

The Askew Wall

(background to *The Third Manifesto*)

Hugh Darwen

HD@TheThirdManifesto.com

Last updated: 03 December 2003

Terminological Equivalences

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

Cuddly term means the same as *Posh term* (and vice versa).

The Perversity of SQL

```
SELECT CITY_NAME  
FROM CITY C1  
WHERE 4 > (SELECT COUNT(*)  
           FROM CITY C2  
           WHERE C1.POPULATION < C2.POPULATION)
```

The Unperversified Version

```
SELECT CITY_NAME  
FROM CITY C1  
WHERE (SELECT COUNT(*)  
       FROM CITY C2  
       WHERE C2.POPULATION > C1.POPULATION) < 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

Foundation for Future Database Systems :
The Third Manifesto
by C.J. Date and Hugh Darwen

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

A Brief History of Data

1960: Punched cards and magnetic tapes

1965: Disks and 'direct access'

1969: 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!)

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.

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.
- Initiated the Dark Ages.

People even think the Wall is Relationland.

There have even been moves back to the Higgledy-Piggledy Model of Data! (Object Oriented Databases)

The Good Things SQL 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.
- FROM clause restricted to named tables.
- Duplicate column names.
- Duplicate rows.
- Nulls.
- Failure to support degenerate cases (e.g. columnless tables).
- Updating views WITHOUT CHECK OPTION.
- Failure to support “=” properly.
- and more to come if we are not careful.

A Thematic Query Example

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

Given :

EXAM_MARKS

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

To derive:

Student	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    EXAM_MARKS
GROUP BY SUBJECT
```

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

This is a CORRECTABLE flaw (well, NEARLY correctable).

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      STUDENT, E.SUBJECT, E.MARK, S.??? -- unnamed column
FROM        EXAM MARKS E,
            (SELECT      SUBJECT, AVG(MARK)
             FROM        EXAM MARKS
             GROUP BY   SUBJECT) S
WHERE       E.SUBJECT = S.SUBJECT
```

This is a correctable flaw. It is not generally bypassable, though sometimes you can create a named view for the “nested” query.

Note that, while lack of support for this is a fundamental error, the suggested fix leads to queries that are difficult to write and difficult to understand. Needless so!

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

Duplicate column names

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

Example 5:

```
SELECT      *
FROM        EXAM_MARKS E, STUDENT S
WHERE      E.ST_NUM = S.ST_NUM
```

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

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

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      STUDENT, E.SUBJECT,E.MARK,S.AVG  
FROM        EXAM_MARKS AS E,  
            (SELECT  SUBJECT,AVG(MARK) AS AVG  
             FROM    EXAM_MARKS  
             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.

Perhaps better if broken down into more digestible steps, using the new WITH feature (SQL:1999).

With WITH

Example 4 (fixed):

```
WITH AVG_MARKS AS  
  (SELECT  SUBJECT, AVG(MARK) AS AVG  
   FROM    EXAM_MARKS  
   GROUP BY SUBJECT)  
  
