG(weight)
FROM artefact
WHERE NOT_DEFAULT(weight)
/* Not standard SQL */
```

obviating the need for the user to know the actual default value. Besides NOT_DEFAULT(ATTRIBUTE), other language extension functions such as NOT_DEFAULT_SUM(ATTRIBUTE) or NOT_DEFAULT_AVG(ATTRIBUTE) could be included to allow standard aggregate functions to be evaluated in the presence of nulls without additional programming.

While this declared approach has some merit, it has two distinct disadvantages which Date fails to consider. The first is the problem of data transfer between systems. This is of great archaeological concern as its information base is essentially cumulative and old data will remain valid alongside new data. Default values in Date's scheme are actual data values and therefore will transfer their values into other data sets, leading to possible misinterpretation. In some instances the default value of a null may have to be assigned on the assumption that the data will never actually need to take on that value; this may be more of a problem for character data than numerical. Date considers only the first of his two possible null situations; in fact there are several possible reasons for data to be 'missing' and it is sometimes important to include the relevant reason as valid data within the system. In an archaeological context, particularly in research planning, it is often desirable to know just why data are absent. As databases become larger it becomes ever more critical that the response to a query should be exactly what was intended. The concept of 'unknown' without qualification is counter-productive and in the case of sensitive information such as SMR data, lack of explicitness or consequent mis-interpretation may have legal repercussions.

2.4a Archaeological 'unknowns'

The range of reasons for which data require null values within a database is largely governed by practical considerations. A list might include:

a) *Not available at this time.* This implies that the record is not yet fully completed but will be eventually, i.e. in an inconsistent state of update similar to that of Date's first type (see above) but not precisely equivalent as it does not mean unknown *de facto*, but unknown to the system at the current time.

b) *Inappropriate attribute.* This is exactly as in Date's second case and abounds in archaeological databases where generalised recording schemas are often a necessity. Decomposition of relations can eliminate this problem but may excessively complicate applications and affect efficiency. The inappropriate attribute could be termed a definitive null (i.e. a void). In some cases this will be self-evident; in a context recording record if a context is a *cut* then obviously soil description fields are redundant. In other cases such redundant fields may not be obvious, for example: stratigraphic physical relationships may include a *butts* field and it would be more appropriate to state explicitly that no butt existed, rather than to assume that the absence of a value implies this and is not simply a recording oversight. Such void fields could also simplify data capture and retrieval. In the stratigraphic case it would not be necessary to include a CONTEXT_TYPE (e.g. Cut or Layer) field as this would be reflected in the voiding of the inappropriate attributes. In retrieval a void attribute could be used to trigger a more appropriate response. For example, in a sites and monuments database, if an owner was also effectively the tenant in the sense of an initial site contact, then a pseudo SQL query could be formed such that:

```
SELECT SMR_number,site_contact=tenant
FROM SMR_file
/* Assuming the system defaults to ignoring tuples
returning VOID fields in the field list */
UNION
SELECT SMR_number,owner
FROM SMR_file
WHERE VOID tenant
/* Note. For this UNION to work the tenant and owner
field must be of compatible types */
```

This avoids the need to duplicate details and to retain explicitly the status, either by considering the owner as the tenant (repeating the owner details in the tenant field) which is not really the case or by explicitly stating in the tenant field 'AS OWNER' or 'NO TENANT', which requires the user to be conversant with the actual field values and complicates input validation. Note that in this example, null values other than void should be returned to make the user aware that the site record was incomplete.

c) *The attribute value not available.* This could occur in cases where for some reason data are lost and may never be replaced. Examples would be the present location of an object which has been lost or site find co-ordinates which were never recorded. While in character fields it may be possible to specify 'LOST' or 'MISSING', in numerical fields one has to resort to obscure default values which can differentiate this truly unknown value from the other cases in this list.

d) *The attribute value was, but is not now, available.* This type of null or unknown occurs in cases where the presence of an attribute is known but its value is not. For example, if an urn was known to be decorated but the detail has been lost then attribute fields for the decoration detail should exist but with null values.

Relation **pot_type**

pot_no	*pot_class*
01	URN

Relation **dec_details**

pot_no	*dec_att1*	*dec_att2*
01	?	?

In which case it would be quite wrong to assume that a listing of decorated pot selected in SQL as:

