1. Beyond the relational database: managing the variety and complexity of archaeological data

Nick Ryan
Computing Laboratory, University of Kent at Canterbury, Canterbury, Kent CT2 7NF, U.K.

1.1 Introduction

The theme of the large scale integrated database system combining many, if not all, aspects of some archaeological activity, such as the excavation process, has been with us for a long time. Starting in the days when archaeologists talked about 'data banks' rather than databases, the primary concern seemed all too often to be with the data entry process. If only the data could be fed into the machine in an appropriate form, then it would be possible to retrieve any required information at the touch of a button. There were, of course, honourable exceptions and this conference has heard many papers concentrating on the use of data in which the database was treated as an incidental if necessary component of a larger system, one that enabled other work rather than being an end in itself.

By the mid-1980s many archaeologists had come to appreciate that there was a real danger that electronic storage would simply provide an alternative to the dusty museum basement as a repository of unorganised and rarely consulted archives. Some were disturbed by the combination of computer storage and an alleged tendency to record '...everything in sight...' (McVicar 1986:102). There was perhaps a danger of creating an electronic equivalent of the earlier crisis in physical recording and dissemination that in Britain had led to the 'Cunliffe report' (CBA/DOE 1982) and its subsequent effects on funding and working practice. Since then, discussion of large scale archaeological databases has been marked by a significantly increased concentration on established design methods that emphasise not just storage, but also the intended uses of the data. This has resulted from many influences, not least of which are the increasing professionalism in the use of computers by many archaeological organisations, and the continuing increase in the capabilities of the available technology.

Perhaps now that multi-user networks of microcomputers and even workstations are appearing in some organisations, and powerful DBMS rather than the earlier generation of overgrown file management systems are available, we are now in a position to contemplate the development of large scale archaeological information systems. Certainly, there have been presentations at this conference during the last two years that suggest that this is possible.

Two examples from the 1990 CAA conference stand out. In an as yet unpublished paper, Andresen and Madsen presented a detailed account of a data modelling exercise aimed at providing a sound basis for a flexible recording system. Boast and Chapman (1991) described one aspect, the processing of stratigraphic data, of a longer term project to develop a large scale archaeological information system at the DUA. Elsewhere, in an article in *Antiquity*, Boast and Tomlinson (1990) have discussed the general requirement to integrate text, data archive, graphics and images that underlies this project.

In this paper I will argue that although the current technology is adequate for producing such a system, it is far from ideal. A rigid separation of tasks and data types between different packages extends to DBMS. Although often claimed to be general purpose systems, this is only true if we accept a very limited view of the general, or are prepared to invest considerable programming ingenuity in our applications. Even then, the result will be a level of performance and flexibility that falls far short of the capabilities of specialised packages designed to deal with, for example, text or graphical data. Without a truly general purpose DBMS capable of managing any type of data with equal facility, any attempt to produce a large scale integrated information system will require extensive programming and present problems for future software maintenance. According to all the text books, these are precisely the problems that DBMS are intended to overcome.

Archaeologists are not alone in facing these problems. Commercial data processing has traditionally been conceived in terms of large numbers of simple transactions applied to regular collections of records. These, often minimal, descriptions of entities contain regular combinations of simple atomic data types. Recent trends in organisational computing and office and plant automation have led to the realisation that this simple view of data processing requirements is an illusion brought about by the limited range of past and contemporary uses of computers. Once the variety of applications increased it became clear that to provide centralised control of information resources and an effective implementation of management information systems it was essential to provide a coherent means of storing, manipulating and accessing a wide variety of data forms. These include office data such as memos, letters, electronic mail, CAD/CAM data, spatial or geographical data, and so on, as well as the more familiar, and highly structured, stock control and personnel data. There is a clear analogy here with the expressed desire to construct archaeological information systems.

Much recent research in database systems has sought to address these problems in a number of ways. Although representing differences of approach, the extensible RDBMS (Stonebraker 1989), the Object Oriented (OODBMS) (Kim 1990) and the Non First Normal Form (NF^2 DBMS) (Dadaram & Linneman 1989) all attempt to provide a more generalised modelling environment able to deal with a wider, ideally open-ended, range of data types. Each attempts to go beyond the storage and maintenance of simple data processing records to provide a uniform system for managing all of the organisational data.

