The Askew Wall

SQL and The Relational Model
(background to *The Third Manifesto*)

Hugh Darwen
HD@TheThirdManifesto.com

Last updated: January 2006

1

Terminological Equivalences (?)

Posh term	**Cuddly term**
Relation	Table
(n-)tuple	Row
Attribute	Column
Domain	(data) type
Domain	(object) class

The table represents a relation! Its predicate:
Cuddly term is the SQL counterpart of Relationland's *Posh term*.

2

The Perversity of SQL

```
SELECT CityName
FROM City C1
WHERE 4 > (SELECT COUNT(*)
           FROM City C2
           WHERE C1.Population < C2. Population)
```

The Unperversified Version

```
SELECT CityName
FROM City C1
WHERE  (SELECT COUNT(*)
        FROM City C2
        WHERE C2. Population > C1. Population) < 4
```

3

The Third Manifesto

or

Date
and Darwen's
Database Dream

4

References

Relational Database Writings 1985-1989
by C.J.Date with a special contribution
"Adventures in Relationland"
by H.D. (as Andrew Warden)

Relational Database Writings 1989-1991
by C.J.Date with Hugh Darwen

Relational Database Writings 1991-1994
by C.J.Date

Databases, Types, and The Relational Model :
The Third Manifesto
by C.J. Date and Hugh Darwen (to appear 2006)

Introduction to Database Systems
(8th edition) by C.J. Date

5

A Brief History of Data

1960: Punched cards and magnetic tapes
1965: Disks and 'direct access'
1970: E.F. Codd's great vision:
"A Relational Model of Data
for Large Shared Data Banks"
1970: C.J. Date starts to spread the word
1975: Relational Prototypes in IBM:
PRTV (ISBL), QBE, System R
1980: First SQL products: Oracle, SQL/DS
1986: SQL an international standard
1990: OODB – didn't come to much in the end
2000: XML? (shudder!)

6

A Brief History of Me

1967 : IBM Service Bureau, Birmingham

1969 : "Terminal Business System" – putting users in direct contact with their databases.

1972 : Attended Date's course on database (a personal watershed)

1978 : "Business System 12"
- a relational dbms for the Bureau Service

1985 : Death of Bureau Service (and of BS12)

1987 : Joined IBM Warwick dev. lab. Attended a Codd & Date database conference in December

1988 : “Adventures in Relationland” by Andrew Warden. Joined SQL standardization committee.

2004 : Retired from IBM

The Wall Around Relationland

The

ASKEW

Wall

What The Askew Wall Has Done

Lots of Good Things, to be sure, but ...

- Untold damage to the Relational Model’s reputation.
- Stifled research in the relational field.

People even think the Wall is Relationland.

There have even been moves back to the “Higgledy-Piggledy Model of Data” (Object Oriented Databases) and the hierarchical model (XML).

The Good Things The Askew Wall Has Done

Codd's vision has come true in the following respects:

- TABLE as the only available structure.
- Value at row/column intersection the ONLY method of storing information.e.g., no pointers, no ordering of rows.
- Orthogonality of tables with respect to data types (domains) over which their columns are defined.
- The catalogue is made of tables, too.
- Query language ALMOST closed over tables and does embrace relational algebra/calculus principles (as well as regrettably departing from them).
- Constraints expressed declaratively, in the schema, and enforced by the dbms.
- No "record-level" (or other) subversion.

but...

The Fatal Flaws of SQL

- Anonymous columns (partly addressed in 1992)
- FROM clause restricted to named tables (fixed in 1992)
- Duplicate column names
- Order of columns is significant
- Duplicate rows
- NULL
- Failure to support degenerate cases (e.g. columnless tables)
- Failure to support “=“ properly
- and lots more, and probably to come

COLUMN NAMING FLAWS

A Thematic Query Example

Ref: "The Naming of Columns", chapter 17 in RDBW 1985-89

Given :

ExamMarks

Name	Subject	Mark
Anne	Relational DB	92
Boris	Object DB	68
Cindy	Object DB	56
Dave	Relational DB	84

To derive:

Name	Subject	Mark	Avg
Anne	Relational DB	92	88
Boris	Object DB	68	62
Cindy	Object DB	56	62
Dave	Relational DB	84	88

Anonymous Columns

Ref: "The Naming of Columns", chapter 17 in RDBW 1985-89

Example 3:

Show the average exam mark obtained by all students in each subject.

```
SELECT     Subject, AVG(Mark)
FROM       ExamMarks
GROUP BY   Subject
```

The "second" column of this table has no name!

