Computers & Geosciences 31 (2005) 780–785

COMPUTERS & GEOSCIENCES

www.elsevier.com/locate/cageo

# The man who wasn't there: The problem of partially missing data

Stephen Henley*

*Resources Computing International Ltd, 185 Starkholmes Road, Matlock DE4 5JA, UK*

Received 8 September 2004; received in revised form 13 January 2005; accepted 13 January 2005

**Abstract**

Existing commercial database management systems offer little or no functionality to handle the complexity of geoscience data—and other environmental science data—particularly in respect of missing and partially missing (incomplete or imprecise) data items. The emphasis of both the relational theorists (Codd, Date, and others) and the developers of database systems is on commercial applications where only rudimentary treatment of missing data is required, in the form of NULLs, and even these are not handled properly by the SQL language.
© 2005 Elsevier Ltd. All rights reserved.

*Keywords:* Database; RDBMS; Missing data; Null; SQL; Logic; Relational; Fuzzy logic

*Yesterday upon the stair,*
*I met a man who wasn't there*
*He wasn't there again today:*
*I wish that man would go away. – Children's nonsense rhyme*

## 1. Introduction

Although one of the earliest relational database management systems (G-EXEC—Jeffery and Gill, 1976a–c) was developed in the 1970s to support applications in the geosciences, in recent years there has been progressively more reliance on general-purpose relational systems developed for 'business' users. This has the unfortunate consequence that little or no thought has been given to the complexities of managing real scientific data, and the resulting mismatch causes problems which have rarely been recognised despite the potentially severe consequences for the integrity of scientific databases.

## 2. Database management systems and data models

The closest that many geoscientists come (or want to come) to database management systems (DBMS) is the Microsoft Access that comes bundled with the Office suite, or a packaged ODBC-compliant system sitting underneath an applications software product. Yet effective management of their geological data is vital for all exploration and mining projects.

From the 1970s onwards, database management has been and remains an intensely fought-over battlefield. The original protagonists were hierarchical and network DBMSs following international CODASYL standards, and relational systems following (more or less) the principles first articulated by Codd (1970). During the 1980s the relational systems came to dominate the marketplace, largely by default as the older

*Tel.: +44 629 581454; fax: +44 1629 581471.
*E-mail address:* stephen.henley@btconnect.com.

COBOL-based hierarchical systems became obsolete with the mainframe computers which hosted them.

The most widely used database management systems, such as Oracle, Access, mySQL, SQLserver, Paradox, Ingres, and others, are all claimed to be relational. Certainly they all use SQL (Structured Query Language) which itself is often assumed to be an indicator of a relational database system. Unfortunately, SQL itself violates some of the relational principles, and fails to support others, so this is not a good criterion.

## 3. The missing data problem

Every geologist knows the problem: there is an incomplete data set—a gold assay has been missed out by the laboratory, or there is a strange-looking co-ordinate value, a stratigraphic interval cut out by an unconformity, or a rock-type description that has been forgotten. If the data must be stored and processed, something must be done to indicate that these values are missing (whether temporarily or for good). In the bad old (good old) days each application program would have its own way of dealing with missing data and its own requirement for coding it. Quite often the solution consisted of inserting a '-99' or some such value in place of the absent data item. There are two big problems with this type of solution: first, the difficulty of ensuring that the missing-data code could not be confused with a legitimate data value, and second, the certainty that different application programs would require different missing-data codes.

With the development of database management systems, the handling of missing-data codes became more systematic. For example, in G-EXEC (a relational data handling system for geoscience developed in the 1970s) each data file (relational table) contained a data description (sub-schema) in which a missing-data code was defined for each column. This missing-data code was carried with the data wherever it was copied, and was recognised and acted upon by all applications programs within G-EXEC. For new columns created by G-EXEC application programs, a missing-data code was set up that was very unlikely to be a valid data value—the highest negative real number which could be represented in the computer concerned. This was perfectly adequate as long as the data were not transferred to a different computer which allowed a different range of real numbers. A similar, though simplified, approach was adopted in developing Datamine (a relational database/applications system for the mining industry, developed in the 1980s—Henley and Stokes, 1983; Henley, 1992), in that $-1.0 \times 10^{30}$ was adopted as a universal numeric missing-data value, while a blank string was used for a character data null value.