In addition to data structure, both object oriented and extensible relational systems provide methods of storing behavioural rules and processes associated with data types, and thus encapsulating a considerable part of the semantic content of data and functional models in the database rather than in the application programs. The database facilities for maintaining the security and integrity of data, as well as the ability to inter-relate data and process in a single coherent model, can be applied to the wide variety of data types found in any complex organisation. This work is leading to rapid evolution of current products to cater for a wider range of data types, and to the development of new forms of DBMS that promise to overcome the limitations of relational systems.

1.2 Integrated systems: current possibilities

Huggett (1989) discussed the question of integration and showed how a careful choice of packages could help to overcome many of the problems encountered when using the wide variety of software necessary for an archaeological project. He suggested that the most suitable packages were those offering a means of customising their operation and user-interface, together with an ability to read and write each other's file formats. Despite the benefits of this approach, Huggett made it clear that this was only a partial solution to the problem, albeit a pragmatic one suited to the limitations of the hardware, software and human expertise available to many archaeological organisations. In the near future, a much higher level of integration would become possible through the use of systems providing a common programming interface to system resources.

Some aspects of Huggett's future scenario are already with us. Users of Microsoft Windows or of Macintoshes will appreciate the benefits, long known by those with access to more powerful workstations, of a consistent user interface and the ease with which data can be transferred between some programs. Simple application building tools are available for these environments that free the programmer from the need to duplicate effort coding many aspects of the low-level operation of the user interface. Provided that they do not impose too many restrictions on the software developer, these tools may help to provide at least one part of a suitable environment for integration. Unfortunately, most are only capable of dealing effectively with small volumes of data with a relatively simple structure. Although useful for building small applications and prototypes, these tools are restricted by the absence of suitable data management facilities essential to large scale systems.

Most DBMS also provide some form of application building tool, usually known as a fourth generation language or 4GL. Many work only with character based displays, but others such as the Ingres Windows/4GL which runs under X provide full access to the facilities of graphical window management systems. Unfortunately, most current DBMS have deficiencies in other areas that reduce their suitability for use in integrated systems. Although not widely realised, it is possible to store almost any information, including text, vector and raster images, in a conventional relational database. Storage, manipulation and retrieval of this data is not difficult, but it must all be done within application programs. The database is only used as a safe method of storage and any manipulation such as text or diagram editing can only be performed by specially written programs, or by exporting the data to another package.

With the exception of some integrity constraints used for validation, almost all knowledge of the behaviour of the data must be incorporated in applications or in external programs. The fundamental data management functions intended to maintain the security and integrity of data, which are provided for simple tabular data by a DBMS, must be performed manually or by purpose written programs. This has serious implications for software maintenance in large scale projects, and this knowledge will not be available to any users employing *ad hoc* SQL queries. In particular there will be no possibility of accessing graphical information stored in the database, whether raster or vector, via the query language as the all important display functions will be part of external programs rather than built into the DBMS. This in turn can only add to the software development and testing problems.

The scale of the problem will, of course, depend largely on the scale and complexity of the system and may not always be apparent to those whose experience is limited to smaller single-user systems. It is easy to think that one can develop the procedures and maintain the discipline necessary to ensure the integrity and consistency of the separate systems in a multi-user environment, but the illusion soon fades when the context numbers on the plans do not match those in the context records, or the pottery from the foundation trench of a roman wall turns out to be a seventeenth century salt-glazed stoneware.

Although many errors can be traced to mistakes in data entry, a significant proportion of these could be avoided by careful use of validation controls and integrity constraints. In an excavation database it should not be possible to assign finds to a context or to insert a stratigraphic relationship if the related context record does not exist. Indeed, the full semantics of such data extend beyond this simple example of referential integrity to more complex behavioural rules. If a context is recorded as a cut then clearly it cannot contain finds or have a soil description and can only participate in a limited set of stratigraphic relationships. A system that does not implement the full set of behavioural rules and constraints appropriate to its data cannot hope to prevent users from entering incorrect data that may compromise the integrity and hence utility of the database.

Many DBMS provide only rudimentary facilities to support these requirements with the result that rules and constraints must be built into the application programs. As a system grows in complexity and more applications are written to access the database, the need to reproduce the checking routines throughout the system will inevitably lead to maintenance problems. If a new check is added or an existing one modified, the changes must be propagated to other programs. Even with a disciplined approach to software development and the

help of suitable software engineering tools, keeping track of such changes can become a severe burden.