This is a CORRECTABLE flaw (well, NEARLY correctable). It was corrected (kind of) in 1992.

It is NOT BYPASSABLE

FROM Clause Restricted to Named Tables

Example 4:

Show for each student in each subject the mark obtained and the average mark by all students in that subject.

```
SELECT          Name, E.Subject, E.Mark, S.??? -- unnamed column
FROM            ExamMarks E,
                  (SELECT    Subject, AVG(Mark)
                   FROM      ExamMarks
                   GROUP BY Subject)  S
WHERE           E.Subject = S.Subject
```

This is a correctable flaw (and was corrected in 1992). It is not generally bypassable, though sometimes you can create a named view for the "nested" query.

Actually, this particular query CAN be done without nesting (exercise for reader!), but the solution cannot be generalized.

The FROM clause fix

Example 4 (fixed):

Show for each student in each subject the mark obtained and the average mark obtained by all students in that subject.

```
SELECT          Name, E.Subject,E.Mark,S.Avg
FROM            ExamMarks AS E,
                  (SELECT    Subject, AVG(Mark) AS Avg
                   FROM      ExamMarks
                   GROUP BY Subject) AS  S
WHERE           E.Subject = S.Subject
```

This is still only an optional conformance feature in SQL:2003. I think it is *very* important. Without it SQL is relationally incomplete.

But it's clunky!

With WITH

Example 4 (fixed and made possibly a bit more digestible):

```
WITH AvgMarks AS
   (SELECT    Subject, AVG(Mark) AS Avg
    FROM      ExamMarks
    GROUP BY Subject)

SELECT Name, E. Subject,E.Mark,A.Avg
FROM    ExamMarks AS E, AvgMarks AS A
WHERE  E.Subject = A. Subject
```

WITH is an optional conformance feature in SQL:2003. Not many implementations have it, and even those that do have it do so only with unpleasant restrictions.

Duplicate column names (1)

Ref: "In Praise of Marriage", chapter 18 in RDBW 1985-89

Example 5:

```
SELECT          *
FROM            ExamMarks E, Student S
WHERE           E.Name = S.Name
```

A very natural-looking join, but there are two columns called Name. These are also duplicate columns, as it happens.

Sometimes such joins generate two columns both called, e.g., Remarks, that are not duplicate columns.

Duplicate Column Names (2)

Thanks to AS you can now even do this :

```
SELECT Col1 AS X, Col2 AS X
FROM    T
```

Enjoy!

19

A Fix for Duplicate Column Names (1)

NATURAL JOIN was added to SQL in 1992 but not widely implemented.

```
SELECT      *
FROM        ExamMarks NATURAL JOIN Student
```

Note elimination of:
- need to write a joining condition in the WHERE clause
- need to write a possibly long list of column names to avoid having the same column twice

20

Why NATURAL JOIN is "Natural"

Relational JOIN is the relational counterpart of logical AND.

Assume predicates:
ExamMarks "*Name* scored *Mark* in *Subject".*
Student "*Name* joined in *Year*, was born on *DoB* [etc.]"

Then the predicate for

ExamMarks JOIN Student

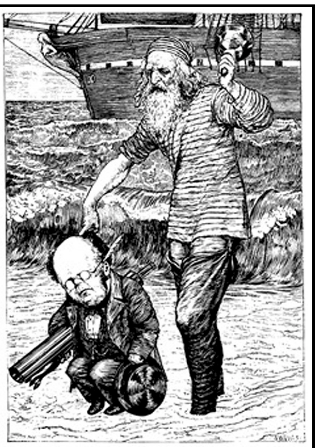
is "*Name* scored *Mark* in *Subject* and *Name* joined in *Year*, was born on *DoB* [etc.]"