SELECT STUDENT, E.SUBJECT, E.MARK, A.AVG  
FROM    EXAM_MARKS AS E, AVG_MARKS AS A  
WHERE  E.SUBJECT = A.SUBJECT
```

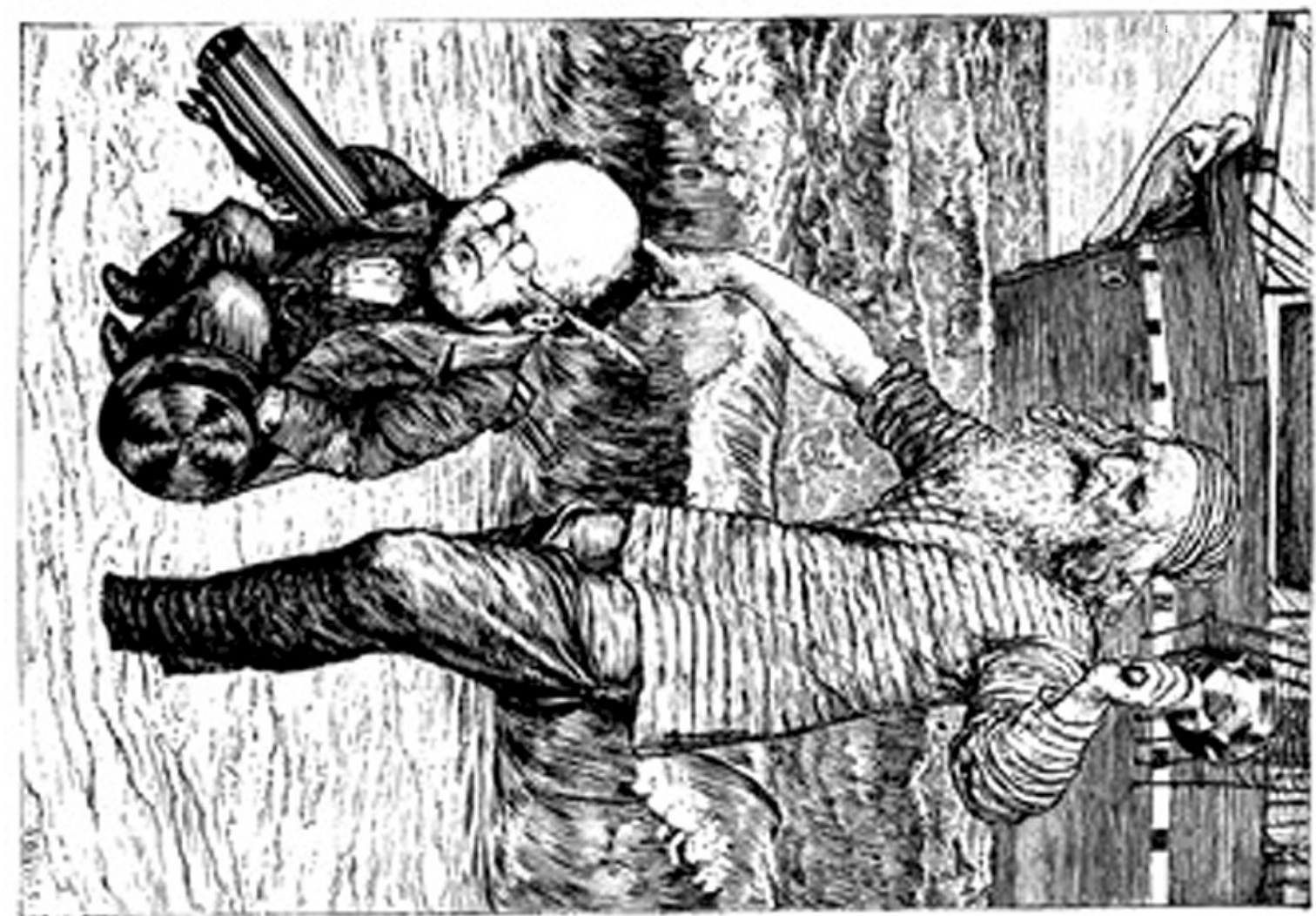
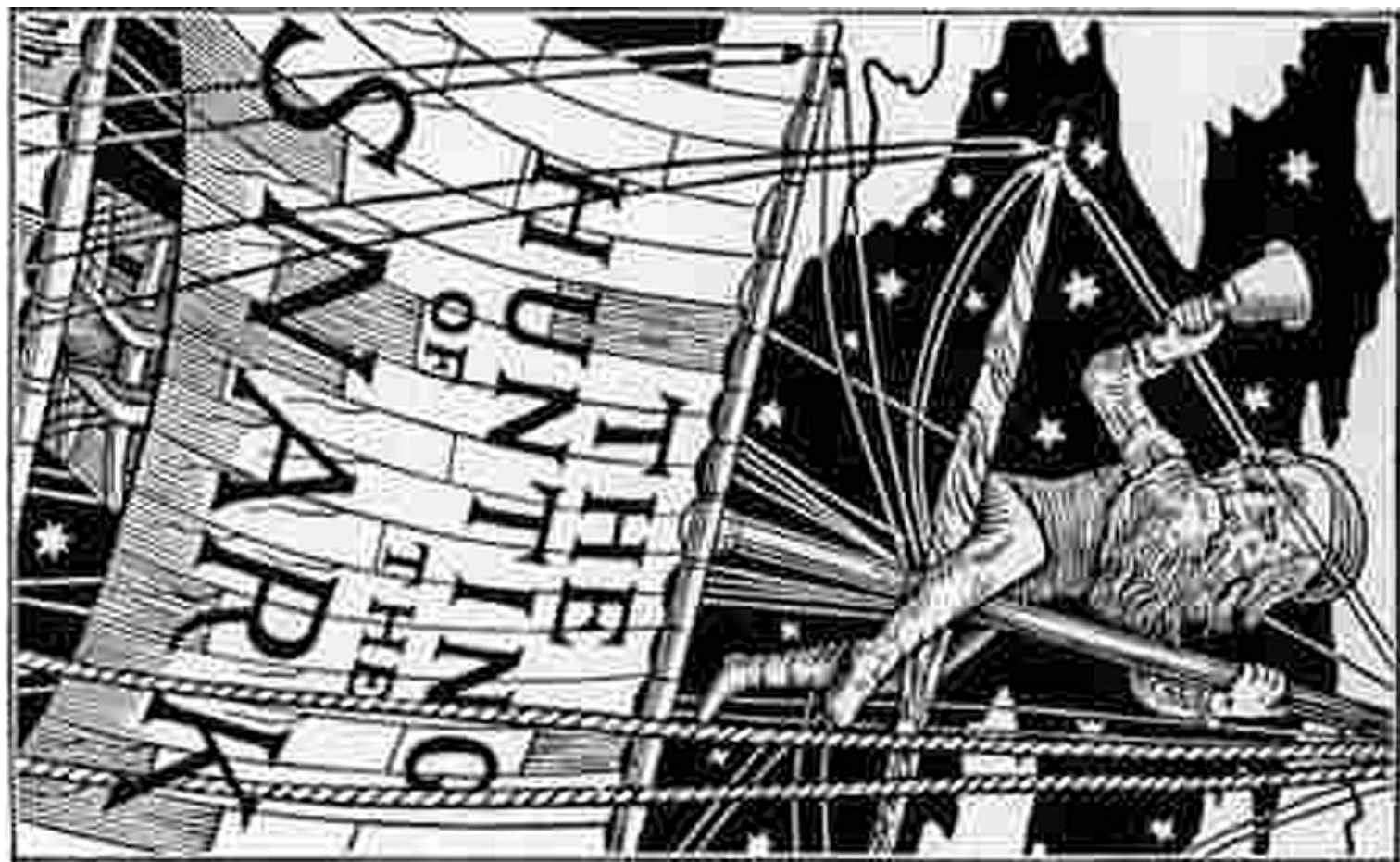
WITH is an optional conformance feature in SQL:2003. Not many implementations have it, and even those that do have it do so only with unpleasant restrictions.

Duplicate Column Names (again)

So now you can even do this :