What is needed is a system that not only provides a consistent user interface for all applications, but also one that provides a consistent means of data storage and access. One major step in this direction can be made by expanding the concept of data to include behavioural rules or 'expert knowledge' about the problem domain. If these rules are stored in the database they can be invoked automatically whenever the data is accessed (Stonebraker *et al.* 1987; Stonebraker 1989). More importantly from the software development and maintenance viewpoint, they are stored only once and do not need to be reproduced in each program that accesses the data.

Another means of implementing behaviour in the database, rather than in the applications, is to extend the variety of data types that the DBMS is able to manipulate. In this way, the fundamental validation checks and the range of legal operations on a data type become part of its definition in the database. Additional general purpose data types such as bitmaps and free text have been added to several DBMS, but a far more significant development is the provision of facilities to support 'user-defined' types. The remainder of this paper is devoted to a discussion and example of the use of a 'user-defined' Abstract Data Type (ADT) mechanism in a context that should be familiar to most archaeologists.

1.3 Adding new data types

Most conventional DBMS provide a minimal range of basic data types; character strings, integer and real numbers. Many also have special data types for storing other commonly used items such as money, dates and time. Usually these additional types are accompanied by at least rudimentary arithmetic and ordering operators. With these it is possible to determine the earlier or later of two dates, the difference in time between two dates, or to add or subtract a period of time to or from a date. Equally important is the ability to create indexes on a date column and to use these to enhance query performance. The range of valid dates varies between packages. Some artificially limit the range by an origin located in the very recent past. Others provide a more generalised model of time, with valid dates covering the range of the Gregorian or, in some cases, the Julian calendar. Unfortunately these dates must be specified precisely and there is no provision for a fuzzy date object suitable for many historical purposes let alone something that could deal adequately with the vagueness of archaeological time (for a discussion of a fuzzy date ADT for historical purposes, see Ryan forthcoming).

Date and time are examples of abstract data types. In addition to these built-in types the next generation of DBMS will provide mechanisms for adding user-defined ADTs. Indeed the objects in an object oriented system are ADTs, and the latest release of the Ingres relational DBMS also provides this facility as an optional extension to the basic product. The definition of an ADT or object consists typically of a data structure used to store values, procedures for translating between the internal, machine oriented, stored form and an external, human oriented, representation, and definitions of the operators and functions that may be applied to the object. In addition several functions used internally by the DBMS must also be provided. This is usually done by writing the necessary 3GL code (in C in the case of Ingres, in C++ or another OO language in most OO systems) to implement the translation, operators and other functions.

The implementation of ADTs is not a trivial exercise and requires rigorous design and testing before finally linking with the DBMS. The effort is often greater than would be necessary if the manipulation of basic data types was performed in a single application, but a number of benefits offset this initial effort. Once installed the new data type is available for use in both queries and application programs in just the same way as the built-in types. To the user it becomes just another built-in type. Because it is stored in the database it can be re-used whenever it is needed. Application programs are greatly simplified and the programmer no longer has to worry about incorporating all necessary integrity checks and behavioural rules into each program.

Some of the major benefits of ADTs in an archaeological information system would come from the definition of suitable text and graphics types to handle the wide variety of sources and products of the excavation process. In the case of textual material this could provide a means of storing and searching transcripts of documentary sources in much the same way as one might use a free text retrieval system. However unlike the retrieval system it would also be possible to use such types for text production in combination with other data held in the database. This might include the possibility of several people being able to work on and refer to parts of a site report. Text could of course be linked to other material in the database either to ensure automatic update of internal references or to provide a combination of the features of both hypertext and database environments.

Graphical ADTs would open the possibility of manipulating raster and vector images. Many of the operations performed now within GIS, image processing software and CAD systems might be added as required. A system for storing and manipulating encoded representations of artefact shapes was proposed by Goodsen in a paper at the 1989 CAA conference (Goodsen 1989). This involved the use of both object oriented and relational DBMS but with the facilities described here there would be no need to split shape representation and structured description between two databases.

