The Association for Institutional Research
Number 33 Fall 1989

AIR
for Management Research, Policy Analysis, and Planning

AIR Professional File

Data Integrity: Why Aren't the Data Accurate?*

Frank J. Gose
Director of Student Data Administration
University of Colorado at Boulder

A term may convey different meanings depending on the context in which it is used. Often the use of a term may mean different things to different people within a given conversation. Even familiar terms are sometimes "fuzzy" in the meaning they convey. Date and White (1988) discuss this phenomena with regard to the use of the term "record":

> At different times that single term can mean either a record instance or a record type; a COBOL-style record (which allows repeating groups) or a flat record (which does not); a logical record or a physical record; a stored record or a virtual record; and so on. (p.9)

A fuzzy meaning attaches to the use of the term "data integrity" when it is used with reference to data-base management systems (DBMS). Probably the most common definition concerns integrity of the physical data base as seen by the data-base administrator (DBA). In this context, data integrity may encompass activities like journaling data-base transactions, record definitions, and the relationships between records or tables in a data base, i.e., the physical design of the data base, data-base performance monitoring, access and status checking on the data base, and security.

Data integrity, however, also concerns the error-free (non-corrupted) or error-prone (corrupted) integrity of data. Corrupted data are found in most data bases and may not necessarily be addressed by any of the activities of the DBA. Brathwaite (1985) provides the following definition:

> Integrity is a measure of the quality and reliability of the data on which computer-based information systems depend. Many computerized data bases currently in use suffer from high error rates in data they receive, and consequently, are riddled with bad data. With incorrect data, the most efficient and sophisticated system is useless. (p.8)

Usually, data problems can be identified and an assessment made of the effort needed to correct the problem. If it is important enough, the data may be corrected on the operating data base. Often, however, the data are corrected on a report but not on the data base or file from which the report was generated. Even when corrected on the data base or file, sources of corruption often are not identified or fixed.

Qualifications and Limitations

In this paper, the term "data base" will refer to the place where original, computerized institutional administrative data are kept. Also, the term will refer to copies of data from data bases, such as data found on flat or extract files used to produce operational reports or analyses.

This paper will not address data integrity issues normally associated with the technical maintenance of a DBMS. For example, the lack of adequate journaling, typically monitored by the DBA group, is not addressed. Also, activities such as maintenance and restoration of production and test data bases are excluded.

Also excluded are discussions of willful attempts to

*Paper presented at the Twenty-Ninth Annual AIR Forum, Baltimore, Maryland, April 30 - May 3, 1989.

corrupt or compromise data bases. For example, attempts to illegally change student grades or remove parking violations are not addressed. Additionally, the activities of "hackers" and computer "viruses" are beyond the scope of this paper.

An operational data base, used for administrative applications at a college or university, is presumed. Beyond possible external attempts to corrupt or compromise a data base, and in addition to technical DBMS maintenance problems, users of administrative data continue to be confounded by corrupted data. This paper addresses some sources of data corruption and what might be done to deal with them.

Data Corruption Sources, Identification, and Actions to Resolve

Sources of data corruption include changes in institutional policies, new meaning associated with a datum, user experimentation with the system, purging/consolidation of corrupted data, referential integrity, inadequate analysis and testing of software, running obsolete versions of a program, restructuring set relationships on a data base, and the trade-off between editing and performance. Each of these is described, including the ease or difficulty of identification, and suggested action(s) for prevention are given. Table 1 (page 5) summarizes the sources of data corruption, their identification, and suggested actions to address them.

Changes in Institutional Policies

One of the most subtle ways that data integrity is compromised occurs when institutional policies change but the computer code which accommodates those policies does not. Even the slightest change in policy logic or in the values associated with a datum, if not similarly reflected in the corresponding computer code, will result in corrupted data. For example, if a fee or tuition policy is changed but the respective computer code is not, data integrity is compromised. A policy changed in a process, such as registration activities, if not correspondingly reflected in computer code, may throw a number of data relationships out of synchronization and produce corrupted data.

An easily identified case of this phenomenon occurs when an institution acquires an application system from an external source, either from another institution or from a commercial vendor. With such an acquisition, service requests from users accumulate quickly because the system does not process properly from the users' points of view. In fact, the application code presents policies different from those in place at the institution. When computer code is changed to accommodate a different process, institutional policy is being rewritten. Clearly, much institutional policy is actually written by programmers and employees in the user offices.