```
SELECT COL1 AS X, COL2 AS X  
FROM T
```

Enjoy!



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 twice doesn’t make it any truer”

(E.F. Codd, approximate quotation)

This is a bypassable flaw :

- 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.

Nulls

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 more debate and anguish than any other Fatal Flaw.

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?

Why nulls ruin everything –

- UNION of sets, cardinality of sets.

Destruction of functional dependency theory

SQL's implementation of nulls 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**.

A Contradiction Caused by NULLS

“Every relation has at least one candidate key”

“One of the candidate keys is nominated to be the primary key”

“Nulls aren’t permitted in the primary key”

“Nulls *are* permitted in alternate keys”

- Consider the projection of STUDENT over RELIGION, a nullable column.
- List the candidate keys of this relation.
- Nominate the primary key.

Failure to Support Nothing

Ref: “Table_Deer and Table_Dumb, chapter 22 in RDBW 85-89, and “The Nullologist in Relationland, or Nothing *Really* Matters”, chapter 13 in RDBW 89-91

SQL doesn't know (much) about the EMPTY SET !

- * 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.

Bypasses for Failure to Support Nothing

Example 6:

“Did any student obtain more than 75 marks in Database Theory ?”

```
SELECT          DISTINCT 'YES!'  
FROM           EXAM_MARKS  
WHERE          MARK > 75 AND SUBJ = 'Database Theory'
```

Example 7:

“What’s the time?”

```
SELECT          DISTINCT CURRENT_TIME  
FROM           STUDENT
```

“=” 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 $\text{Length}('A') < \text{Length}('A ')$

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

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 and "hidden" surrogate keys.

Nothing wrong with systematic timestamps.
Nothing wrong with system-generated keys.

Why hide them?

Why The Flaws Are “Fatal”

- ❖ The Shackle of Compatibility
- ❖ The Growth of Redundancy
- ❖ Desired extensions can be difficult or impossible to specify (because of nulls, e.g.)
- ❖ Soundness and Elegance don't belong in SQL

Errors Here to Stay

- * Duplicate Rows
 - * Nulls
 - * **SELECT-FROM-WHERE**
 - * **WITHOUT CHECK OPTION**
 - * scalar subqueries
- (and probably many others)

Why Duplicate Rows Hurt

For example:

Enhanced view updatability in SQL3 -
(e.g., updatable joins)

requires mapping from rows in view to
rows in database.

Impossible with duplicates in the
database!

Why Nulls Hurt Even More

Suppose “X=X” returns “unknown”

Can we safely conclude “X IS NULL” ?

Not in modern SQL!

How $X=X$ *Unknown* Yet X NOT NULL

For example:

1. X is ROW (1, null)
2. X is POINT (1,null)
3. X is ROW (POINT(1,1), POINT(null,3))

ROW(...) is a row “constructor”.

POINT(a,b) is a “constructor” for values in the user-defined data type POINT.

Consequences?

SELECT-FROM-WHERE

```
SELECT title,  
        CONTAINS_SCORE(text,'Prince')
```

```
FROM DOCUMENTS
```

```
WHERE
```

```
        CONTAINS_SCORE(text,'Prince') > 50
```

See how SQL's quaint syntax enforces undesirable repetition? - and this is only a very simple example!

At last this error is starting to hurt.

How to Extend, then Restrict

In SQL:2003, with user-defined functions:

```
SELECT title, score FROM
  (SELECT T.*,
    CONTAINS_SCORE(text,'Prince')
    AS score
  FROM DOCUMENTS T) AS DUMMY
WHERE score > 50
```

In relational algebra:

```
EXTEND DOCUMENTS ADD
  CONTAINS_SCORE(text,'Prince') AS score
WHERE score > 50 {title,score}
```


Scalar Subqueries

We wish to support “nested tables” in SQL, but we are thwarted by ill-advised syntax in SQL:1992.

```
SELECT DNO, ( SELECT ENO  
              FROM EMP E WHERE  
              E.DNO=D.DNO ) AS EMPS  
FROM DEPT D
```

Scalar subquery or nested table ?

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

Why Subqueries are Redundant

E.g.:

```
SELECT *  
FROM EMP  
WHERE HIRE_DATE =  
(SELECT MIN(HIRE_DATE) FROM EMP )
```

is the same as

```
SELECT *  
FROM EMP NATURAL JOIN  
(SELECT MIN(HIRE_DATE) AS HIRE_DATE  
FROM EMP) AS POINTLESS_NAME
```

Why “Correlation Names” Are Redundant

“Correlation names” were needed in old SQL pre 1992 to avoid ambiguity when two or more columns have the same column name.

But now SQL supports column renaming in the `SELECT` clause.

This even solves the “`SELECT X, X`” problem!

`(SELECT X AS X1, X AS X2 ...)`

Why Longhand Joins are Redundant

E.g.:

```
SELECT *  
FROM EMP E, DEPT D  
WHERE E.DEPTNO = D.DEPTNO
```

= EMP NATURAL JOIN DEPT

= EMP JOIN DEPT USING(DEPTNO)

Why HAVING is Redundant

E.g.:

```
SELECT DEPTNO,  
       AVG(SAL) AS AVG_SAL  
FROM EMP  
GROUP BY DEPTNO  
HAVING AVG(SAL) >999
```

=

```
SELECT * FROM  
  ( SELECT DEPTNO, AVG(SAL)  
    AS AVG_SAL  
  FROM EMP  
  GROUP BY DEPTNO ) DUMMY  
WHERE AVG_SAL > 999
```

Why GROUP BY is Redundant

E.g.:

```
SELECT DEPTNO,  
       AVG(SAL) AS AVG_SAL  
       MAX(SAL) AS MAX_SAL  
FROM EMP  
GROUP BY DEPTNO
```

is better done by (why “better”?):

```
SELECT DEPTNO  
       (SELECT AVG(SAL) AS AVG_SAL,  
        MAX(SAL) AS MAX_SAL,  
        FROM EMP  
        WHERE E.DEPTNO = D.DEPTNO)  
FROM DEPT D
```

(This uses a “row subquery”.)

The Relationlander's Promise

I solemnly promise ...

... *never* to use the word “relational” when I mean SQL, ...

... cross my heart and hope to die.

Object Oriented Databases

- * Some good database motherhood
- * Some good ideas (at least one)
- * Some bad ideas (e.g., objects)
- * Lack of commonly agreed model
- * Failure to embrace relations

But potentially well poised, if only...

Rapprochement

A bringing together of objects and relations

Widely sought, because:

- * Some Objectlanders want to be able to do what Relationlanders do with tables - specially ad hoc queries and declarative constraints.
- * Some Relationlanders want to do some more complicated things that require user-defined data types of arbitrary complexity.

First Normal Form

Ref: "Relation-Valued Attributes, or Will the Real First Normal Form Please Stand Up?", in RDBW 89-91.

Absolutely fundamental.

Terribly misunderstood.

Relationland's "Great Encapsulator".

1NF – a Shakespearean Example

<1> KILLED

Killer	Victim
Brutus	Caesar
Brutus	Brutus
Hamlet	Laertes
Hamlet	Polonius
Laertes	Hamlet