In commercial database management systems, the question of nulls (shorthand for missing data values) became a central issue. In most early systems, and in many database systems to the present day, a hard-coded null-value solution was adopted—most commonly an empty character string. As Codd, the originator of the relational data model, pointed out, however, any column in any table is drawn from a 'domain' or extended data type which could be defined to include any ranges of values—including any special value which is chosen to be the 'null'—and so no 'null' representation can be assumed not to be identical with some real data value. Although his arguments were made in the context of the relational model, they indeed apply equally to any other type of database management system.

Conventional logic allows for just two truth values, *true* and *false*, leaving no room for uncertainty and no provision for missing information. It was recognised very early that this was inadequate for database management systems, and the 'null' concept was introduced. SQL provides a three-valued logic (3VL) solution with most logical operations involving nulls leading to a new logic value *unknown*. Unfortunately, as Date (1995, Chapter 9) demonstrates, standard SQL offers incomplete support even for this simple 3VL model and as a result can lead to serious database integrity problems. His preferred solution is to use only 2VL and a 'default value' approach: the database designer or application developer is responsible for defining one or more special values in each column, each with a set of operations that are allowed (and which must be coded by the designer or developer).

Codd (1990, pp. 203–204) argues convincingly that it is an abdication of the responsibility of the database management system to maintain integrity, if such a crucial role is left to the user or the application to define on a case-by-case basis. One solution to this problem, which he proposed, lies in attaching an extra one-byte column to each data column. This extra column would contain a flag or 'mark' for each missing data value in the column, and the data item in the column would simply be ignored whenever a mark was encountered. This is the method which was adopted in DB2 and some other IBM database management systems. It has the merit that it guarantees there cannot be any confusion between nulls and any legitimate values.

If nulls are used, then however they are coded, both Codd and Date assume that they imply a system of three-valued logic: when comparing values (for example in retrieval or table–join operations) there is either a match (True) or a mismatch (False)—or a 'Maybe' (truth value 'unknown') when one of the operands is a missing value. In his second version of the relational database model, Codd (1990) takes the argument one step further. He identifies two kinds of missing data. The ordinary 'missing but applicable' value—which might be supplied later (the missing gold assay)—is simply unknown, while a much stronger form of 'missing and inapplicable' (the stratigraphic interval cut out by an unconformity) is unknowable and can never be supplied. Codd proposed that these be represented by different marks A and I which could then be handled separately (when required) in manipulation and interrogation of the database. This yielded a four-valued logic system (T, F, A, I).

More recent work by Date and Darwen (1998), as mentioned above, retreats from this multi-valued logic and attempts to prescribe the use of nulls altogether, proposing instead 'special values' (which seem to be little different from the 'default values' previously condemned by Codd). These seem to be very much like the G-EXEC missing-data codes mentioned above, and are either valid values drawn from the data type, or are values excluded from a data type which—as a result—no longer includes the full range of representable values. Crucially, such 'special values' place the interpretation and manipulation burden entirely on applications, with the accompanying risks of inconsistencies arising from different treatment in different places. As far as the geologist is concerned, this is a very retrograde step. In fact, Codd saw that extending the 'null' definition to handle partially missing data was going to be necessary, though he did not pursue this idea. In summary: although Date and Darwen's proposal allows multiple special values, it provides no help for the partial-data problem, and the treatment of missing data is clearly not a priority for them. This is a task which they leave to the applications, which, in effect, is a return to the situation in the pre-relational era.

Pascal[1] proposes another solution: partitioning tables into multiple relations in such a way as to eliminate all occurrences of nulls. Although it certainly achieves this objective, it is neither elegant nor easy to implement, and potentially requires re-structuring of the database every time an update is done. None of the existing commercial database management systems provide the infrastructure necessary to automate such a procedure.

What the geologist needs, indeed, is quite the converse of complete elimination of nulls: a wide range of representations of missing and partially missing data. Codd's 'mark' concept could easily be extended to include at least the following (and there are certainly many more):

- present but illegal (e.g. a co-ordinate value found to be bad),
- present but suspect (a co-ordinate value thought to be bad),
- below a threshold (e.g. below detection limit),
- above an upper threshold,
- outside a defined range,
- not within a list of acceptable values,
- undefined as a result of a computation ( e.g. 0/0 or log(-$x$) )

In most of these cases, a combination of a mark and a data value will be needed. The result could be a multi-valued logic with a number of different flavours of 'maybe' which could allow for processing in different ways by applications programs as well as during database manipulation.