The two *Name*s are the same variable!

21

DUPLICATE ROWS

22



Duplicate Rows

Ref: "The Duplicity of Duplicate Rows", chapter 5 in RDBW 89-91 "The Keys of the Kingdom", chapter 19 in RDBW 85-89, and:

"If something is true, saying it again doesn't make it any truer"
(E.F. Codd)

<u>This is a bypassable flaw:</u>

- Declare at least one candidate key for every base table.
  and ask for support for system-generated keys.
- Always write DISTINCT after the word SELECT
  and complain to supplier if this makes duplicate-free queries go slower.
- Never write the word ALL after UNION
  and demand decent optimization here, too.

but, alas, it is not a correctable flaw.

Usability problems should be recognized and solved, but NOT by departing from fundamental principles.

24

Are Duplicate Rows Really Harmful?

Well, they needlessly complicate the language, making it more difficult than it ought to be to define, teach, and learn.

And they allow the implementation to place some of the performance burden needlessly on its users (when is it safe to write UNION ALL or to omit DISTINCT?)

But look at the trap somebody fell into here:

```
SELECT COUNT(*) AS HowMany, AVG ( Age ) AS HowOld
FROM
  (SELECT Emp#, Age FROM Emp NATURAL JOIN
     (SELECT Emp#
      FROM WorksIn
      WHERE Dept# IN ( 'D3', 'D7' ) ) AS dummy ) AS dummy
```

“How many people work in departments 3 and 7, and what is their average age?”

In Tutorial D:

```
SUMMARIZE
  ( ( ( WorksIN WHERE Dept# = 'D3' OR Dept# = 'D7' ) { Emp# }
    JOIN Emp ) { Emp#, Age } )
  ADD ( COUNT AS HowMany, AVG( Age ) AS HowOld )
```

25

NULL

26

NULL

Ref: "Into the Unknown", chapter 23 in RDBW 85-89. See also chapters 8 ("NOT" is not 'Not!") and 13 ("EXISTS is not 'Exists'!" and the whole of part IV(chapters 17-21) in RDBW 89-91

Cause of *much* debate and anguish.

There's even a split in the relational camp (E.F. Codd proposed "A-marks", "I-marks" and a 4-valued logic).

How many different things can NULL mean? Is it valid to treat all nulls alike?

NULL ruins everything –
- UNION of sets, cardinality of sets.

Destruction of functional dependency theory

SQL’s implementation of NULL is even worse than the best suggested by theoreticians. And it’s not completely BYPASSABLE, because SQL thinks that the sum of the empty set is NULL! Nor is it CORRECTABLE.

27

Is NULL a Value?

```
CREATE TABLE NT ( N INTEGER ) ;
INSERT INTO NT VALUES NULL;
INSERT INTO NT VALUES NULL;
```

Now, what is the cardinality of:

(a) SELECT * FROM NT WHERE N = N
(b) SELECT * FROM NT WHERE N <> N
(c) SELECT DISTINCT * FROM NT

Which answers are consistent with NULL being a value?

What difference would it have made if it *had* been a value?

28

3-Valued Logic: The Real Culprit

Relational theory is founded on classical, 2-valued logic.

A relation *r* is interpreted as a representation of the extension of some predicate *P*.

Let *t* be a tuple with the same heading as *r*.

If tuple *t* is a member of *r*, then the proposition *P*(*t*) is taken to be TRUE; otherwise (*t* is not a member of *r*), *P*(*t*) is taken to be FALSE.

There is no middle ground. The Law of The Excluded Middle applies.

There is no way of representing that the truth of *P*(*t*) is unknown, or inapplicable, or otherwise concealed from us.

SQL’s WHERE clause *arbitrarily* splits at the TRUE/UNKNOWN divide.