The implementation of ADTs for these applications would be a substantial task, and even an outline description would not be possible here. As a far simpler example, consider a point in 2D space such as an Ordnance Survey National Grid Reference. The conventional representation is a character string consisting of two alphabetic characters indicating a 100km square, followed by an even number of digits providing an East/North coordinate pair within the

100km square. For example, the centre of Canterbury is located near TR149578. That is 14.9km east and 57.8km north of the origin of square TR, itself 600km east and 100km north of the National Grid origin. This conventional form also contains information about the measurement precision of the specified point. In this example, a 'six-figure' reference, the point is located to the nearest 100m.

For arithmetic purposes it is more convenient to use a purely numerical coordinate system in which the centre of Canterbury would be specified as 61491578, but this form is generally less familiar and less intelligible to human readers. However it does provide a suitable basis for an internal representation for the ADT. The translation functions would be designed to convert between the external character string representation, and an internal numeric form.

In this way the user would see grid references displayed in the familiar form, but the internal representation would be more appropriate for use as two dimensional spatial coordinates. Thus we might define the following data structure to hold the absolute numeric coordinates in units of, say, centimetres, together with an indication of the precision of the original grid reference. In conventional C this might be expressed as:

```
typedef struct {
        long    east;
        long    north;
        long    precision;
} NGR;
```

The input translation function would generate the internal representation by removing the grid letters and substituting equivalent numeric values suitable for arithmetic manipulation, and recording the unit of precision. It would include a check for legal grid square letters and for valid numeric values. Any illegal values would be rejected:

```
NGR *NGR_in ( char *ext_str );
```

Thus the statement:

```
NGR_in ( "TR149578" );
```

would result in the stored values:

```
east        = 61490000
north       = 15780000
precision   = 10000
```

The output translation function would remove the numeric representation of the 100km square and restore the grid letters, taking care to print the result to the specified precision:

```
char * NGR_out ( NGR p );
```

If the database included coverage of sites at a European, or even world scale, it would be preferable to adopt an internal representation that used geographical coordinates rather than the local form used in this example. The input and output functions would then involve a transformation between national grid references and this internal form. Further functions could be added to enable transformation between the internal form and one suited to any required map projection. (For a discussion of these transformations and their implications for archaeological databases, together with some example functions in FORTRAN, see Scollar 1989).

So far we have seen how a new data type might be used to store and retrieve data in a convenient form. However, the real power of ADTs is only realised when a set of suitable operations is also defined. For example, in addition to the simple external textual representation, it might also be useful to provide functions to give access to the numeric values of the coordinate pair:

```
long        NGR_E ( NGR p );
long        NGR_N ( NGR p );
```

Valid binary operators that might be applied to NGR objects would include a test of equality:

```
boolean     NGR_equal ( NGR p1, NGR p2 );
```

and the distance between two points (again in cm) could be evaluated using a function:

```
long        NGR_distance ( NGR p1, NGR p2 );
```

Once the code to implement these and any other required functions has been written, tested and installed, the grid reference ADT could be used as follows. Given a relation of the form:

```
SITE ( Site_Name, ngr, ...)
```

in which ngr is a column of type NGR, the following query, expressed here in a hypothetical extended SQL, could be used to retrieve the names of all sites within a distance of 10km of the specified point:

```
SELECT site_name
FROM site
WHERE NGR_distance ( 'TR150580', ngr ) < 1000000
```

The ability to deal with spatial data might be extended by specifying grid square, line and polygon data types. Operations and functions on these would include tests for overlap, calculation and comparison of area, and tests to determine if a point is inside a polygon or grid square. For example, a grid square type could be constructed from the coordinates of its south-west and north-east corners:

```
GRID_SQUARE *grid_square( char *SE, char *NW );
```

And a function to test whether a point was inside the square could be provided as follows:

```
boolean     inside ( NGR *ngr, GRID_SQUARE *gs );
```

It might be argued that all this is possible using standard SQL with its simple character and integer columns. For example, given the relation

```
SITE ( site_name, grid_letters, easting, northing, ...)
```

the query:

```
SELECT site_name
FROM site
WHERE grid_letters = 'TR' AND
      easting >= 1000000 AND
      easting < 2000000 AND
      northing >= 5000000 AND
      northing < 6000000
```

would return all sites within the specified 10km grid square. An ADT version:

```
SELECT site_name
FROM site
WHERE inside ( ngr, grid_square ( 'TR1050, TR2060' ))
```

is easier to write but what other benefits does it provide?