Such a semantically 'rich' data structuring might also allow the use of fuzzy data as advocated by Bardossy and Fodor (2004), which may well become important in the future, and might indeed provide a general framework for expressing missing, incomplete, and imprecise numeric data.

Complexities may arise even in apparently simple textual data. For example, in a drillhole log, there is quite often a 'comments' field. There will be entries in this for a few intervals, but for many logged intervals this may be blank. It does not necessarily mean that information for these fields is 'missing'—merely that there is nothing sufficiently interesting about the particular interval to merit a comment. Indeed, where something really is missing there will often be a comment to indicate it, such as 'note to be added later', where the logged interval is so interesting that it requires further analysis. Where there is no comment, the storage representation is often an empty string. Unfortunately this is interpreted by many SQL systems as 'null'—or in other words 'missing'. Genuinely missing information can be of several types, none of which might be interpreted as 'null'—for example, 'note to be added later' as above, or 'lost sample'.

There are further complications relating to the logic used in database operations. Most database management systems are designed for use in the commercial world, where exact equality can be assured—for example, part serial numbers, numbers of parts ordered, prices, and dates. In contrast, with scientific data, the degree of certainty can be very variable. When comparing two data values, the normal numerical comparisons ($=$, $>$, $<$) are too rigid. What is needed is an additional set of operators such as 'approximately equal to' (within a defined tolerance?) and 'closest match to'. There also needs to be some mechanism for indicating the precision or reliability of stored data in a systematic way.

This is not merely a set of academic quibbles. With progressively more geoscientific software systems adopting ODBC standards, and abdicating responsibility for their database management to commercial products such as Oracle or Access, it is essential that the capabilities of those systems meet the requirements of the scientist and the engineer. As both Codd (1990) and Date and Darwen (1998) point out, the SQL language is

[1]The final null in the coffin? Outline of a relational solution to missing data, Practical Databasc Foundations #8, September 2004 http://www.dbdcbunk.com.

grossly deficient even in its handling of commercial database management problems, let alone more complex scientific and engineering data, and even the most advanced and expensive DBMSs built around the use of SQL will suffer from all of its defects.

## 4. Imprecise and incomplete data

'Select all sample data where the gold grade is greater than 10 g/t'. This is a simple request, which can be translated directly into SQL to perform a straightforward retrieval operation on a table within a database. That is the first impression, at least. A little more thought would show, however, that it is not so simple at all. Let us imagine that one of the records that is retrieved contains a gold assay of 11 g/t. This matches the selection criterion, doesn't it? Well, not exactly. A gold assay of 11 g/t might reflect a sample gold grade of 11 g/t. Or the sample gold grade might *really* be 5, 9, 12, or 20 g/t. The real grade is actually unknown. The assay result, whether it is a single determination or an average of several replicate determinations, can never be more than an estimate of the true gold grade of the sample.

This example emphasises that there is a clear distinction to be drawn between the real-world truth (what the actual gold grade is—which we can never know precisely) and the recorded data (which is a representation of our measurements which will generally be imperfect and imprecise. The database management system can only contain the recorded measurements—not the true values—and therefore numerical operations must take into account the imprecision. Exact equality and inequality tests on the recorded numbers are clearly inappropriate.

All commercial SQL database management systems were developed primarily for handling business data, where the precision (if not the accuracy) of numeric data can be relied on and numbers are assumed to be exact. Thus a similar-looking request 'Select all departments where the number of employees is greater than 10' usually presents no problems.

So how can we address the gold assay database problem?

(1) We could ignore the problem, and do a simple SQL database retrieval. This is what existing mining software systems would do if they rely on ODBC database connections. There is no problem in doing this, as long as the user interprets the results correctly: the data set returned contains all samples whose *recorded* gold grade is greater than 10 g/t. It must be accepted that some of the retrieved records represent samples whose *true* gold grade are lower than 10 g/t and some samples whose *true* grade is higher than 10 g/t will not be retrieved.