29

Surprises Caused by SQL’s NULL

1. SELECT * FROM T WHERE X = Y OR NOT (X = Y)
   is not equivalent to SELECT * FROM T
2. SELECT SUM(X) + SUM(Y) FROM T
   is not equivalent to
   SELECT SUM(X + Y) FROM T
3. IF X = Y THEN ‘Yes’; ELSE ‘No’
   is not equivalent to
   IF NOT (X = Y) THEN ‘No’; ELSE ‘Yes’

Thanks to Lex de Haan for these, from his collection.

See http://www.oracle.com/technology/oramag/oracle/05-jul/o45sql.html

30

Why NULL Hurts Even More Than It Once Did

Suppose "*x* = *x*" returns "unknown"

Can we safely conclude "*x* IS NULL" ?

Suppose *x* "is not the null value"?

Can we conclude "*x* IS NOT NULL"?

Not in modern SQL!

31

How *x*= *x* *Unknown* Yet *x* NOT NULL

For example:

1. *x* is ROW (1, null) - or even ROW(null, null)
   ROW(...) is a row "constructor".

2. *x* is POINT (1,null)
   POINT(a,b) is a "constructor" for values in the user-defined data type POINT.

3. *x* is ROW (POINT(1,1), POINT(null,3))

Consequences?

32

NULL Misleads Optimisers!

Latest example: (Oracle9i Enterprise Edition Release 9.2.0.7.0)

```
SQL> SELECT * FROM T WHERE C=C OR NOT C=C;

C(X, Y)
----------------
POINT(1, 2)
POINT(1, 2)

SQL> SELECT * FROM T WHERE ( NOT C=C ) OR C=C;

C(X, Y)
-----------------------
POINT(1, NULL)
POINT(1, 2)
POINT(1, NULL)
POINT(1, 2)
```

33

TABLE_DEE AND TABLE_DUM

34

TABLE_DEE and TABLE_DUM

Ref: "TABLE_DEE and TABLE_DUM", chapter 22 in RDBW 85-89, and "The Nullologist in Relationland, or Nothing *Really* Matters", chapter 13 in RDBW 89-91

Two very important relations that SQL is unaware of.

Consider the question, "Do we have any students?"

In **Tutorial D**: Student { }

SQL counterpart would be SELECT DISTINCT FROM Student

The result is a relation of degree 0 (no attributes) and either one tuple (TABLE_DEE) or none (TABLE_DUM).

Interesting property of TABLE_DEE: for every relation *r*, *r* JOIN TABLE_DEE = *r*

35

Failure to Recognise DEE and DUM

Consequences of SQL's failure to recognise DEE and DUM:

* Can't have a table with no columns.
  * Can't DROP the only remaining column.
    Correctable, not bypassable.
  * Can't SELECT no columns at all.
    Correctable, somewhat bypassable.

* FROM clause can't specify "no tables".
  Correctable, somewhat bypassable.

* Primary and foreign keys can't be empty.
  An empty PK implies at most one row.
  Correctable, not bypassable.

and the above set of nullological observations is still growing

36

Bypasses for Absence of DEE and DUM

Example 6:

"Did any student obtain more than 75 marks in Relational DB?"

```
SELECT   DISTINCT 'Yes!'
FROM     ExamMarks
WHERE    Mark > 75 AND Subject = 'Relational DB'
```

Example 7:

"What's the time?"

```
SELECT   DISTINCT CURRENT_TIME
FROM     Student
```

37

It Could Have Been Worse ...

... if SQL had paid proper attention to degenerate cases.

SQL fails to recognise TABLE_DEE and TABLE_DUM. These in turn depend on the existence of the 0-tuple. Suppose SQL had not made this oversight. Then …

```
CREATE TABLE T ( C1 ROW ( ) ) ;
INSERT INTO T VALUES ( ROW ( ) ) ;
```

Query	Result Cardinality
SELECT * FROM T WHERE C1 IS **NOT** NULL	1
SELECT * FROM T WHERE C1 IS NULL	1

C1 "is not the null value"; also, no field of C1 "is the null value".
But it is also true that *every* field of C1 "is the null value"!

