DBMS
DATABASE & CLIENT/SERVER SOLUTIONS
OCTOBER 1994
VOLUME 7 NUMBER 11

The Relational Model Turns 25

INTERVIEW

$3.95 (Canada $4.95)
10
70989 35564

NEW AND REVIEWED

Capsule, db-UIM/X, and Access Developer's Toolkit

DBMS
DATABASE & CLIENT/SERVER SOLUTIONS

October 1994
Vol. 7 No. 11

Cover Art Direction by Barbara Licata

Cover Illustration by Ron Licata

PRODUCTS

COLUMNS

DEPARTMENTS

INTERVIEW

COMING NEXT MONTH

In-depth looks at GUI-testing tools and data modeling techniques, and enterprise accounting system infrastructure.

DBMS (ISSN 1041-5173) is published monthly, except in June, which is semimonthly and contains the *DBMS* Buyer's Guide Issue, by Miller Freeman Inc., 411 Borel Ave., Suite 100, San Mateo, CA 94402. Second-class postage paid at San Mateo, CA, and additional offices. ***Subscriptions:*** For subscription inquiries, change of address, call: 800-334-8152 (U.S. and Canada), 619-745-2809 (all other countries). Or write *DBMS* at P.O. Box 469039, Escondido, CA 92046-9039; fax 415-905-2233. Please allow 4-6 weeks for an address change to take effect. Single copy price is $3.95 (Canada $4.50). Subscriptions are $24.95 for one year (13 issues); $43.95 for two years (26 issues) in the U.S. and its territories, Canada (GST# R124771239) and Mexico. $35.00 per year. All other foreign: $60.00 per year. Foreign orders must be prepaid in U.S. Dollars on a U.S. bank. ***Postmaster:*** Send address changes to: *DBMS*, P.O. Box 469039, Escondido, CA 92046-9039. ***Foreign Newsstand Distributor:*** Worldwide Media Service Inc., 115 E. 23rd St., New York, NY 10010; 212-420-0586.

The Relational Model Turns 25

...AND WE'RE STILL TRYING TO GET IT RIGHT.

BY DAVID MCGOVERAN

Codd began the 1970 version (see Ref. 1 at the end of this article) of his famous 1969 paper on the relational model (see Ref. 2) with the words: "... users ... must be protected from having to know ... the internal representation [of data] ... Activities of users at terminals and most application programs should remain unaffected when the internal representation ... is changed." This was the key goal of the relational model.

At that time, application code (including control flow structure) was tightly coupled to the implementation of data structures. In addition, there was little uniformity in that physical implementation: The data structures and access methods were often specific to each application. This approach resulted in high maintenance costs, slow development, and error-prone code. Sharing data among applications was difficult and file-oriented. The very concept of database consistency was ill-defined. The relational model promised to solve these problems, as well as provide other business benefits.

Nonetheless, many applications are still suffering from the same old problems. Some of these are even so-called "relational" applications. Although the data-processing world has progressed greatly in the last 25 years, the industry continues to pay for the sins of the past. Legacy and heritage systems still place a burden and a constraint on businesses. In this 25th year since the introduction of the relational model, it seems appropriate to ask a few questions:

- Why are we still fighting these problems?
- What should we expect of an implementation of the relational model?
- Has the relational model delivered, and, if not, why?

In summary, just what is the state of attempts to implement the relational model?

Goals

I am sometimes asked, "Why should anyone care whether RDBMS vendors implement the relational model?" I find this question both amusing and sad. To me, the question is exactly like asking, "Why should anyone care whether automobile manufacturers follow the laws of physics?" The answer to both questions: You want a product that works; one you can understand, one that performs, one that meets