(2) We can look into the statistics of sampling and assaying error. If we want to be reasonably sure of finding all samples whose real gold grade is likely to be greater than 10 then this is what we must do. The statistics will help us to define some (lower) grade (say 7 g/t) which will allow us to be sure to some degree of confidence (say 90%) that we have retrieved all the samples that we want. Of course we also retrieve a number of samples whose real gold grade is below 10 g/t—but from reported grades, which are all we have, there is no way to tell which is which. Given this lower reported grade, we can substitute it in the SQL statement and again use a standard database SELECT.

This is all very well for a simple retrieval, but what if we want to do something more complex? Perhaps we want to compare the grades in two different grain-size fractions. We could of course do a simple SQL comparison (e.g. 'SELECT * FROM assaytable WHERE Au1 > Au2'). This would work, but of course only on reported values, not on the unknown real grades. If we know something of the sample statistics, we could use statistical methods such as a Student's *t* test, and even if we know very little, we could use non-parametric statistics. But what can we do if one or both of the grades are reported as 'below detection limit' or 'trace'? What would SQL make of such a data value in Au1 or Au2 or both?

Indeed, 'trace' or 'below detection limit' is a classic example of incomplete or partially missing data. There is some information (the grade is known to be very low) but certainly not enough to be able to do numerical comparisons between 'trace' values for different samples ('trace < trace' should evaluate to *unknown*), though some numerical comparisons between trace and actual numeric values certainly are possible ('trace < 10 g/t' evaluates to *true*, at least if they are interpreted purely as reported values), so 'trace' cannot be treated as if it were a 'null'.

So how do we deal with such cases in a database? Conventional logic allows for just two truth values, *true* and *false*, leaving no room for uncertainty and no provision for missing information. It was recognised very early that this was inadequate for database management systems, and the 'null' concept was introduced. SQL provides a three-valued logic (3VL) solution with most logical operations involving nulls leading to a new logic value *unknown*. Unfortunately, as discussed above, Date demonstrated that standard SQL offers incomplete support even for this simple 3VL model and as a result can lead to serious database integrity problems. Date's and Pascal's 'anti-null' stance, as discussed above, and their proposed solutions, are clearly inappropriate, and even Codd's 4VL model is probably inadequate. However, it can be argued that Codd's model might be generalised to an *n*-valued logic model in order to handle the more complex situations of imprecise and partially missing data in fields such as the geosciences. Alternatively, a general solution could well be offered by the use of fuzzy logic (Bardossy and Fodor, 2004) which might also avoid the extreme complexity of *n*-valued logic systems which is feared by Date. At present, however, much of the intelligence needed for managing such data is developed on a case-by-case basis for applications, with results including:

(a) database integrity can be compromised,
(b) database management systems with restricted logical functionality can return incorrect results, and
(c) wheels are re-invented many times, with corresponding waste of effort and risk of errors and inconsistencies.

There is a very large variety of applications software used to process geological data. It is impossible to generalise about how each of these programs treat missing data. Indeed, this is the reason why it is so important that the database management system must provide a valid and consistent means of handling nulls. One important area that remains to be dealt with is the statistical and geostatistical treatment of imprecise and incomplete data. Some work has been done on fuzzy kriging (Bardossy and Fodor, 2004, Chapter 5) but this addresses only a small part of the overall problem.

## 5. Conclusions

Commercial DBMSs that are accessed through SQL are very bad at handling missing data, and have no mechanisms at all for handling imprecise or 'partially missing' data. They use an incomplete 3-valued logic, and a 'null' whose meaning is not properly defined. In the geosciences there is actually a wide range of types of data which can be wholly or partially missing for various reasons, and as a result any data management process that relies on SQL cannot be relied upon to give correct results.

Relational database specialists Codd, Date, and Pascal are all in agreement that SQL is inadequate (indeed incorrect) especially in its treatment of missing data, however they disagree on how best the problem should be solved. All of them argue in the context of business data and fail to consider the more demanding requirements of scientific data.

The differences between Codd and Date in their approaches to the problem of nulls are highlighted in the debate between them (Date, 1995, Chapter 9). Both use a variety of esoteric arguments, but their positions can be summarised simply. Date believes that a relational database should contain no nulls at all. An operation on a data item (say, testing for equality to a given string) should produce just two possible answers—'true' or 'false'—and all logic used should be restricted to these two values. Codd takes a more pragmatic view that there really are situations where the value of a data item is unknown, and further that there are two sorts of 'unknown': missing and applicable and missing but inapplicable. Codd does not dispute that reality (if it could be known fully) contains only 'true' and 'false' statements, but asserts that nevertheless in the real world as modelled by databases it is necessary to use 3- or 4-valued truth tables (though this might not necessarily require multi-valued logic).