At least 5 errors of language design are involved in this little example!

38

MISCELLANEOUS FURTHER FLAWS

39

"=" Is Not "equals"

Modern SQL supports user-defined "equals" functions, for user-defined data types.

We would like to require these to honour the rule that if a=b then for all f, f(a) = f(b)

Unfortunately SQL itself already fails to honour it: 'A' = 'A ', but Length('A') < Length('A ')

Unpleasant consequences for GROUP BY, NATURAL JOIN, DISTINCT, foreign keys, etc.

40

The Sin SQL Has Not Committed

(yet)
In the Relational Model, the only method of representing information is by a value at some row/column intersection in some table.

The proponents of TSQL2 (temporal extensions to SQL) want "hidden" timestamps.

Violation of Relationland's uniformity of representation of information - hidden data needs additional operators to access it.

41

Clunkiness of SELECT-FROM-WHERE

In SQL:

```
SELECT Name, Total_Pay FROM
      (SELECT E.*,                      Must qualify * here!
              Salary + Bonus AS Total_Pay
       FROM Emp E) AS dummy
WHERE Total_Pay > 1000
                                        Required but useless name!
```

In **Tutorial D**:

```
EXTEND Emp ADD
             ( Salary + Bonus AS Total_Pay )
WHERE Total_Pay > 1000 {Name, Total_Pay}
```

42

A Murky Story

On the subject of:
SELECT E.*, Salary + Bonus AS Total_Pay
...

Must qualify * here!

In 2005 a UK proposal to correct the silly mistake and allow
SELECT *, Salary + Bonus AS Total_Pay
was vigorously opposed by the USA, led by Oracle and IBM, and consequently defeated. Why?

Because "* leads to maintenance nightmares, and [we are] not aware of any customer request or requirement for the feature" and "its use should be discouraged".

43

Why "Maintenance Nightmare"

Because of yet another violation by SQL of the relational model, which stipulates that there is no significance to any *order* in which the attributes of a relation might appear.

But for getting data out of and into a table, SQL uses

SELECT/FETCH INTO :v1, :v2, … and INSERT INTO *target source*

The mapping from source columns to target columns is by column number.

Defining the order means defining it for every query operator, including FROM and UNION, which thus fail to be commutative (as they should be, as relational counterparts of AND and OR). Are they associative?

The correct approach is to map columns to variables by name, not by order.

44

"Nobody Ever Asked For It"

A couple of other nice ideas from ISBL that "nobody has ever asked for" :

SELECT all columns except specified one. **In Tutorial D**:

r { ALL BUT a, b, … }

Very often you know the ones you don't want and there aren't so many of them either. (In 2005 a UK proposal to add * EXCEPT (…) to SQL was also rejected on the grounds that it would encourage use of the hated *.)

Rename selected columns and keep the others. In Tutorial D:

r RENAME (a AS x, b AS y, …)

Handy for getting the right attributes to match up for JOIN, UNION, etc.

45

The Folly of "Structured Queries"

In old SQL, the WHERE clause could not be used on results of aggregation, so they had to invent HAVING (with same meaning as WHERE):

```
SELECT D#, AVG(Salary) AS Avg_Sal
FROM Emp
GROUP BY D#
HAVING AVG(Salary) >999
```

But would we ever have had HAVING if in 1979 one could write:

```
SELECT * FROM
     ( SELECT D#, AVG(Sal) AS Avg_Sal
       FROM Emp
       GROUP BY D# ) AS dummy
WHERE Avg_Sal > 999                    ?
```

46

The Clunkiness Again

In SQL (as just seen):

```
SELECT * FROM
     ( SELECT D#, AVG(Sal) AS Avg_Sal
       FROM Emp
       GROUP BY D# ) AS dummy
WHERE Avg_Sal > 999
```

In **Tutorial D**:

```
SUMMARIZE Emp BY { D# } ADD ( AVG(Sal) AS Avg_Sal )
WHERE Avg_Sal > 999
```