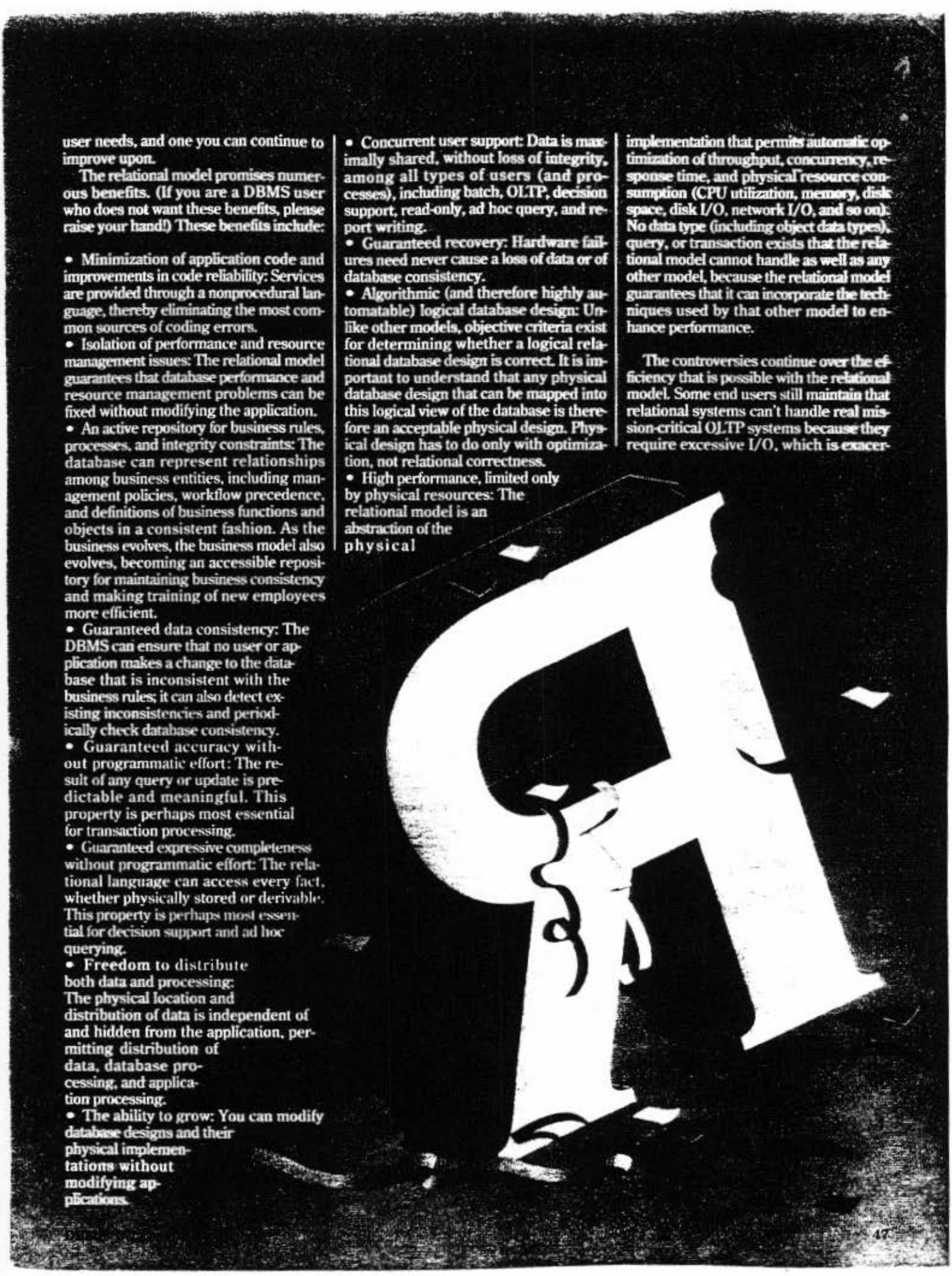
David McGoveran is president of Alternative Technologies (Boulder Creek, California), a relational database consulting firm founded in 1976. He has written numerous technical articles and is the publisher of the "Database Product Evaluation Report Series."

user needs, and one you can continue to improve upon.

The relational model promises numerous benefits. (If you are a DBMS user who does not want these benefits, please raise your hand!) These benefits include:

- Minimization of application code and improvements in code reliability: Services are provided through a nonprocedural language, thereby eliminating the most common sources of coding errors.
- Isolation of performance and resource management issues: The relational model guarantees that database performance and resource management problems can be fixed without modifying the application.
- An active repository for business rules, processes, and integrity constraints: The database can represent relationships among business entities, including management policies, workflow precedence, and definitions of business functions and objects in a consistent fashion. As the business evolves, the business model also evolves, becoming an accessible repository for maintaining business consistency and making training of new employees more efficient.
- Guaranteed data consistency: The DBMS can ensure that no user or application makes a change to the database that is inconsistent with the business rules; it can also detect existing inconsistencies and periodically check database consistency.
- Guaranteed accuracy without programmatic effort: The result of any query or update is predictable and meaningful. This property is perhaps most essential for transaction processing.
- Guaranteed expressive completeness without programmatic effort: The relational language can access every fact, whether physically stored or derivable. This property is perhaps most essential for decision support and ad hoc querying.
- Freedom to distribute both data and processing: The physical location and distribution of data is independent of and hidden from the application, permitting distribution of data, database processing, and application processing.
- The ability to grow: You can modify database designs and their physical implementations without modifying applications.
- Concurrent user support: Data is maximally shared, without loss of integrity, among all types of users (and processes), including batch, OLTP, decision support, read-only, ad hoc query, and report writing.
- Guaranteed recovery: Hardware failures need never cause a loss of data or of database consistency.
- Algorithmic (and therefore highly automatable) logical database design: Unlike other models, objective criteria exist for determining whether a logical relational database design is correct. It is important to understand that any physical database design that can be mapped into this logical view of the database is therefore an acceptable physical design. Physical design has to do only with optimization, not relational correctness.
- High performance, limited only by physical resources: The relational model is an abstraction of the physical implementation that permits automatic optimization of throughput, concurrency, response time, and physical resource consumption (CPU utilization, memory, disk space, disk I/O, network I/O, and so on). No data type (including object data types), query, or transaction exists that the relational model cannot handle as well as any other model, because the relational model guarantees that it can incorporate the techniques used by that other model to enhance performance.

The controversies continue over the efficiency that is possible with the relational model. Some end users still maintain that relational systems can't handle real mission-critical OLTP systems because they require excessive I/O, which is exacer-



bated by normalization. Some popular consultants claim that the data model itself is unsuitable for tasks such as decision support. Object database proponents claim that the relational system can't support anything but simple data types.

RDBMS vendors build great products that violate the model, claiming that their products must address "the real world." If these claims were the only evidence, I would have to conclude that the relational model is not worth much. At the same time, I must stand by the claims I have made previously in favor of the relational model. So, what is wrong? Let's look at what RDBMS vendors have accomplished and how, and try to see if we can discover what, if anything, has gone wrong.

We've Come a Long Way, Maybe

Since the first commercial RDBMS products began to ship in the late 1970s, users have reaped many benefits. Vendors such as Cincom, Digital, IBM, Informix, Hewlett-Packard, Oracle, Sybase, and Tandem should be praised for these achievements, which would not have been possible without the relational model. There are several shining examples of relational technology's success, including the following:

- Relational DBMSs are the de facto "open systems" database solution at all levels of business. Each year, RDBMSs support a wider variety of platforms and configurations. Indeed, the idea of an "open systems" database is hard to imagine without the relational model.
- There is now a single, standard language (SQL) for data access. As a result, RDBMS applications have achieved some degree of portability and interoperability, and training costs have dropped.
- Because of the high-level nature of relational operations, we now write and debug less code than we had to with nonrelational systems. In particular, we no longer need to write sort, merge, or filter routines, and we have a guarantee of accurate, expected results.
- Database integrity and transaction consistency can be managed outside of the application to a degree not possible with nonrelational DBMSs.
- DBMSs are significantly more reliable, recoverable, available, and functional than ever before.
- Data modeling is now a science (though analysis is not): We can identify mixes of transactions and data structures that cause data anomalies or conflicts, and fix the problems.
- DBMS performance can often be improved without requiring modification of application code (and, in many cases, without human intervention). Parallelization of DBMS operations has made great leaps forward in recent years.
- Distribution of applications and databases is rapidly becoming a viable approach, with features such as asynchronous replication, triggers, and stored procedures greatly improving both performance and ease of use.
- Enterprise-wide database designs and implementations are now feasible (though they are still in their infancy).

Despite these improvements, applications are still being developed using low-level, record-at-a-time database or file access. Some RDBMS vendors have even designed their products to facilitate such nonrelational access. They claim that their products "must work in the real world," or that the business demands that they "deliver what customers want and need."

■ Relational DBMSs are the de facto "open systems" database solution at all levels of business.