```
SELECT p.pot_no, p.pot_class
FROM pot_type p,dec_type d
WHERE p.pot_no = d.pot_no
```

always corresponds to a usable tuple in the **dec_details** relation. It would be possible to add an additional attribute field **decorated** (Yes or No) to the **pot_type** relation but the absence of a referenced tuple in **dec_details** would not give the user any idea of the reason for its absence. The null indicator given to the attributes in this situation could be identical to those in (c) but it is the need to declare an effectively null tuple (excepting its primary key) that differentiates this instance from tuples which have blank fields for other reasons.

e) *The binary logic unknown*. While value unknown may be replaced by MISSING in some contexts (e.g. an unknown weight, for we know that a object must have a weight) we come up against three part logic in situations of presence or absence. In the above example of the decorated urn, what if we were unaware as to the presence of decoration? If we employed a **decorated** attribute then we would have to cope with the possibility of null value meaning MAYBE decorated. Once again this could be, and if the **decorated** field was omitted certainly should be, reflected in the **dec_details** relation as a MAYBE tuple which cannot be represented in the same way as the unknown attribute value tuple, when the presence of decoration is known but not actual values.

f) *Conscious omission of an attribute*. If an attribute is deemed inappropriate by criteria not reflected in the recording schema, then this should be explicitly indicated. For example a comprehensive site finds recording system will include three dimensional co-ordination details. It is still common practice, on sites with large numbers of finds, to be selective, often unpredictably, in whether a find receives full co-ordination, two dimensional or none at all. The attributes are not inappropriate in the sense used in (b) above and not unknown but OMITTED which takes the meaning, for good or bad, that the attribute value may or may not have been available, but was ignored. The quality of the data is therefore open to better scrutiny and assessment. An unknown value may indicate poor excavation or recording, an omitted value should reflect the strategy of the excavator. More generally, the OMITTED null would allow sub-sets of records from different workers to undergo UNION with non-common attribute columns inserted with OMITTED nulls giving a much clearer indication of the information content and possibilities for further enhancement.

Perhaps more than any other discipline, archaeology has to cope with irretrievably incomplete information. Destroyed sites cannot be restored, long dead antiquarians cannot be interrogated, a lost portion of an artefact is often never found and recording standards either do not exist or are constantly 'adjusted' to suit the individual. The most we can hope to do is to exploit fully the information we do have. In this, the comprehensive handling of 'unknowns' in database systems is an essential goal.

2.4b Internal representation of null values

With the default value approach of Date, null representation relies on valid data of unique form. This could be adapted to suit multiple null types, as for example:

Null Type	*Character*	*Numerical*
INCOMPLETE	'!!!'	-1
MISSING	'???'	-2
VOID	'###'	-3
OMITTED	'XXX'	-4

These then behave as any other values and may be used in conditional statements in SQL queries to obtain the required information. For example if a researcher wished to get a listing of stone axe-hammers which could then be used for study, the following query could be formed:

```
SELECT artefact_ref,artefact_location
FROM general_file
WHERE artefact_type='AXE-HAMMER',
      material='STONE',
      artefact_location !='???'
```

The researcher would then get a listing showing which axe-hammers were immediately accessible or may be accessible, excluding those which were not. This would, therefore, put the researcher in a much better position to assess and plan any proposed work.

Another approach is hidden fields which indicate the null status of a visible field. This would add little to storage requirements and is a method used to represent null valued fields in some SQL implementations. A variation on this could be that the null status is part of the field representation at storage level and is interpreted appropriately when field values are retrieved from storage. Both these have the problems of how actually to make the user aware of the type of null represented during screen operations, in output and how to input them. They also require modifications of the DBMS at the lowest levels and require query language extensions to handle them. How they would be downloaded to transfer to another system, preferably in ASCII format, is also problematical and an area where standards would need to be implemented.

It appears that the question of null valued fields is not one of the most active areas of DBMS research. Perhaps more fruitful solutions may be found through the exploitation of AI/DBMS interfaces considered in section 2.7.

2.5 Abstract Data Types

As previously mentioned above *abstract data types* are extensions to the data types provided within current DBMS, the *date* type being quoted as one of the few provided as standard above the primitive character, integer and float types. Few systems have facilities for defining new types and these are experimental or not generally available. Most information is abstract but can be represented as a basic type. The DBMS has little internal knowledge of the meaning of the data and will happily perform aggregate functions, such as average or sum, on any field of numerical type, whether this is logical or not (e.g. sum site context reference numbers or perform a union on weight and length attributes).

These are then virtual data types which only have full meaning when utilised in applications where this meaning is fully understood by the user. Thus an Ordnance Survey grid reference (OSGR) may be stored as type character e.g. 'SE4530056700', but it is up to the user to provide within his application appropriate code to convert this representation into cartesian co-ordinates for plotting or comparison. How much more useful would it be to represent a grid reference as an internal data type? If the representation were well implemented, we would expect to be able to input or retrieve a grid reference in more than one format, e.g. as a conventional OSGR, absolute eastings and northings, or even in universal co-ordinates of latitude and longitude. We would also expect a range of functions and operators to be provided, including a function to return the distance between two grid reference points (see Ryan this volume). The obvious advantage is that application programming becomes less procedural and hence simpler; the implemented code is application independent and fully portable to other sites using the same DBMS. For archaeology this would help to utilise scarce resources and expertise more productively, while increasing the explicit meaning of stored information.

ADTs have limitless applications; another useful example would be a statistical date type which could be declared and used as in the following illustration in extended SQL syntax:

```
/* declare table specification */
CREATE TABLE site_records (
      site_name vchar(30),
      .
      .
      radiocarbon_date stat_date)
/* field named 'radiocarbon_date' of abstract type
'stat_date' */

/* example query */
SELECT contemporary_sites=site_name
FROM site_records
WHERE radiocarbon_date = 3000:2
/* where 3000 is in years B.P., ':' is a separator and 2 is
the specified number of standard deviations */
```

producing a listing of sites with radiocarbon dates which at 2 standard deviations overlap 3000 BP (problems of corrected/uncorrected etc. are glossed over to keep the example clear). In addition, a user defined ADT could also provide internal validation as in:

```
CREATE TABLE pot_attributes(
      pot_ref uniq_ref_type('XXNNN.NN'),
      .
      .
      fabric_code char_code('XX': 'F[A..H]'))
```

where the parentheses contain format specifiers and permissible values. These specifications can then be used by the system in applications, to provide validation and compatibility error messages during compilation and run-time, easing application development and, perhaps more importantly, internally documenting the specifications.

While extendibility seems desirable for archaeology there are some problems. EXODUS (Carey *et al.* 1986), a prototype extendible DBMS, is essentially a powerful modular system — kernel DBMS facilities plus software development tools. Such a system is designed to create highly application-specific DBMS, but the cost is the high level of expertise required of the database implementors and the cited paper makes it clear that this complex task would be beyond most end users. This is echoed by Stonebraker (1988) when he notes:

> 'It is clear that software houses may have the sophistication; however, extensions may prove daunting for the less-skilled person', (p. 478).

This may be the great stumbling block to their development for use in archaeology and a drastic change from the do-it-yourself simplicity offered by currently available DBMS. One worry is that data could become so intimately linked to a particular extended system that transfer of even the raw data without degrading the 'knowledge base', never mind any ADTs or procedural fields (see section 2.6), becomes a non-trivial if not impossible task. None of the papers on extendible databases address the problem of transportability of complex data structures and extended data types, which is impractical and short-sighted in the ever-changing world of DBMS products.

2.6 Procedures as database extensions

While abstract data types may be appropriate in cases of simple data types such as a single OSGR or statistical date (see above), more complex types (e.g. vector data for shapes) can be handled by procedures. This introduces a field type which can take on values that are collections of commands in a supported query language (Stonebraker *et al.* 1987a). The method is argued to retain the simplicity of the relational model while addressing situations where the model has been considered inadequate. Among a wide range of benefits (*ibid.*), of particular note are the ability to store queries and to handle data of unpredictable composition. Stored queries could be applied to store domain specific algorithms and definitions of views as well as frequently encountered conventional queries. In the case of unpredictable composition it would be possible to tailor attribute relations without generalised schemas thus:

```
CREATE finds(
      find_no=i2,
      strat_unit=i2,
      find_attributes=procedure)
```

where relations of find types might be:

```
roof_tile(find_no,fabric_type,length,breadth,...)
floor_tile(find_no,dec_method,design_code,...)
coin(find_no,level,coin_date,obv_inscription,...)
```

and a typical finds record could then be:

```
find_no=1022
strat_unit=724
find_attribute="SELECT *
            FROM ROOF_TILE
            WHERE ROOF_TILE.find_no=1022"
```

This allows retrieval of a particular find together with its appropriate attributes. The approach would create

unnormalised relations if finds with differing attribute relations were retrieved and so applications programs must be written to accept the more complex form of the tuples. The advantage of stored procedures is that it goes some way towards binding meta data (information about the relationships between relational tables and their intended usage through stored queries and stored algorithms) within the database itself in a standardised and transferrable format. The logical extension of this idea is the coupling of expert systems to DBMS.

2.7 Expert systems and relational databases

The potential of linking DBMS to expert systems (ES) has long been recognised but because of a traditional split between knowledge engineering and database design the marriage of the two is a relatively recent development (Higa & Liu Sheng 1989). The basic premise is to get the advantages of fully developed data management facilities with the facility of semantic support in an intelligent front end or user interface. This may be in a loosely coupled form with the intelligent front end issuing requests for data to the DBMS. 'To the DBMS, the ES appears to be just another user', (Fishman 1986:92) — the DBMS simply fills in the blanks as requested. Alternatively in a tightly coupled form the DBMS is the source of the ES database. i.e. relations are database predicates used to directly evaluate goals. The future aim, totally integrated knowledge-based management systems, is still some years off, although some DBMS are providing rules systems as in POSTGRES (Stonebraker *et al.* 1987b) and Prolog/DBMS interface facilities are available on most platforms including microcomputers. This paper can only give a flavour of how these developments may be applied in archaeological situations.

The role of expert systems in archaeology is controversial possibly due to over-expectation of their usefulness. It has been suggested that ES will fossilise knowledge, but conversely that one system needs to be built, verified and agreed upon (e.g. in samian identification), before an advance has been made (Baker 1988). The author regards this view as quite blinkered and unappreciative of the possibilities. The value of expert systems, if used as the basis for intelligent interfaces to databases, is quite the reverse. Rather than one agreed classification system (which is probably wrong anyway!) ES could support multi-expert views of a common dataset making it possible to classify according to expert A or expert B. As further data are added and inconsistencies between A and B studied, a further expert, C, could be added thus building on, but not replacing older, and still possibly valid, systems. In some cases it may be possible to build rule bridges across similar but not identical datasets so that they may be used as one, maintaining the independence of workers to evolve their techniques without being hampered by out-moded standards. As a simple illustrative example, consider a cropmark classification system (after Edis *et al.* 1989) where a COMPLEX is a combination of types ENCLOSURE, LINEAR FEATURE, LINEAR SYSTEM, MACULA, without differentiating between discrete or superimposed. Another system (Palmer 1983) differentiates between discrete or superimposed using the terms CLUSTER COMPLEX and SUPERIMPOSED COMPLEX. By applying the following rules:

> if system Edis and type COMPLEX then type(Palmer) is CLUSTER COMPLEX or SUPERIMPOSED COMPLEX
>
> if system Palmer and type CLUSTER COMPLEX or SUPERIMPOSED COMPLEX then type(Edis system) is COMPLEX

which can be expressed in prolog as:

```
/*Site type(Site ref.,Classification system,Site Class)*/

/*Convert Palmer system to Edis*/
type(Ref,edis,complex):
      type(Ref,Palmer,superimposed_complex);
      type(Ref,Palmer,cluster_complex).
/*One solution*/

/*Convert Edis system to Palmer*/
type(Ref,Palmer,superimposed_complex):
      type(Ref,edis,complex).