Date's preferred solution, the 'default-value' approach, avoids the use of anything identified as a 'null'. However, it merely shifts the burden of dealing with missing information from the database management system to the application or the user. The problem is more acute for observational sciences such as geology than for the commercial world, because there are many more ways in which data may be missing or incomplete, as discussed above. Furthermore, SQL-based database management systems fail to meet the requirements of either Date or Codd, and certainly fail to meet the more complex requirements of geoscience.

Is there some compromise which meets the requirements of both Codd and Date? Indeed, are they even in fundamental disagreement at all? And do we, as practising geoscientists, need to worry ourselves about all this theorising? The answers to these three questions, in order, are 'maybe', 'no', and a resounding 'yes'. It is crucial that we understand how our database engines work, and how to ensure the integrity of the data which we use as our raw material for deposit modelling and mine planning. Without panicking, we need to be aware that the standard ODBC database interface behind our mining software does not give us licence to forget about the database problems. There are ways to deal with the problems. From simpler to more complex:

1. attention to database design—for example, a full understanding and proper use of relational database normalization,
2. work-arounds to handle the various missing-data and incomplete-data problems,
3. development of a more science-friendly database management environment.

(1) is something that anyone should be able to do, and should be required to do, when setting up and operating a database. The database design process should always include full normalisation at least to third normal form, and this ought to eliminate any need for Codd's 'inapplicable' missing values. However, it is possible to take this process of normalisation to extremes where it becomes self-defeating. Pascal's suggestion that tables

should be partitioned to remove all occurrences of nulls could lead to an unworkably complex database.

(2) will certainly require some careful logical thinking (especially when it involves slippery questions such as geometric precision) and may require some applications programming. It may be possible to develop work-arounds to avoid the problems of handling nulls within existing database management systems. This might include the creation and use of special values (default values) as advocated by Date. However, these will require special treatment by applications—or if the user has no access to applications coding, the use of specially written software to convert and structure the data, as an applications pre-processor.

(3) involves the development of a whole new database management environment suitable for observational sciences. It may not have the multi-user and transaction-processing features required by an airline reservations system—but it would handle missing or incomplete data in a consistent and logical way, and should provide a rigorous framework for management of such data. This is the real solution to the problem. Existing database management systems are built on the 'closed world' assumption that the database contains all data relevant to the particular universe of discourse. In observational sciences, an 'open world' model is more appropriate. One of the implications of this is that there must be proper provision for gaps in knowledge—missing data—and one of the implications is that it may be necessary to use a more complex logic system. While Date may be right in stating that *n*-valued logic is too complex, a practical and appropriate alternative may already be available in fuzzy logic.

## References

Bardossy, G., Fodor, J., 2004. Evaluation of Uncertainties and Risks in Geology. Springer, Berlin, 221pp.

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

Codd, E.F., 1990. The Relational Model for Database Management: version 2, Addison-Wesley, Reading, MA 538pp.

Date, C.J., 1995. Relational Database Writings, 1991–1994; Addison-Wesley, Reading, MA 542pp.

Date, C.J., Darwen, H., 1998. Foundation for Object/Relational Databases: The Third Manifesto. Addison-Wesley, Reading, MA 496pp.

Henley, S., 1992. Orebody modelling in a relational database framework. In: Dowd, P.A., Royer, J.J. (Eds.), Second CODATA Conference on Geomathematics and Geostatistics, Sciences de la Terre, Ser. Inf., Nancy, 31, 477–484.

Henley, S., Stokes, W.P.C., 1983. Use of graphics computers in mine planning. In: Surface Mining and Quarrying, Institution of Mining and Metallurgy, London, pp. 323–328.

Jeffery, K.G., Gill, E.M., 1976a. The design philosophy of the G-EXEC system. Computers & Geosciences 2 (3), 345–346.

Jeffery, K.G., Gill, E.M., 1976b. The geological computer. Computers & Geosciences 2 (3), 347–349.

Jeffery, K.G., Gill, E.M., 1976c. The use of G-EXEC for resource analysis. Mathematical Geology 9 (3), 265–272.