Identification. Generally the impact of this kind of corruption is easy to spot when affected data are accessed, either through operational reporting or regular operational processing. Affected users are usually the first to notice these problems.

Suggested Action. Precluding these problems in a proactive way requires the use of standard information-processing techniques that have been around for a long time: thorough systems analysis and design, and equally thorough testing prior to implementation. Open communication among affected users is necessary throughout the process.

New Meaning Associated with a Datum

Many institutions have well-meaning users, adroit at improvisation. This phenomenon results from a user doing the wrong thing for the right reason, and it may go unnoticed indefinitely. Typically, a user with a pressing need knows that submitting a service request for the change will result in a long delay. The user needs the change immediately. The result is an ad hoc, localized, improvisation that is usually communicated reluctantly and after the fact, when some other office encounters corrupted data. Two types of improvisation may occur.

Type One. The first type might occur with the case of a table of U.S. institutions indexed by unique institutional codes. The data-base system might provide a unique record occurrence for each U.S. institution from which a student submitted transfer work. A separate record occurrence might be provided for the free form entry of information on institutions outside of the U.S. and not found in the table. In the case where a U.S. institution might not be found in the institutional table, an enterprising user might quickly opt to use the free form capability by creating U.S. entry on a record for institutions outside of the U.S. The result is a commingling of U.S. institutions on two different record types for a student: one for U.S. institutions and one for institutions outside of the U.S.

Identification. This kind of corruption is usually easy to spot, when the data are accessed. However, some office other than the one that corrupted the data will report the problem (often an Institutional research office).

Suggested Action. Steps to be taken proactively to preclude this kind of problem are these: First and most critically, open communication, combined with internal education and commitment to data integrity, must be in place. Secondly, data administration and other offices concerned with data integrity must have scanning software in place to monitor key data elements and their maintenance.

Type Two. The second type of improvisation occurs when a particular code, meaning one thing, is appropriated to take on a new, additional meaning. For example, a code might be used to deny enrollment or some other activity as a result of expiration of a deadline, e.g., no previous enrollment for some number of preceding semesters. When the system encounters this code in a student's attempt to register, the registration will be blocked until the student reapplies or eliminates the reason for the block. Using an established code to block enrollment for a different reason, such as another kind of hold, will corrupt the data. Only the initiating user knows.

Identification. This kind of data corruption is hard to spot and usually will not be reported by the office responsible for it. More likely, the problem will come to light in incidental conversation, long after many data have been corrupted.

Suggested Action. Data integrity scans are of little direct use. After the fact and indirectly, they may be of use by analyzing relationships between fields. Open communication combined with education and commitment to data integrity must be in place in order to address this problem.

User Experimentation with the System

This source of corruption also comes in two forms: entry of undefined data and of fictitious data for testing.

In the first form, a user may try to get the system to take a datum that has not been legitimately defined to it. This is a form of experimentation that results when it is presumed that service requests will be met with long delays. The vehicle for this action is poor editing capability in the application software. This kind of corruption may go unnoticed for a while, but usually not for long. Data, clearly out of line, show up readily on reports or screens. Often, the corrupted data may cause abnormal termination of computer jobs. The result is both good and bad. It is good because the corrupted data are identified. It is bad because of delays required to correct the data.

Identification. This form of corruption is usually easy to spot, but it must be spotted by offices other than the one reponsible for it.

Suggested Action. Internal education regarding data integrity concepts and open communication can preclude or minimize the problem. Data integrity scans may serve as an early warning system.

In the second version of this phenomenon (entry of fictitious data), a user may enter data that look normal but are fictitious. This is a form of trickery, frequently resulting from the need to do real testing on the system. The best testing is often presumed to take place in the "production environment." Maintaining a legitimate "test environment" can be difficult and time consuming. The consequence of testing in a production environment, however, can be disastrous. While attempts to "flag" fictitious data are reasonable in order that the data will not be reported, all software programs that access the data must be modified or "hooked" to check for the flags.