type(Ref,Palmer,cluster_complex):
      type(Ref,edis,complex).
/*Two solutions*/
```

This provides a naive example (the Prolog would not actually work correctly in practice) which illustrates both the reduction of information (Palmer to Edis system) or uncertainty (Edis to Palmer system) resulting from such rule bridges. Such assessment may lead to the development of minimum standards for record structures to avoid some problems, but not to stifle new approaches requiring variations from the standard, thus making past work partially useful or at least assessable.

2.8 Concluding notes

A recurring theme throughout this paper is the acceptance of the underlying axiom that archaeology (as a set of accepted methodologies applied to a standardise information base) is, and may always remain, undefinable, and that there is no correct or feasible way of encapsulating a system which can cope with the variety of problems associated with the data it produces or the methodology it applies in its analysis of those data. As such, we must develop open-ended systems which allow continual update and feedback so that the results, inevitably interim, can be accessed intelligently by future generations. Computerisation in any form can only be considered acceptable if it meets this criterion. Bridging the gap between data and the information that data constitutes is perhaps the greatest challenge to the archaeological application of IT. Archaeological database systems, in their present form, are poor contenders to meet the challenge. Although we may have to wait for off-the-shelf systems to become available, there is no reason why the present technology cannot be used or developed within the discipline provided we are aware of the limitations — with the right approach we may even be able to aid in the development of general solutions as archaeology can be considered stimulating ground for the computer scientist. What we must avoid is unjustifiable 'black box' systems or we may find the discipline tarred with the familiar '...manipulation of ambiguous data by means of dubious methods to solve a problem that has not been defined,' leading to the ultimate indictment of '...lies, damned lies, and archaeological information.'

Acknowledgments

The authors wish to acknowledge the aid of Nick Ryan in providing certain information and references to recent DBMS developments. The content is however, the full responsibility of the authors alone and is the result of background work for a SERC funded PhD, at the University of Bradford. An oral presentation of this paper was given at the 1990 Computer Applications in Archaeology Conference at Southampton.

References

BAKER, K.G. 1988. "Towards an archaeological methodology for expert systems", in C.L.N. Ruggles & S.P.Q. Rahtz (ed.), *Computer and Quantitative Methods in Archaeology 1987*, British Archaeological Reports (International Series) **393**, Oxford, British Archaeological Reports: 229–236.

CAREY, M.J., D. FRANK, MURALKRISHNA, D.J. DEWITT, G. GRAEFE, J.E. RICHARDSON & E.J. SHEKITA 1986. "The architecture of the EXODUS Extensible DBMS", in *Proceedings of the Object-Oriented Database Workshop*: 52–65.

CODD, E.F. 1970. "A relational model of data for large shared data banks", *Communications of the ACM*, **13**(6): 377–387.

COPELAND, G. & D. MAIER 1984. "Making Smalltalk a database system", in *Proceedings (1984) SIGMOD Conference*: 316–325.

DATE, C.J. 1986. *Relational Database: Selected Writings*, London, Addison-Wesley.

EDIS, J., D. MACLEOD & R. BEWLEY 1989. "An archaeologists guide to the classification of cropmarks and soilmarks", *Antiquity*, **63**: 112–126.

EVANS, D.M. 1984. "A National Archaeological Archive — computer database applications", in *Computer Applications in Archaeology 1984*, Birmingham, University of Birmingham: 112–118.

FISHMAN, D.H. 1986. "The DBMS-Expert System connection", in G. Ariav & J. Clifford (eds.), *New Directions for Database Systems*, 87–101.

GORDON, S. 1991. "How safe is your data?", in K. Lockyear & S. Rahtz (eds.), *Computer Applications and Quantitative Methods in Archaeology 1990*, British Archaeological Reports (International Series) **565**, Oxford, Tempus Reparatum: 75-79.

HIGA, K. & O.R. LIU SHENG 1989. "An Object-Oriented methodology for database/knowledgebase coupling: an implementation of the structured entity model in Nexpert System", *Data Base*, Spring 1989: 24–29.

PALMER, R. 1983. "Analysis of settlement features in the landscape of prehistoric Wessex", in G.S. Maxwell (ed.), *The impact of aerial reconnaissance on archaeology*, Council for British Archaeology Research Report **49**, London, Council for British Archaeology: 41–53.

SMITH, D.J. 1984. "Some problems of missing data in archaeology", *Science and Archaeology*, **26**: 15–16.

SNODGRASS, R. 1987. "The Temporal Query Language TQuel", *ACM Transactions on Database Systems*, **12**(2): 247–298.

STONEBRAKER, M. (ed.) 1988. *Readings in Database Systems*, California, Morgan Kaufmann.

STONEBRAKER, M., J. ANTON & E. HANSON 1987a. "Extending a database system with procedures", *ACM Transactions on Database Systems*, **12**(3): 350–376.

STONEBRAKER, M., E. HANSON & C. HONG 1987b. "The design of the Postgres Rules System", in *Readings 13th International on Very Large Databases*.

STONEBRAKER, M., E. WONG, P. KREPS & G. HELD 1976. "Design and implementation of INGRES", *ACM Transactions on Database Systems*, **1**(3): 189–222.