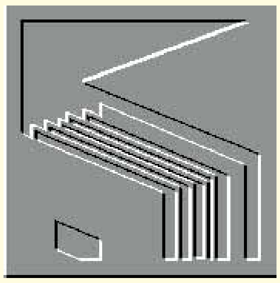


Filleted **Codd**

Ted Codd's rules are quite involved; so Mark Whitehorn has wrapped up an easily digestible takeaway message on the subject of RDBMSs, with further reading recommended.

Over the past few months, I have covered the 13 rules that Ted Codd originally defined for RDBMSs. As I mentioned at the start, there is a fine line between accuracy and verbosity. Even though I have carefully trodden this line, it has taken a long time to cover all the rules; so long that it may be difficult to see the take-home message. Additionally, some of these rules have become less meaningful as the years have passed and RDBMSs have evolved.

With this in mind, it seems worthwhile to try to determine some of the important characteristics we should expect from a modern RDBMS running on the PC. This does not include heavyweight, mission-critical RDBMSs, but it does include the Access, Paradox, dBase, Approach, Delta 5, dataEase, SuperBase type of products.

Paradox 5.0 for Windows Credit Screen
Borland Paradox for Windows
Start Credits
Stop Credits
Resume
Speed up
Slow Down
Reset

Any reference to any real persons, living or dead, is strictly coincidental. No animals (ducks in particular) were harmed (or hit by lightning) during the production of this credit screen.

Fig 1 *The gang screen in Paradox 5.0 for Windows*

Before anyone asks, I'm not presenting these as Whitehorn's Rules (I haven't even numbered them from zero); they are not definitive, neither are they complete. View them more as a starter set designed to promote discussion. If you think any are wrong, incomplete, or missing, then please let me know.

The following are *not* in order of importance:

• An RDBMS must store data as values in tables and not in any other way.
• It must be possible to declare a primary key for each and every table and it must be possible to use multiple fields to form a primary key.
• The RDBMS must ensure that any field declared as a primary key, or part of a primary key, is not allowed to contain null values.
• Every piece of information in a table must be accessible by using a combination of the table name, field name and primary key value.
• Null values must be not be treated as equal in joins.
• Joins on non-identical field types must not be allowed.
• When joins are performed on tables containing existing data, the referential integrity of that existing data must be checked and the join must fail if the data violates the proposed join.
• An RDBMS must maintain a data dictionary for each database which stores information about the joins between the tables, referential integrity, etc. Access to the tables which circumvents this data dictionary should be forbidden.
• Rules controlling data entry to specific fields must be storable in the data dictionary and applied at the table level.
• The RDBMS must have a comprehensive control language (for example, SQL).
• In addition, the RDBMS must have a GUI interface which allows end-users to perform simple tasks such as querying, reporting etc.
• The results of queries (answer tables) should, whenever possible, offer the option of editing.
• It must be possible to alter multiple records with a single command.
• The RDBMS must support referential integrity, with cascade update, cascade delete and so forth.
• The RDBMS must support the maintenance of indices as well as sorting.

I must stress that this list merely represents a starting point for discussion so, contributions please. Meanwhile, I can explain some of my choices:

"An RDBMS must store data as values in tables and not in any other way" — this is the Same as Codd's rule 1 (see *PCW*, March 1995).

"It must be possible to declare a primary key for each and every table, and it must be possible to use multiple fields to form a primary key" — most Windows RDBMSs support primary keys but not all support the use of multiple fields as primary keys. I find this to be a mind-boggling omission for the simple reason that if they are not supported, it is impossible to make a many-to-many join.

For those who aren't familiar with the concept: suppose you want to keep a database which stores information about Students and the Courses they attend. You need a table for each, but how do you store the information about which student attends which course(s)? The problem is that each attends many courses and each course is attended by many students. Many-to-many relationships are common in the real world and hence they are common in databases.

If you were foolish, you might try to include a field in the Student table for every course that the student may attend. But how many fields do you allow? The average student might attend six courses a year but the very industrious could go to ten — an exceptionally gifted student might want to attend 12; so what if your system was to allow only for eight?

A much better way is to use a Students table only to store data about the students themselves; a Courses table to store data relating to the actual courses; and a third table to store information about who is attending which course. A brief examina-

MICROMART
CLASSIFIED

tion of the tables in *Fig 2* should be enough to allow you to determine that Mike Wellington is attending Cytogenetics and Intro. to Polymorphism (among others). However, Sally Jones isn't attending either of those courses (despite attending 11 others). It should also reveal that if she wants to add Cytogenetics to her list of courses, all that is necessary is to add a single record to the Attend table with the entries 2 and 2.