RDBMS users are still frustrated by product deficiencies. Although relational has clearly won the database model battle for industry acceptance and, in some ways, dominance, many users still find reasons to avoid an RDBMS solution. Among the more common reasons I hear are:

- RDBMSs use more I/O than their nonrelational counterparts in performing the same function.
- RDBMSs do not support complex data structures (including objects, multidimensional tables, hierarchies, and so on) or inheritance.
- RDBMSs can't manipulate certain data types (for example, textual databases).
- SQL is too difficult to use.
- RDBMS operations are inherently just too slow.

For the most part, such users tend to equate commercial RDBMS technology with the relational model, which is a serious error. RDBMS marketing, sales, and engineering personnel often make the same error. In some cases, so-called experts and vendors equate commercial RDBMS products with the relational model as an excuse for omitting important functionality and as a motivation for promoting the next product release, which will be "post-relational." I believe that the problems assigned to the relational model are actually caused by violations of the model, either in product implementation or in use.

The Relational Model: Vintage 1969

Recently, I reviewed Codd's 1969 and 1970 papers, writing down each of the features he specified for the relational model (by Codd's counting there are about 50). Each time I read them, I am impressed by the depth of the articles and their succinctness. I almost always find a gem that I previously overlooked. If you haven't read the papers, please do, but be forewarned that Codd didn't use much space explaining the features he proposed. Without the writings and lectures of C. J. Date, much of the papers would be too obscure for the average reader. (See "Interview," page 62.)

Conceptually, we can divide the features introduced in 1969 into three categories: structure, manipulation, and integrity. Let's look at each of these in turn, commenting on the status of today's RDBMSs.

First, the structural features specify a clean separation between the external view of the database and the internal view of the database, each supported by its own language. The external view includes how data is viewed and manipulated by users, programmers, and those who specify and maintain the logical content of the database. This is the place where relations, integrity constraints, and relational operations reside.

The internal view of the database includes how the data is stored, placed, and manipulated by the DBMS software (for example, the disk, tape, or memory data structure and access methods, degree of redundancy, and so on). Only vendors are intended to have direct control over manipulations performed at the internal view level. At most, users specify which of several supplied physical storage structures and index types are to support some structure defined in the external view. Users must not be permitted to mix references to the internal and external views, except as needed to map named structures in the external view to their implementation in the internal view. This mapping must exist outside the application as well, being the province of the DBMS.

Today's RDBMSs consistently fail to exploit relational's powerful separation of the internal and external views. For the most part, they provide only a single physical storage structure that simply mimics tables, giving physical database designers few choices. In particular, users seldom have the choice of how to store a relational table (for example, as an array, linked list, or tree structure); of storing multiple tables together (for example, to optimize joins or access to repeating groups) or single tables in separate partitions (horizontal or vertical); of storing multiple copies of the data; of creating multitable or functional indexes; and so forth.

Even worse, in today's RDBMSs, the

separation is corrupted: Users should be able to modify physical storage structures without affecting the external view. Instead, RDBMS storage options, which belong to the internal view, are generally offered as clauses in the SQL CREATE TABLE command, which properly belongs to the external view. This entanglement of internal and external views makes it impossible to restructure the storage of a table (its internal implementation) without dropping its external definition. Likewise, when redundancy in physical storage is supported, the external view fails to hide this fact from users, who are often made aware of the copies. A similar problem exists with respect to physical partitions of a table being stored separately on disk.

This mixing of internal and external has many negative consequences, not the least of which is a perverted understanding of the relational model. Specifically, most users think that the relational model forces data to be stored as tables, altogether failing to recognize the distinction between internal and external. Furthermore, in many implementations where internal and external issues are confused, both error behaviors and restrictions on integrity constraints depend on the physical implementation (such as the existence of certain indexes).

Without separation, vendors cannot support new storage structures and access methods in new versions of their products without affecting applications. Even worse, users cannot implement both a conceptual data model (defined by logical relationships) and a distinct physical data model (defined by resource and performance requirements). Subsequently, database administration and system management are much more difficult than necessary. This problem becomes even more apparent when vendors try to implement distributed database functionality.

■ While SQL implements most of the first-order predicate calculus and set operations, it fails in the most crucial ways to be a relational language.