This process, itself, consumes additional resources, presumes excellent documentation and communication, and unnecessarily complicates code. Additional bugs may be incorporated in the application code. Invariably, some software will not be modified. Subsequently, also, the expected removal of fictitious data may not be complete. The result is corrupted data that remain on the data base and unnecessarily complicated application code. Future processing that accesses these data may further compound the matter.

Identification. This form of corruption may be easy to spot or hard to spot. Again, the office responsible for it will usually not report it as corrupted data.

Suggested Action. A broad-based institutional commitment to data integrity is a "must" before this problem can be successfully addressed. A true commitment to testing in a test environment is necessary. Resources must be committed to establishing and maintaining a reasonable test environment. Data integrity scans may be of help in order to identify corrupted data elements, but usually such scans will need to analyze relationships between fields, rather than analyze a corrupted field directly.

Purging/Consolidation of Corrupted Data

Purging and/or consolidation of data are processes frequently used in a computing environment to meet many reasonable needs: restoration of much needed disk space, improved efficiencies in processing, simplification of the data base, etc. These processes may also represent means for dealing with unwanted or corrupted data.

Purging takes place when data are removed from the system; usually this means from on-line and real-time access. A commonly accepted procedure is to perform some kind of backup of the system or data prior to the purge. Thus, there should be an historical copy of the data from which a future restoration could take place. The expectation, however, is that the need for restoration is remote.

The problem with purging, however, is that invariably the purged data will need to be accessed in the future. The problem of corrupted data is not solved, only delayed. With the passage of time, and usually poor documentation, corrupted data are more difficult to understand and explain. As always, time is of the essence, and what was previously out of sight and out of mind may now come back with a vengeance to haunt the researcher. The process of purging, too, can post its own problems and will be discussed later in this paper.

Consolidation of data takes place when data are regrouped in different formats (different records or different tables) and possibly relocated elsewhere on the data base. The purpose is to recapture available disk space and improve computer resource utilization when the consolidation data or other data are accessed.

As with purging, the problems of corrupted data are not addressed. A compounding factor in this process is that the consolidation logic itself may introduce further data corruption problems. Relationships between data elements may be confounded, and the actual movement of data may not be totally accurate. Thus, while the original problems of data integrity are not addressed, they may be worsened in this process.

The computer department and many users may work in concert to support purging as a means of dealing with data integrity problems. The problem with this approach, however, is that integrity problems remain; they are only hidden.

Identification. These purging and consolidation problems will be easy to spot when the data are restored and accessed; however, the compounding factors of time and poor documentation may make understanding of the restored data additionally difficult.

Suggested Action. Data elements (names, formats, and contents) must be documented, even with corrupted data. Data must have as high a level of integrity as is economically, operationally, and technically feasible. Purging, which serves many useful purposes, must not be used as a tool to address data corruption problems.

Referential Integrity

Referential integrity is a concept which has received increased notoriety with the advent of the mainframe relational data-base management system, DB2. Universities and colleges are faced with problems like the following: Unique combinations of curriculum codes, department codes, and student majors are established. That is, for a given student major, only certain department codes and curriculum codes are applicable for enrollment, and vice versa. Also, some of these combinations may be applicable only for certain periods of time as new combinations are approved and old ones are discontinued.

The following situation is not uncommon. An internal table of allowable curriculum, department, and major combinations exists. Individual students have unique, respective codes associated with their data. At some point, a change is made to the internal table, possibly a deletion or redefinition of what had been an allowable combination. Affected students' data, however, are not

correspondingly changed. The next time the record of any of these students is accessed and the data checked against the table, such as at registration, the student is identified as having an invalid curriculum, department, and major combination. Frustration and delay for both the student and the employee often result. Data corrupted in this fashion represent time bombs waiting to detonate.

Some DBMS like DB2 claim to be able to handle referential integrity within the data base itself (Date & White, 1988). That is, the programmer does not have to program referential integrity constraints into the application code. If referential integrity constraints have been properly described to the DBMS, an update to the table would cause corresponding modifications to affected student data, or other options would be presented to the person trying to do the update. Date and White (pp. 133 & 441) describe this situation as maintaining consistency within foreign keys on the data base.

Identification. This kind of problem may be hard to spot until it makes itself known. Data integrity scans of critical elements may identify some of these problems before they create real havoc on the system.