Of course, you don't actually go to this table and write these numbers in; you use an attractive GUI interface which allows you to pick the student's name from one combo box and the course from another. Then the system writes the numbers in for you.

So, what has all this to do with multiple fields in primary keys? Hopefully, it is clear that ID has to be a primary key in the Students table to ensure that each student has a unique number. Similarly, CODE must be a primary key in Courses. However, neither ID nor CODE on its own, can be the primary key in the Attend table. Instead, the primary key must be composed of both fields, used together. If both are used, the table can have multiple entries in the ID field, as it can in the CODE field as follows:

ID	CODE
1	2
1	3
2	2

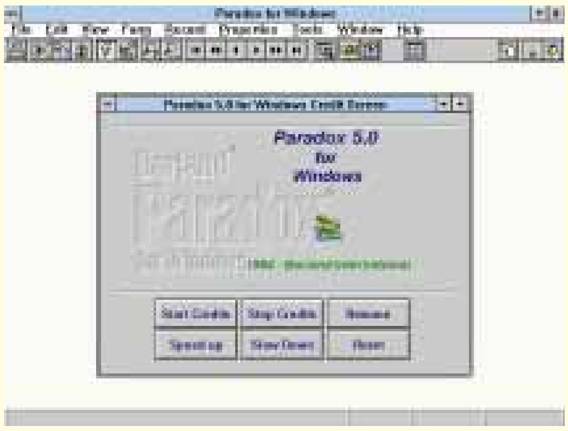
But the following is forbidden:

ID	CODE
1	2
1	2

This actually matches reality very well since the same student, no matter how gifted, cannot attend the same course more than once (at least, not simultaneously).

Any RDBMS that doesn't support multiple fields as a primary key cannot manage many-to-many joins effectively.

See the join?



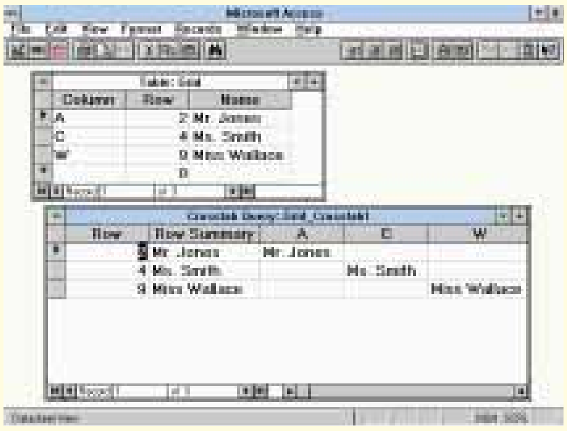


Table: Grid

Column	Row	Name
A	2	Mr. Jones
C	4	Ms. Smith
W	9	Miss Wallace
	0	

Crosstab Query: Grid_Crosstab

Row	Row Summary	A	C	W
2	Mr. Jones	Mr. Jones		
4	Ms. Smith		Ms. Smith	
9	Miss Wallace			Miss Wallace

Fig 2 (top) *Using a third table to create a many-to-many join between two others (see page 306)*
Fig 3 (above) *The solution to the problem of gridding data (see page 309)*

Time wasting

Oh no, not another another gang screen *(Fig 1)*. This time it's Paradox for Windows 5.0:

1. Turn ScrollLock on, NumLock on, and CapsLock off.
2. Select Help, About.
3. Press Alt Shift Z (that is, the three keys simultaneously).
4. Hold down Ctrl and Left click the logo in the Help About box.

Assuming that the wind is from the South, and that there are no more than two penguins in the room with you, the gang screen will appear *(Fig 1)* — note the snide remark about ducks.

To understand this, you need to know that: the code name for Access was Cirrus; and that the original Access gang screen shows a pair of ducks (Pairodux), which are destroyed by a lightning bolt from a small, fluffy cloud.

Recommended reading

When I started to run through Ted Codd's original rules some months ago, I promised a book list of recommended texts and asked readers for suggestions. Thanks to all who contributed — I have grouped the books below (numbering system after Ted Codd):

Group 0: Readable, informative, non-rigorous.