47

The Folly of Coercion

Consider the so-called scalar subquery, expressed by placing a query inside parentheses:

```
SELECT D#, ( SELECT  E#
             FROM  Emp E
             WHERE E.D#=D.D# ) AS Emps
FROM Dept D
```

Scalar subquery or nested table ?
I.e., is Emps an employee number or a set of employee numbers? (loosely speaking)

(Answer: an employee number)

48

Why The Flaws Are "Fatal"

❖ The Shackle of Compatibility
(existing syntax cannot be deleted)

❖ The Growth of Redundancy
(plugging holes gives new solutions where existing solutions already available)

❖ Desired extensions can be difficult or impossible to specify
(e.g., because of nulls)

❖ Also shackled by existing style

49

Errors Here to Stay

* Duplicate Rows

* NULL

* Columns being ordered

* SELECT-FROM-WHERE

* scalar subqueries

(and probably many others)

50

The Growth of Redundancy

Since SQL:1992, the following features (e.g.) have been redundant:

- subqueries
- correlation names
- doing joins in longhand
- the HAVING clause
- the GROUP BY clause

51

OBJECT SUPPORT

52

Object Oriented Databases

A couple of good ideas:

- Database variable types available for local variables too
(but a relational database *must* be restricted to *relation* variables)
- User-defined types (classes)

A questionable idea:

- Type inheritance via extension rather than by specialisation
(e.g., 3D_POINT subtype of 2D_POINT ???)

Bad ideas (for relational database purposes):

- "Persistence is orthogonal to type" (see first good idea)
- Object identifiers (because they are *pointers*)
- Class extents (when used for the purpose of relations)
- Operator ("method") definitions bundled with type definitions
- "Selfish methods" (I.e., the "distinguished parameter")

53

Rapprochement

A bringing together of objects and relations.
Widely sought, because:

* Some Objectlanders wanted to be able to do what Relationlanders do with tables - specially ad hoc queries and declarative constraints.

* Some Relationlanders wanted to do some more complicated things that require user-defined data types of arbitrary complexity.

54

A Multimedium Relation

BirdName	Info	Pic	Video	Song	Migr
Robin					
Thrush					
Sparrow					

Predicate:
Info is information about bird BirdName, and
Pic is a picture of BirdName, and
Video is a video of BirdName, and
Song is BirdName's song, and
Migr is BirdName's migration route.

55

SQL for The Bird Table

Correct (and supported):

```
CREATE TYPE Text ( T CLOB );
CREATE TYPE Picture ( P ROW ( R INT, G INT, B INT ) ARRAY );
CREATE TYPE Sound ( … )
CREATE TYPE Movie ( Vid PICTURE ARRAY, Aud SOUND );
CREATE TYPE Map ( … ) ;

CREATE TABLE Bird ( Birdname VARCHAR(30),
                    Info Text,
                    Pic Picture,
                    Video Movie,
                    Song Sound,
                    Migr Map,

                    PRIMARY KEY Birdname ) ;
```

All fine and reasonable so far. BUT …

56

The New Fatal Flaw

Incorrect (and not only supported, but encouraged by the vendors):

```
CREATE TYPE BirdStuff ( Birdname VARCHAR,
                        Info Text,
                        Pic Picture,
                        Video Movie,
                        Song Sound,
                        Migr Map ) ;

CREATE TABLE Bird OF BirdStuff
                    ( Info NOT NULL,
                      Pic NOT NULL,
                      [etc.]
                      REF IS Birdref SYSTEM GENERATED ) ;
```

Relationland's careful and fundamental distinction between types and relations is muddied ("The First Great Blunder")

Banishment of pointers was one of Codd's two main motivations for the relational model. The REF bit brings them back ("The Second Great Blunder")

How could such a stupid mistake have arisen?

57

Possible Reasons for The "Great Blunders"

Object class definitions have components called attributes.