Firstly, the importance of an improvement in the expressive power of the query language should not be dismissed so quickly; consider, for example, the conventional SQL equivalent of the earlier query using the distance function. Alternatively, note that the conventional SQL query to find sites within a grid square becomes significantly more complex if the required search area covers two or more 100km squares, whereas the ADT version is unchanged.

Secondly, in the present example the selection condition has been reduced to a single function called by the query executor. This can improve performance as the database server needs to apply only one test to each record rather than the previous five. The same functions are available to both the query language user and to the programmer producing 3GL and 4GL applications. The implementation is stored once only, at the point where it can be used most efficiently. As a result, the application programs are greatly simplified and are much easier to maintain.

A further benefit would come from a system that provided additional access methods suited to this form of data. Conventional linear indexing methods are poorly suited to spatial data. The system might be able to make use of indexes in processing the above SQL query, but any attempt to process queries involving overlapping areas is likely to completely defeat any indexing or query optimisation strategy available in current DBMS. This lack of suitable indexing and access methods is one of the major reasons why some GIS employ a special purpose DBMS for data storage, one that is optimised for handling spatial data.

The POSTGRES experimental DBMS (Stonebraker & Rowe 1986), from which some of the new techniques used in the commercial Ingres product are derived, will eventually allow the addition of specialised access methods to improve performance in cases that are poorly served by linear indexing methods. In the current release, an Rtree access method is provided as an example. This is a storage strategy based on the minimum bounding boxes of objects and is ideally suited to two-dimensional spatial data (Guttman 1984; see also Beckmann *et al.* 1990 for a recent discussion of variations on this method).

Returning to the topic of the excavation database, it is clear that there are many areas in which ADTs can help to improve both the functionality and performance of large scale information systems. The spatial types discussed above can be extended to include the third dimension. Suitable display functions for both vector and raster graphic data types can be included as part of their ADT definition. When combined with a user interface that permitted graphical interaction, many aspects of the functionality of current GIS could be incorporated into a DBMS.

Plans, photographs and other graphical material together with appropriate display functions would permit much closer control of the wide variety of complex data to be handled in a coherent manner. Instead of relying on error-prone human input to record a list of composite plan and photograph numbers for every context, the task of finding illustrations could be largely automated. Such a query might ask for all drawings covering an area of the site within five meters of the centre of a context. If the bounding volume of a photograph is recorded it would be possible to request all photographs including a chosen point, and taken within so many days of the context excavation date. Selective display of parts of an image and the ability to overlay images would contribute to the construction of composite plans from context outlines in direct response to database queries.

The next stage would involve the development of CAD and text processing packages that were able to use the DBMS rather than their own private file formats as their primary data store. Only in this way would it be possible to ensure complete data integrity between data items manipulated by the functionally different front-end packages. Of course, archaeologists should not be expected to write their own packages for such purposes. Ensuring that the DBMS provides adequate performance to support industrial CAD systems is a prominent area of current database research. Implementation of many of these ideas can be expected within the next few years. As an interim solution it may be possible to use packages that provide their own macro programming languages to implement one end of a link with the DBMS.

1.4 Conclusions

In this short paper it has only been possible to deal with one important aspect of recent database developments. As has already been stressed, the development of ADTs is not a trivial task, but in a well designed database much of this work would be done in the application programs. Transferring the responsibility for the management of data types from these programs to the DBMS simplifies the tasks of writing and maintaining programs, but also brings many other benefits in terms of efficiency and in extensions to the query language. Clearly it is not a task to be undertaken lightly, nor is it suited to small scale database projects with a limited lifespan. However, for large scale systems such as an archaeological information system that may be used to record and process the material from many sites over a number of years the effort can be justified in terms of the potential saving of maintenance problems, improved query performance and the extended applicability of the query language.

The next generation of DBMS will include many other features that may prove equally applicable to the problems of archaeological data management. I intend to deal with three of these, inheritance, arrays and recursive queries in a later paper, but in concluding this paper I will return to the question posed in the

introduction of whether we are ready to begin building large scale information systems to support major excavation and survey activities.