Vendor implementation of the external view is only moderately successful. Certainly, users perceive data as rows and columns. However, RDBMSs violate many of Codd's specifications: They permit duplicates, do not require primary-key identification, place significance on column order, limit row width according to physical page size, do not label columns with domain names, do not support domains (active, simple, and non-simple), do not support a generation identifier, and often do not support a general naming scheme. Incidentally, Codd noted that some nonsimple domains might have relations as elements, although these would be normalized away if the DBMS knew about their nonsimple character. The concept of domain, a collection of atomic data values and legal operations on those values, is intended to hide data element complexities that are not relevant to an application or user. All columns should be defined as drawing their values from a particular domain, much in the way a mathematical function $y = f(x)$ maps values from a domain of values x into a range of values y. Domains can be either fundamental or derived, and need not be restricted to simple data types such as integers, character strings, floating point numbers, and so on.

Codd required that the language to be used in defining and manipulating the external view should be based on the first-order predicate calculus. It should support

logical inference and various restricted set operations (intersection, union, set difference, and Cartesian product). Other operations it should define include projection, column permutation, restriction, join, and composition. The language should support declaration of relations and primary keys, and should be used for queries, insertions, deletions, and updates. The language should allow the user to form expressions in which any combination of columns could be either known (the values specified by the user in the expression) or unknown (the values determined by the system as part of the result).

He noted that this language would suffice if all relations were in at least First Normal Form (primary keys, no duplicates or repeating groups, with every row representing the same kind of fact). This assumes that operations remove duplicates from a result. Codd also recognized that some deletions and updates would trigger others automatically, based on update dependencies (referential-integrity enforcement). The language should be able to invoke arithmetic and other functions defined in another language, but should not itself contain these functions.

While SQL implements most of the first-order predicate calculus and set operations, it fails in the most crucial ways to be a relational language. In particular, it treats both relations and nonrelations indiscriminately and tends to produce nonrelations as results (thus, it doesn't remove duplicates). SQL does not implement the universal quantifier, which would let you state that a property is true of all rows in a table (essentially an iterated logical AND over all values of a logical variable). It also contains arithmetic and other functions directly, rather than allowing arbitrary functions to be implemented in another language and invoked from SQL. This permits the expression of paradoxical statements while inhibiting user extensions to the language that would reduce complexity.

Codd also defined the need for declaration of time-independent integrity constraints within the language. He deemed batch constraint checking necessary, in addition to dynamic constraint enforcement (which implies dynamic constraint checking). Part of his concern was that the DBMS should be able to recognize and enforce state consistency, and deduce the redundancies applicable to a named relation.

As implemented in today's products, SQL's support for integrity constraints between relations (multitable or database constraints) remains weak. Despite the recent (and optional!) addition of declarative referential integrity to the SQL standard (SQL-89) and subsequently to most products, SQL's concept of integrity is crippled. It simply does not enforce the use of primary keys or the integrity of domains as needed to provide explicit support of active domains.

The 1979 Paper

In 1979 (about the time the first commercial RDBMS products shipped), Codd clarified and ex . nded the relational model (see Ref. 3). In this 1979 paper, he made it clear that relations could be either fundamental or derived and that some derived relations, called views, should be referable by name. He discussed how many new capabilities were either inherent in the 1969 explanation of the relational model or were natural extensions to it, including:

- 3VL (three-valued logic) to the model (with internal and external forms) and versions of the relational operations to support 3VL. This is the one serious error in Codd's work, in my opinion, because many-valued logics introduce numerous logical anomalies and non-intuitive results without adding value to the model (see Refs. 4, 5, 6, and 7)
- the divide operation (defined first in 1971)
- surrogate keys (system-generated and maintained primary keys)

TABLE 1. State of the art (selected 1969 relational features)