Suggested Action. Referential integrity constraints must be incorporated in the system, either in the application code or, preferably, in the DBMS itself.

Inadequate Analysis and Testing of Software Maintenance and Enhancement

This is the age-old problem (Brooks, 1982) of inadequate systems analysis before enhancements or modifications are made to the system. The problem is compounded by equally inadequate testing and debugging prior to implementation in production. Limited or poor communication with affected users during analysis as well as during testing only worsens the problem.

This problem is apparent by the large queue of pending service requests at most institutions. Most people associated with administrative computing have become anesthetized to this phenomenon. Martin and McClure (1983), however, offer a striking word of warning. They claim that "Maintenance dominates the software life cycle in terms of effort and cost" (pg. 7). Further, "A fundamental problem with software maintenance is that when a change is made it often introduces unforeseen side effects. Fixing a bug has a great chance of introducing a new bug" (pg. vii). They continue:

> As flaw-fixing introduces new flaws, more and more time is spent on fixing these secondary problems rather than on correcting the structure that caused the original problem. The system steadily becomes less and less well ordered. Some complex systems reach a point where maintainers cease to gain ground. Each new fix introduces new problems. The system has become too unstable to be a base for progress. (p.8)

Martin and McClure further refine the maintenance category:

> . . . most maintenance work falls into the category of perfective maintenance. This is somewhat of a revelation to the data-processing community, challenging the common belief that most maintenance effort is expended in a "fire-fighting" mode reacting to emergency repairs. In reality, most maintenance work can be anticipated and planned much like new development activities. Furthermore, user enhancements dominate the perfective maintenance category. (p.28)

This phenomenon results in the whole system of service requests becoming bogged down. An apparent lack of responsiveness from the computer department produces a high level of institutional frustration with computing. This problem is extremely difficult to assess at a given institution because of a number of factors.

First, the queue of service requests is frequently not available for general review. The computing department and/or a given user department may not encourage general dissemination. Second, poor communication results. Third, both the computing department and a given user department may wish to minimize the amount and effect of flaw-fixing for fear of negative attention to themselves. The result at many institutions is a growing queue of service requests and corresponding frustration throughout the organization.

Identification. Corrupted data resulting from this phenomenon come in various forms, and, consequently, may be both easy and hard to spot.

Suggested Action. Some of the classic tools of information processing are important to use with this problem: thorough analysis, design, and testing prior to implementation in production. In the increasingly distributed but integrated information system environment, open communication is requisite. Service requests must be thoroughly reviewed and prioritized by an office or offices free of vested interests and representing an institutional point of view. The central institutional administration must back this point of view. Users submitting service requests must, in some way, be accountable for their requests. In most computing environments, all parties must understand that computing is a limited resource and not all service requests can be honored. It is far better to do a few service requests well than to do many poorly.

Running Obsolete Versions of a Program

This phenomenon may include the use of wrong versions of maps, tasks, job streams, other modules, etc. It is usually the result of an error committed by someone in the computing department, although users may enter incorrect parameters for a run prepared for them, or, in some cases, they may do some of their own programming which affects the data base. Irrespective of the incorrect version of code and the responsible party, the impact may range from incidental to disastrous. The results of this activity may go unnoticed indefinitely.

Identification. This form of data corruption may be both easy and hard to spot, depending on the data that are corrupted. Data integrity scans may provide a means of early detection.

Suggested Action. The key to precluding this problem is accurate, thorough documentation—documentation that is used.

Restructuring Set Relationships on a Data Base

Often done in the interests of purging, but sometimes done as a result of data-base enhancements, the impact of this phenomenon usually is immediate and long lasting. Figure 1 shows hypothetical set relationships before and after a purge. A network DBMS structure is presumed. The set relationships before a purge are one-to-many from record A to record B and from record A to record C. A one-to-one relationship is presumed be-

tween occurrences of records C and B. Programs may be written to access occurrences of record B directly from record A or through similar occurrences of record C. Of course, occurrences of record C may be accessed through record A.

If, as a result of a purge or other reason, selected occurrences of record B are deleted without the deletion of corresponding occurrences of record C, a clear integrity problem results. The data base is restructured. A unique occurrence of record B will not exist for each unique occurrence of record C. Additionally, not only are data no longer available on occurrences of B when corresponding occurrences of C remain, but programs, modules, etc., may abnormally terminate if they have not been modified to accommodate the break in set relationships. This phenomenon has a variety of impacts that may be felt indefinitely.