Title	**Author**	**Publisher**	**ISBN**
The Relational Database	Carter	Chapman and Hall	0-412-55090-3
SQL and Relational Databases	Vang	Microtrend	0-915391-42-2

Group 1: Less readable, still informative, more rigorous.

Title	**Author**	**Publisher**	**ISBN**
An Introduction to Database Systems	Date	Addison Wesley	0-201-54329-X
Understanding Relational Databases (with examples in SQL-92)	Pascal	John Wiley & Sons	0-471-585-38-6

Group 2: Rigorous, not bedtime reading unless you're rather weird.

Title	**Author**	**Publisher**	**ISBN**
Relational Database Writings 1985 -1989	Date	Addison Wesley	0-201-50881-8
Relational Database Writings 1989 -1991	Date with Darwin	Addison Wesley	0-201-54303-6
Relational Database Writings 1991 -1994	Date	Addison Wesley	0-201-82459-0
Fundamentals of Database systems and Navathe	Elmasri	Benjamin Cummings	0-8053-1753-8

Group 3: Completely turgid, totally rigorous, not worth reading unless you are a masochist, but essential to have on your bookshelf. If you want to appear a true professional, scuff up the book a little so that it appears well-thumbed and annotate the margin occasionally in pencil. These annotations are better if they appear cryptic:
"!" is a good one.
"Really!" is excellent
"Really!?" is even better.
"No" is dangerously authoritative and best avoided
"Ted now considers this incorrect — Pers. Comm." is the ultimate, as long as you think you can carry it off.

Title	**Author**	**Publisher**	**ISBN**
The Relational Model for Database Management Version 2	Codd	Addison Wesley	0-201-14192-2

Joking apart, if you really want to know where Ted Codd's thinking went after the original rules, it's all in this book. But don't expect a light read. Incidentally, Codd deals with an interesting problem in this book: in 1988, H. W. Buff published a paper entitled "Why Codd's rule No. 6 Must Be Reformulated" proving that Rule 6 is flawed. Rule 6 says that essentially, all views (answer tables) should be updatable if an algorithm can show that it is "safe" to do so. (This statement is rather simplified, but see *PCW* June 1995 for more detail.) Buff's paper shows that an RDBMS can never support this rule because "there does not exist any algorithm which can decide, given any view, whether it is updatable or not". In this book, Codd has modified rule 6 by defining an algorithm which will identify a good percentage of updatable views. I knew of the problem from Codd's book but had never seen Buff's paper. I am greatly indebted to Mike Jackson, a Reader in Software Engineering at the University of Wolverhampton, who sent me a copy. For most of us, the fact that this rule is unenforceable under all circumstances is not crucial. What does matter is that RDBMSs like Access will let you edit most answer tables from queries, and RDBMSs like dBase do not.

Questions & Answers

Too small

"In the June issue, the screenshots containing SQL code are tiny and very difficult to read. Could you put the SQL in the text body where it can be more easily read?"
Yes, and sorry for the problem.

On the rack

"I hope you can help me with a query regarding Microsoft Access Reports. I have a table of data containing a person's name and a grid reference (e.g Mr Jones. A1). I would like to display a grid in a report with column headers from A to Z, and row headers from 1 to 24. I would then like to place the person's name in the corresponding grid reference. For example:

	A	B	C
1	Mr Jones		
2			
3			

…and similarly throughout the grid.

I would be grateful if you could give me some pointers as this is giving me some headaches, and sleepless nights."

I think that a query would serve you better than a report in this instance: I have made a small table (fig 4) and built a cross tab query which should satisfy your needs. The SQL for the crosstab query is:

```
TRANSFORM First(Grid.Name) AS
[The Value]
SELECT Grid.Row, First(Grid.Name) AS
[Row Summary]
FROM Grid
GROUP BY Grid.Row
PIVOT Grid.Column;
```

I have made the fields Column and Row the joint primary key, which means that you can only ever have a single entry for each grid location. If you want the grid to be complete (that is, to show every possible cell, irrespective of whether or not it contains a person's name), you can add every single grid location to the table and leave the value in the name field null. Or, you could just add A1 to A24 and B1 to Z1 to the table, whereupon the answer table will obligingly show every possible location. If you don't want to see the Row Summary (which is inserted by the Access wizard), simply delete it from the query.

PCW Contacts

Mark Whitehorn welcomes readers' correspondence and ideas for the Databases column. He's on **penguin@cix.compulink.co.uk**