Feature	Implementation Status
Separation of Internal and External	
"host" language (H)	N
physical implementation defined in H	N
arithmetic (and other) functions defined in H	N
data independence	P
partial redundancy permitted in internal view	N
consistency enforced in internal view	N
connections distinct from relationships	P
independence of internal orderings	Y
independence of indexing	P
independence of access path	P
storage declaration in H	N
uncontrolled external redundancy forbidden	N
Structural	
a model based on *n*-ary relations	N
First Normal Form enforced	P
each row represents an *n*-tuple of an *n*-ary relation	P
order of rows is immaterial	Y
order of columns is immaterial	N
all rows are distinct	N
domain name labeling of columns	N
no practical limit on relation degree	N
support for names of relationships	P
active domains	N
simple domains	N
nonsimple domains	N
relations as domain elements	N
primary keys required	N
foreign keys supported	P
generation identifiers for relations	N
general naming scheme	P
Manipulative	
universal data sublanguage (R)	P
non-subvertible	Y
based on first-order predicate calculus	P
relational operators with duplicate removal	
join (various types)	N
project	N
restrict	Y
product	P
composition	N
column permutation	Y
universal and existential operators	P
used to declare relations and primary keys	P
user defined functions invoked in R	N
supports high-level data manipulation (queries, insertions, deletions, and updates)	Y
triggered deletions and updates based on dependencies	P
any aspect of a relation can be a known or unknown	Y
time-independent constraints between relations	P
relational closure	P

continued on page 60

- types and subtypes
- association relations
- generalization and specialization
- single and multiple inheritance
- cover aggregation
- graph operations
- domain operations
- operations on collections of relations
- event-type relations, with predecessor and successor functions
- various types of integrity to support the new functionality

Of this list, partial support in commercial RDBMSs has been provided for Codd's 3VL (much to the detriment of the products, in my opinion). Also, a weakened version of surrogate keys, called system-generated keys, has been added to RDBMS products during the intervening years. All the other features explained in 1979 are still lacking.

Related Technical Work

A large portion of the massive work done on transaction isolation, database recovery, database consistency, database design (dependency theory and normalization theory), and query optimization since 1969 has benefited from and been a benefit to the relational model. Academic and industrial research have made it possible to implement most of the relational features in practical ways. The viability of Codd's early goal of logical data independence continues to improve; with most views and derived tables now being updatable (see Refs. 8, 9, and 10), there is little operational reason to differentiate derived tables from base tables. To the user, all derived tables (including query results and views) work the same way as base tables.

These advances have found only partial implementation in commercial RDBMSs. For example, real query optimizers lag far behind the theory, partially because the theory deals with relations while products deal with nonrelational tables. Lack of support for the relational model is a crippling disease that vendors and users should not tolerate.

The 12 Rules and Beyond

In 1985, Codd gave a popular summarization of the relational model in his now-famous "Twelve Rules" (see Ref. 11). The rules gave the user community, in the context of the technology of the day, a set of guidelines for quickly determining whether a DBMS product deserved to be called relational. Unfortunately, but predictably, vendors seized upon the paper as a marketing tool and gave it a variety of simplistic, self-serving interpretations. Suffice it to say, these rules should not be considered the technical definition of the relational model. Beware of those who define the model so simplistically. Tell them to read Codd's 1969 and 1979 papers.

Nonetheless, the 12 rules did serve as a rallying point. They are probably responsible for emphasis in the late 1980s on referential integrity and primary keys. They also served as a platform to spread a better understanding of the prohibition against subversion of the relational language and the need for relational closure. The prohibition against subversion requires that users be allowed to access and manipulate data only through the relational operators. Closure requires that the result of any relational operation on a relation yield another relation, thereby guaranteeing that operations can be nested.

Given that Codd defined 333 rules in his 1990 book (see Ref. 12), vendors often complain that the number of requirements for their products to qualify as relational keeps growing. This is a misunderstanding. Version 2 of the relational model consists primarily of the features of the 1969 paper, along with a few features from the 1979 paper, with considerable detail added. In other words, vendors should have been able to implement the greater portion of Version 2 based on the 1969 and 1979 papers. Reading the preface to Codd's 1990 book, you can detect considerable frustration and disappointment in Codd's perception of the state of the industry. I heartily sympathize with these feelings.

Well, What Do You Know?

Perhaps the greatest failure of the RDBMS industry is inadequate training. Throughout the commercial existence of relational products, there has not been a time when either users or vendors have had access to a good supply of trained relational professionals. Neither vendors nor users can be held fully responsible for this problem. In a sense, its existence is a measure of just how successful even the partial implementation of the relational model has been.

Vendors, in their quest to bring products to market quickly, have been forced to hire individuals who do not understand the model. These well-meaning individuals have marketed, defined, and designed products, changing forever the market's understanding of what is relational. Similarly, users have had to obtain training from these vendors, resulting in applications designed to use anything but the relational model. Few so-called relational professionals really understand the theory, and therefore cannot assess the root cause of product or application failures. This situation, ultimately caused by the premature success of pseudo-relational products, has led to considerable dissatisfaction among users. The result is cries for denormalization, navigational access, and object databases.