Cassius	Caesar

Note : exactly ONE VALUE at each row/column intersection

Predicate: “Killer, a character in Shakespeare, killed Victim, a character in Shakespeare.”

*Attribute names, KILLER and VICTIM, represent the **place-holders** of the predicate.*

*The relation name KILLED is the **verb** of the predicate.*

*“Character in Shakespeare” is the **domain** of KILLER and of VICTIM.*

This domain is “supported” by the Relational Model of Data, as is every domain you can possibly imagine!

WALK OUT OF any lecture that tells you that the Relational Model supports only certain domains, such as numbers, character strings, dates and times.

Not in 1NF

<2> KILLED

Killer	Victim
Brutus	Caesar Brutus
Hamlet	Laertes Polonius
Laertes	Hamlet
Cassius	Caesar

“Caesar Brutus” IS NOT EXACTLY ONE VALUE.

But this isn't even a relation!

In 1NF Again

<3> KILLED

Killer	Victim
Brutus	{ Caesar, Brutus }
Hamlet	{ Laertes, Polonius }
Laertes	{ Hamlet }
Cassius	{ Caesar }
Polonius	{ }

*Domain of Victim is now
“set of characters in Shakespeare”*

Getting from <1> to <3> needs a “grouping” operation.

Getting from <3> to <1> needs an “ungrouping” operation.

It seems that to be in 1NF is nothing more (or less) than to be a relation.

Why 1NF Encapsulates

An atomic value: “cannot be decomposed into smaller pieces by the DBMS...” (E.F.Codd)

Note : “by the DBMS” (and wonder what that means)

*Relational operators work with relations.
They don't know anything about the domains!*

Expressions such as KILLER = 'Brutus' are qualifiers for relational operators, and their evaluation belongs with domains, not the relational operators.

A Note on Atomicity

*Codd's motivation : Simplicity
to make corporate databases readily approachable
by a large and diverse community of users.*

*But some things just are complex
no getting away from it.*