It is certainly possible to commence the design of such systems, but implementation will depend on the availability of suitable DBMS. Extensible relational systems have the benefit that they can be developed as extensions to already well-proven products. The inclusion of POSTGRES-derived techniques in Ingres and a stated commitment to continue this process is probably the most prominent example. Similar features can be expected to appear rapidly in competing products, some of which already offer features not found in Ingres. It should be possible to make use of many of these features as they become available, but to be able to do so will depend on appropriate design decisions in the early stages of system development. One of the most important of these decisions will be not to waste time on developing large-scale software that may be redundant before it has even been implemented. This is not to suggest that all recording activity should cease until better tools are available. Archaeologists must continue to work on current projects with whatever is available, but those who are initiating large-scale projects need to be aware of the limitations of current tools and of the potential of recent developments.

Several object oriented systems are also available, but in many cases these are limited to a navigational style of access. In other words, they are more suited to supporting applications than report generation and *ad hoc* queries. Some regard this absence of a powerful set-oriented query language as a step backwards to the limitations of the CODASYL network DBMS model. Others are concerned by the absence of a firm theoretical basis. However, these systems are still in their infancy and we can expect rapid development and a positive response to these criticisms. For this reason I believe that it would be unwise to make an expensive commitment to a single product at this stage. However, there is much to be commended in the object oriented approach and irrespective of the use of an OODBMS, the emerging techniques of object oriented analysis and design (see for example Booch 1991) hold considerable promise and significant advantages over current methodologies.

Whether the object oriented DBMS eventually becomes the dominant form, it is clear that there is considerable life left in the relational systems. Indeed they are now only beginning to achieve in practice many of the promises that have always been inherent in the relational model. Both object oriented systems and the new generation of extensible relational systems offer means of managing an open ended range of data types. As a result they both have the potential to provide a uniform data management system capable of dealing with the variety and complexity of archaeological data.

References

BECKMANN, N., H-P. KRIEGEL, R. SCHNEIDER & B. SEEGER 1990. "The R*-tree: an efficient and robust access method for points and rectangles", *Proc. ACM SIGMOD International Conference on Management of Data*: 322–331.

BOAST, R. & D. CHAPMAN 1991. "SQL and hypertext generation of stratigraphic adjacency matrices", in K. Lockyear & S. Rahtz (eds.), *Computer Applications and Quantitative Methods in Archaeology 1990*, British Archaeological Reports (International Series) **565**, Oxford, Tempus Reparatum: 43–51.

BOAST, R. & Z. TOMLINSON 1990. "Computers in the city: an archaeological information system", *Antiquity*, **64**: 662–666.

BOOCH, G. 1991. *Object Oriented Design with applications*, Redwood, California, Benjamin/Cummings.

CBA/DOE 1982. *The Publication of Archaeological Excavations*, London, Council for British Archaeology.

DADARAM, P. & V. LINNEMAN 1989. "Advanced Information Management (AIM): advanced database technology for integrated applications", *IBM Systems Journal*, **28**(4): 661–681.

GOODSEN, K. 1989. "Shape information in an artefact database", in S. Rahtz & J. Richards (eds.): 349–361.

GUTTMAN, A. 1984. "R-trees: A dynamic index structure for spatial searching", *Proc. ACM SIGMOD International Conference on Management of Data*: 47–57.

HUGGETT, J. 1989. "The development of an integrated archaeological software system", in S. Rahtz & J. Richards (eds.): 287–293.

KIM, K. 1990. *An Introduction to Object-Oriented Database Systems*, Cambridge, Massachusetts, MIT Press.

MCVICAR, J. 1986. "Using microcomputers in archaeology: some comments and suggestions", in *Computer Applications in Archaeology 1985*, London, Institute of Archaeology:102–108.

RAHTZ, S. & J. RICHARDS, (eds.) 1989. *Computer Applications and Quantitative Methods in Archaeology 1989*, British Archaeological Reports (International Series) **548**, Oxford, British Archaeological Reports.

RYAN, N. forthcoming. "Dealing with time and uncertainty in historical databases", *Proc. of the 5th International Conference of the Association for History and Computing*, Montpelier, September 1990.

SCOLLAR, I. 1989. "Geodetic and cartographic problems in archaeological databases", in S. Rahtz & J. Richards (eds.): 251–269.

STONEBRAKER, M. 1989. "Future trends in database systems", *IEEE Trans. on Knowledge and Data Engineering*, **1**(1), March.

STONEBRAKER, M. & L. ROWE 1986. "The design of POSTGRES", *Proc. ACM SIGMOD International Conference on Management of Data*.

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