■ I have to admit that I have never seen a relational DBMS.

The Neo-Relational Model

I have to admit that I have never seen a relational DBMS. I have seen many great pseudo-relational (or SQL) DBMSs; that is, products labeled relational that implement a few of the ideas in the relational model, but simply ignore many of the crucial ones. Nonetheless, I can't say I want to go back to the days of the pre-"relational" DBMS. Writing and maintaining applications with prerelational DBMSs was just too hard and too uncertain. Designing and maintaining the database was worse.

It is amazing that, after 25 years, RDBMS vendors have heeded so little of Codd's 1969 paper. (See Table 1, pages 56 and 60.) However, the reason becomes clear when I ask relational professionals if they have ever read either the 1969 paper or the 1970 version of it. Most say they have not.

Perhaps we need a new name for the relational model. Let the existing products have the old name. We'll certify them as "truly relational." They can advertise as such. Then we'll explain to them that what we want and need, and what they will profit from the most, is something called the "neo-relational model." The neo-relational model will solve most, if not all, of the problems we have with "truly relational" products. Best of all, this great model was invented in 1969 by a guy named Codd. We can even show them his definitive research papers! They are publicly available.

TABLE 1. State of the art (selected 1969 relational features) (concluded)

Integrity	
system-captured detailed semantic information	P
system deduction of applicable redundancies	N
constraint statements	P
system-determined state consistency	P
dynamic constraint checking	P
batch constraint checking	N
user-accessible journal of state changes	N

Y = yes; N = not supported; P = partially supported

Notes: This table is based on the typical features and functionality in the major RDBMS products, including CA-OpenIngres, Cincom Supra, IBM DB2, Hewlett-Packard Allbase/SQL, Informix-OnLine, Oracle 7, and Sybase SQL Server System 10. The evaluation is conservative: If the feature is not implemented consistent with the definitions of relation or relational operations, if it is implemented in the internal view when it should be in the conceptual view (or vice versa), or if the implementation impairs the relational promise, the evaluation is either N or P (partially supported), depending on the degree of support. This list should not be used for [illegible] solely for the purpose of illustrating the current lack of [illegible] its constituent features.

References

1. Codd, E. F., "A Relational Model of Data for Large Shared Data Banks," reprinted in *Readings in Database Systems*, M. Stonebraker, ed., Morgan Kaufmann, 1988. (Originally published in 1970.)
2. Codd, E. F., "Derivability, Redundancy and Consistency of Relations Stored in Large Data Banks," *IBM Research Journal*, RJ 599 (#12343), August 19, 1969.
3. Codd, E. F., "Extending the Database Relational Model to Capture More Meaning," reprinted in *Readings in Database Systems*, M. Stonebraker, ed., Morgan Kaufmann, 1988. (Originally published in 1979.)
4. McGoveran, D., "Nothing From Nothing (or, What's Logic Got to Do With It?)", *Database Programming & Design*, 6(12):32–41, December 1993.
5. McGoveran, D., "Classical Logic: Nothing Compares 2 U," *Database Programming & Design*, 7(1):54–61, January 1994.
6. McGoveran, D., "Nothing From Nothing Part III: Can't Lose What You Never Had," *Database Programming & Design*, 7(2):42–48, February 1994.
7. McGoveran, D., "Nothing From Nothing Part IV: It's In the Way That You Use It," *Database Programming & Design*, 7(3):54–63, March 1994.
8. Date, C. J. and McGoveran, D., "Updating Union, Intersection, and Difference Views," *Database Programming & Design*, 7(6):46–53, June 1994.
9. Date, C. J. and McGoveran, D., "A New Database Design Principle," *Database Programming & Design*, 7(7):46–53, July 1994.
10. Date, C. J. and McGoveran, D., "Updating Joins and Other Views," *Database Programming & Design*, 7(8):43–49, August 1994.
11. Codd, E. F., "How Relational Is Your Database Management System?" *Computerworld*, October 14 and 21, 1985.
12. Codd, E. F., *The Relational Model for Database Management, Version 2*, Addison-Wesley, 1990. ■