Besides, atoms are notoriously splittable after all.

*Encapsulation
helps us to get to grips with complexity.*

Encapsulation makes molecules behave like atoms.

Embracing Complexity

Pictures of buildings :

BUILDING

Name	Picture
Reims Cathedral	< picture >
Durham Ox	< picture >
etc.	

Predicate : “Picture is a picture of building Name”

Domain of Name is “name of building”

Domain of Picture is “picture of building”

But with “Atomic” Domains Only

Pictures of buildings :
















BUILDING

Name	Pixel#	Colour	Brightness
Reims Cathedral	42	red	very
Reims Cathedral	9	blue	a little
Durham Ox	33	brown	maximally
Reims Cathedral	1	white	somewhat
etc.			

Predicate :

“Pixel number Pixel# of the picture of building Name is coloured Colour, Brightness brightly.”

A Multimediu Database

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.

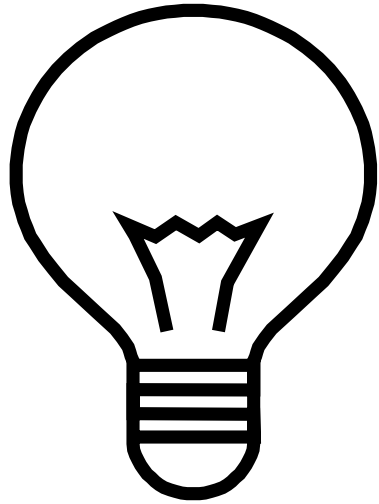
A New Fatal Flaw Has Loomed

Some attempts at rapprochement treat rows as objects, and sets of rows as sets of objects.

This approach is doomed

The relational equivalent of object class is DOMAIN, not relation !

A Guiding Light



All logical
differences are
big differences
(Wittgenstein)

All logical mistakes are big mistakes
(Darwen's corollary)

All non-logical (psychological) differences are
small differences
(Darwen's conjecture)

The Dream Database Language

1 will faithfully embrace the Relational Model of Data.

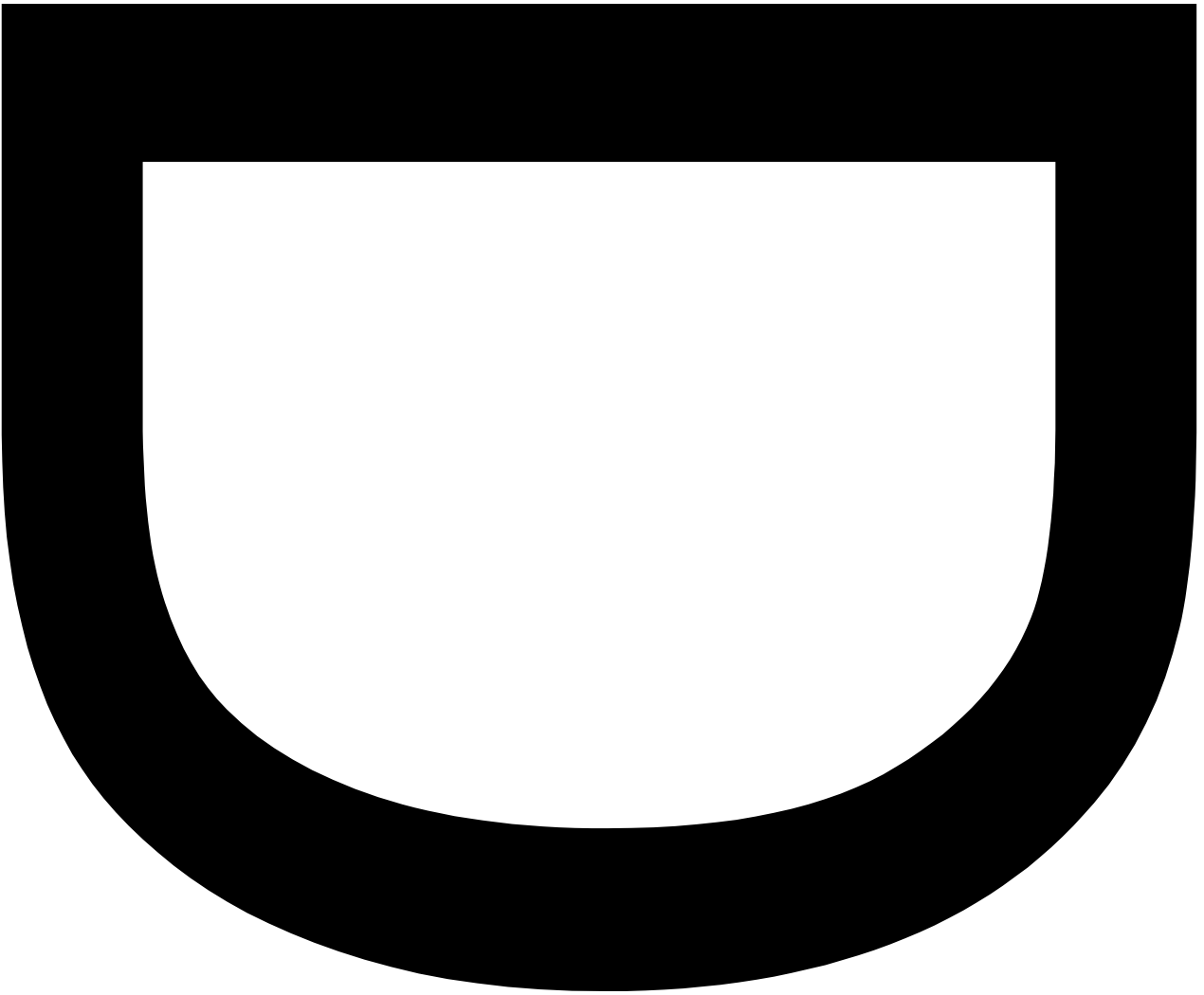
- NO EXTENSION*
- NO PERVERSIONS*
- NO SUBSUMPTIONS*

2 - will support user-defined domains and user-defined functions of arbitrary complexity.

(3 will allow SQL to be implemented in it for temporary use (until SQL finally expires).)

4 will provide unprecedented chivalry.

5 will be named....



*The Dream is
coming true!*

Visit <http://www.alphora.com>