Hence the wrong equation, class extent = relation variable
instead of the right equation, class = type (= Codd's "domain")

Objects have object identifiers (= SQL's REF values)

But oids are pointers (because they point). There are no pointers in Relationland (except possibly under the covers, where they can't be seen).

58

Object DB Structure

variable   variable

name   name

Object

oid

"state" (i.e., a value)

name   name

variable   variable

NOT WHAT WE WANT !

(soon leads to spaghetti)

59

Relational DB Structure

relation variable   relation variable

name   name   …

value   value
(a relation)

What we want instead!

(cannot make spaghetti)

60

Terminological Rapprochement

Objectland	Relationland
Class	Domain (now Type)
Object	(Variable)
Object identifier	none (but we have keys)
Method	Function, Procedure
Message	Operator invocation
Inheritance	Inheritance (thanks?)
Polymorphism	Polymorphism (thanks?)
Distinguished parameter	none (& don't want!)
none, and really wanted!	Relation

61

CLOSING REMARKS

62

The Relationlander's Promise

I solemnly promise …

… *never* to use the word "relational" when I mean SQL, …

… cross my heart and hope to die.

63

The Dream Database Language

1. *faithfully embraces the Relational Model of Data.*
   - *NO EXTENSION*
   - *NO PERVERSIONS*
   - *NO SUBSUMPTIONS*
2. *supports user-defined types and user-defined operators of arbitrary complexity*
3. (*allows SQL to be implemented in it for temporary use (until SQL finally expires)*)
4. *provides unprecedented chivalry*
5. *is generically referred to by the name ....*

64

D

65

Projects based on *TTM*

Rel: an implementation of **Tutorial D**
by Dave Voorhis, University of Derby

D4: a commercial implementation
by Alphora, Utah, USA
front-end to SQL (so includes nulls, alas)

Duro: a C API for relational algebra
and transactions, by René Hartmann

Further info at www.thethirdmanifesto.com

66

Some Guiding Principles

Some important principles that we have become particularly conscious of, for various reasons.

Some have always been with us.

Some arise from a retrospective look at our manifesto.

Some may even be said to have informed our manifesto.

67

Logical Differences

Principle #1
(our motto)

"All logical differences are big differences"
(Wittgenstein)

So all logical mistakes are big ones!

And we think all non-logical differences are small ones. In the database context, at least.

68

Values and Variables

Principle #2

"We retain and embrace a clear distinction between values and variables"

(Object Orientation seems to have blurred this distinction.)

69

Data Types and Relations

Principle #3

Data types and the relational model are orthogonal to each other.

Corollary :

The relational model has no jurisdiction concerning which data types a relational system should support.

(except we *must* have relation types, and BOOLEAN!)

70

Types are Not Tables

Principle #4

Types are to tables as nouns are to sentences!

So we cannot accept the equation
object class = relation
that some ORDBMSs (and SQL!) attempted to embrace.

"object class = domain" works fine.

71

Domains as Predicates

Questioning Principle #4 some have asked :

"But aren't domains predicates, too?" meaning "aren't they therefore relations, too?"

Well, yes - E.g. "*i* is an integer"

But in that case, what is the domain of *i* ?

72

Model and Implementation

Principle #5

We retain a strong, clear distinction between model and implementation.

So, we will not define our abstract machine in terms of what the system "really does".

73

A Database is Not a Model

Corollary

A database is an account of some enterprise, not a model of it.

In a relational database, the account is in the form of tuples, each of which is to be interpreted as some statement of *belief*. Under this interpretation, the system is able to derive certain other, non-stated beliefs when asked to do so.

74

Conceptual Integrity

Principle #6

"Conceptual integrity is *the* most important property of a software product"

(Fred Brooks, 1975)

Of course, you must *have* concepts before you can be true to any. These had better be:

a. few

b. agreeable to those invited to share them

75

Reims Cathedral

76

Conceptual Integrity

Principle #6 (bis)

"This above all: to thine own self be true,
And it must follow, as the night the day,
Thou canst not then be false to any user."

(from Polonius's advice to D, by WS with HD)

77

The End

78