Identification. This form of data corruption is usually easy to spot.

Suggested Action. There are two keys to precluding this kind of problem. First, open communication and thorough documentation regarding the purge must exist prior to the purge. Thorough testing of the purge prior to implementation must also be accomplished. Second, offices responsible for reporting from the data base must be in positions of authority to approve or disapprove the purge.

The Trade-off between Editing and Performance

Many of the aforementioned sources of data corruption are at least peripherally related to the absence of adequate validation code or editing routines. However, the absence of such code stands on its own as a source of data corruption. Programmers frequently do not like to write this kind of code, and arguments are often made by members of the computer department that such code complicates programs, consumes resources, and slows response time. When response time problems exist and the queue of service requests is long, it is easy to make the argument that additional editing capability is not desired.

Interestingly, some users will argue against validation code for the reason that they are unduly restricted in their right to enter the data they want. This is a form of gambling, and a heavy price may be paid in terms of data integrity: validity and reliability. A particularly difficult situation presents itself when a key user is allowed to oppose editing, in concert with the computer department. Another office that may have to access the data base for reporting, such as an institutional research office, will struggle to produce accurate reports.

Identification. This kind of data corruption is usually easy to spot, depending on the data that are corrupted.

Suggested Action. The basic problem with this phenomenon is the historic problem of consistent responsibility and authority. Often, offices that have the authority to decide these issues do not have the responsibility to report from the corrupted data base. Conversely, offices with responsibility to report from the corrupted data base (e.g., institutional research) often do not have the authority to either clean the data or contribute to decisions such as those having to do with editing. Consistency of responsibility and authority must be established. Offices interested in data integrity must have the authority to contribute to this kind of decision.

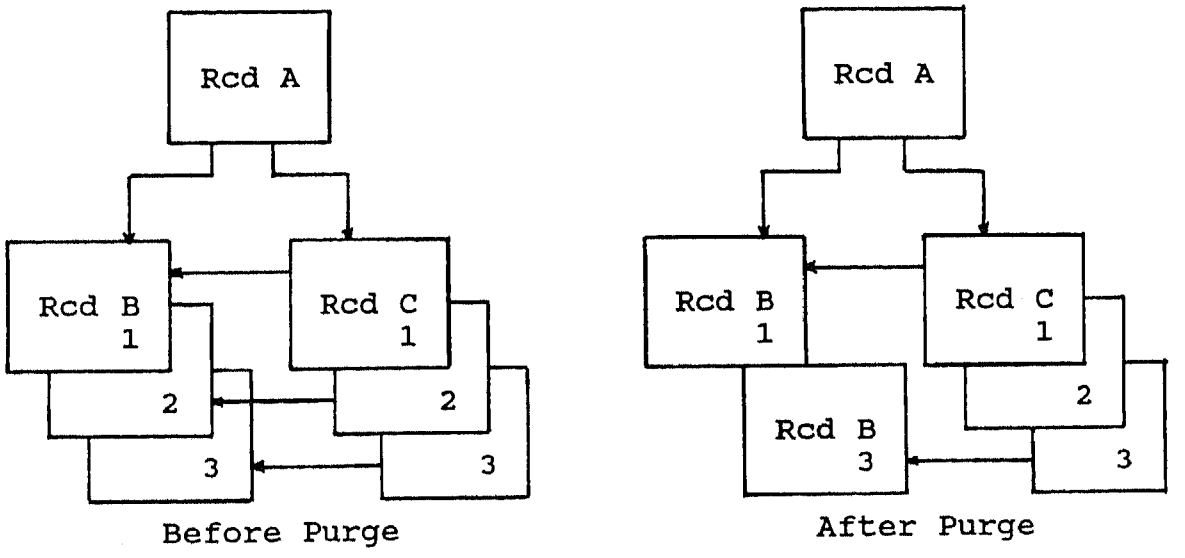

Figure 1. Set relationships before and after purging.

Table 1

Summary of Data Corruption Sources, Identification, and Actions to Address Them

Sources (Abbreviated)	Identification	Suggested Action
Policies	Easy	Analysis, Design & Testing Open Communication
Changed Datum Meaning	Easy	Open Communication Internal Education Data Integrity Commitment Scanning Software
	Hard	Same As Above
User Experimentation	Easy	Internal Education Open Communication Scanning Software
	Easy/Hard	Data Integrity Commitment Commitment To Test Environment Scanning Software
Purging/Consolidation	Easy	Documentation Data Integrity Commitment
Referential Integrity	Hard	Scanning Software Impose Referential Integrity Constraints
Analysis & Testing	Easy/Hard	Analysis, Design & Testing Open Communication Review & Prioritize Service Requests by Third Party Backing by Central Administration User Accountability Do a Few Well
Obsolete Code	Easy/Hard	Scanning Software Documentation
Restructuring	Easy	Open Communication Documentation Testing Responsible Approval
Editing vs. Performance	Easy/Hard	Responsibility & Authority

Technological Tools

The current literature of computing contains frequent references to modern technological tools such as intelligent systems, computer-aided software engineering (CASE) tools, data dictionaries, etc. These tools are powerful and capable of increasing staff productivity. They are, however, no panacea for solving the kinds of data corruption problems discussed in this paper. For example, in the DBMS environment, a data dictionary could be used to keep obsolete code from being executed. A data dictionary also could be used to maintain standard editing routines. However, some computer sites do not operate a DBMS. Some that do may not adequately use their dictionaries. Some do not have data dictionaries. Finally, there is no standard definition of a data dictionary. A data dictionary may be a powerful tool, but it alone will not solve the kinds of problems discussed here.

CASE tools are in an early stage of development. They tend to be specialized in focus and may hold promise for addressing some of the problems we have discussed. Currently, however, with the exception of some capability lodged in the DBMS itself (for example, referential integrity in DB2), there has been minimal impact of these tools on the sources of data corruption.

The human being is the critical link in the successful operation of a computerized system. All of the sources of data corruption discussed here may be found in any computer operation. The current state of technological evolution is not such that tools themselves can solve these problems.

Conclusion

This paper has focused on accuracy and reliability aspects of data integrity and its maintenance. Concerned, involved, and responsible institutional professionals in data administration, institutional research, analytical studies, admissions and records, and other offices will benefit and be challenged by raised awareness of the accuracy and reliability aspects of data integrity.

An underlying theme in this paper is consistency of responsibility and authority. The central institutional administration must value data as a corporate resource, establish a process of relationships that openly declares that stance and commit resources to that effort.

Dr. Kaye Howe, vice chancellor, University of Colorado-Boulder, in a personal communication dated February 15, 1989, presented an analogy to describe the importance of data in an institution's health and effectiveness. She said that data in the institution is like blood in a living organism. If data (blood) are corrupted (diseased), the organization (living organism) is weakened and sick. Some and perhaps all parts of the organization become weakened. The whole organization struggles for health and effectiveness.

Serious commitments to the maintenance of data integrity, throughout the organization, are essential for institutional well-being.

References

Brathwaite, K. S. (1985). *Data administration: Selected topics of data control.* New York: John Wiley & Sons.

Brooks, F. P. (1982). *The mythical man-month: Essays on software engineering.* Massachusetts: Addison-Wesley.

Date, C. J. & White, C. J. (1988). *A guide to DB2 (2nd ed.).* Massachusetts: Addison-Wesley.

Martin, J. & McClure, C. (1983). *Software maintenance: The problem and its solutions.* New Jersey: Prentice-Hall.

For information about back issues of the AIR Professional File:

The Association for Institutional Research
314 Stone Building, Florida State University
Tallahassee, Florida 32306-3038 (904) 644-4470

The *AIR Professional File* is intended as a presentation of papers which synthesize and interpret issues, operations, and research of interest in the field of institutional research. Authors are responsible for material presented. The *File* is published up to four times per year by the Association for Institutional Research.

Editor-in-Chief: John A. Lucas
Director, Planning & Research
William Rainey Harper College
Algonquin & Roselle Roads
Palatine, IL 60067

Managing Editor: Jean C. Chulak
Administrative Director
The Association for
Institutional Research
314 Stone Building
Florida State University
Tallahassee, FL 32306-3038