*to find about Alphora's "Dataphor" and database
language D4*

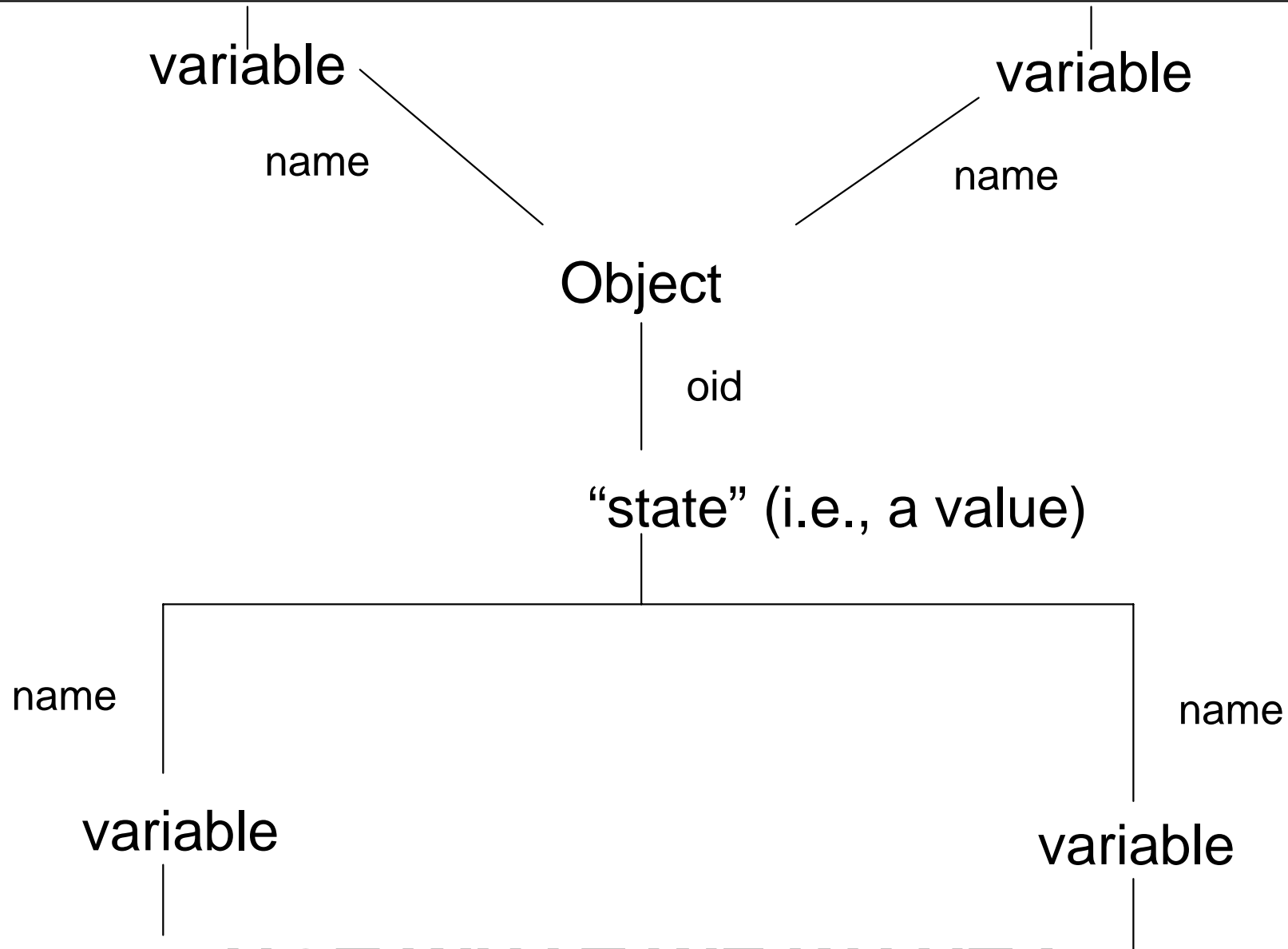
*D4 is a faithful implementation of The Third Manifesto
and therefore possibly the first commercially available
implementation of E.F. Codd's*

Relational Model of Data (1970)

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

Object DB Structure



NOT WHAT WE WANT !

(soon leads to spaghetti)

Relational DB Structure

relation variable

relation variable

name

name

...

value

value

(a relation)

What we want instead!

(cannot make spaghetti)

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.

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.

Values and Variables

Principle #2

“We retain and embrace a clear distinction between values and variables”

(Object Orientation seems to have blurred this distinction.)

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.

We reject absolute atomicity in favour of our clarified definition of 1NF.

Absolute Identity

Principle #4

There's no such thing as absolute identity.

Identity means understanding and agreeing some sense in which “this is the same value as that” can be interpreted. Belonging to the same data type is that sense.

So values don't need to carry “oids” around with them and, in fact, they don't !

Types are Not Tables

Principle #5

Types are to tables as nouns are to sentences!

So we can't accept the equation "object class = relation" that some ORDBMSs are attempting to embrace.

"object class = domain" works fine.

Domains as Predicates

Questioning Principle #5 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 ?

Model and Implementation

Principle #6

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”.

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.

Conceptual Integrity

Principle #7

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

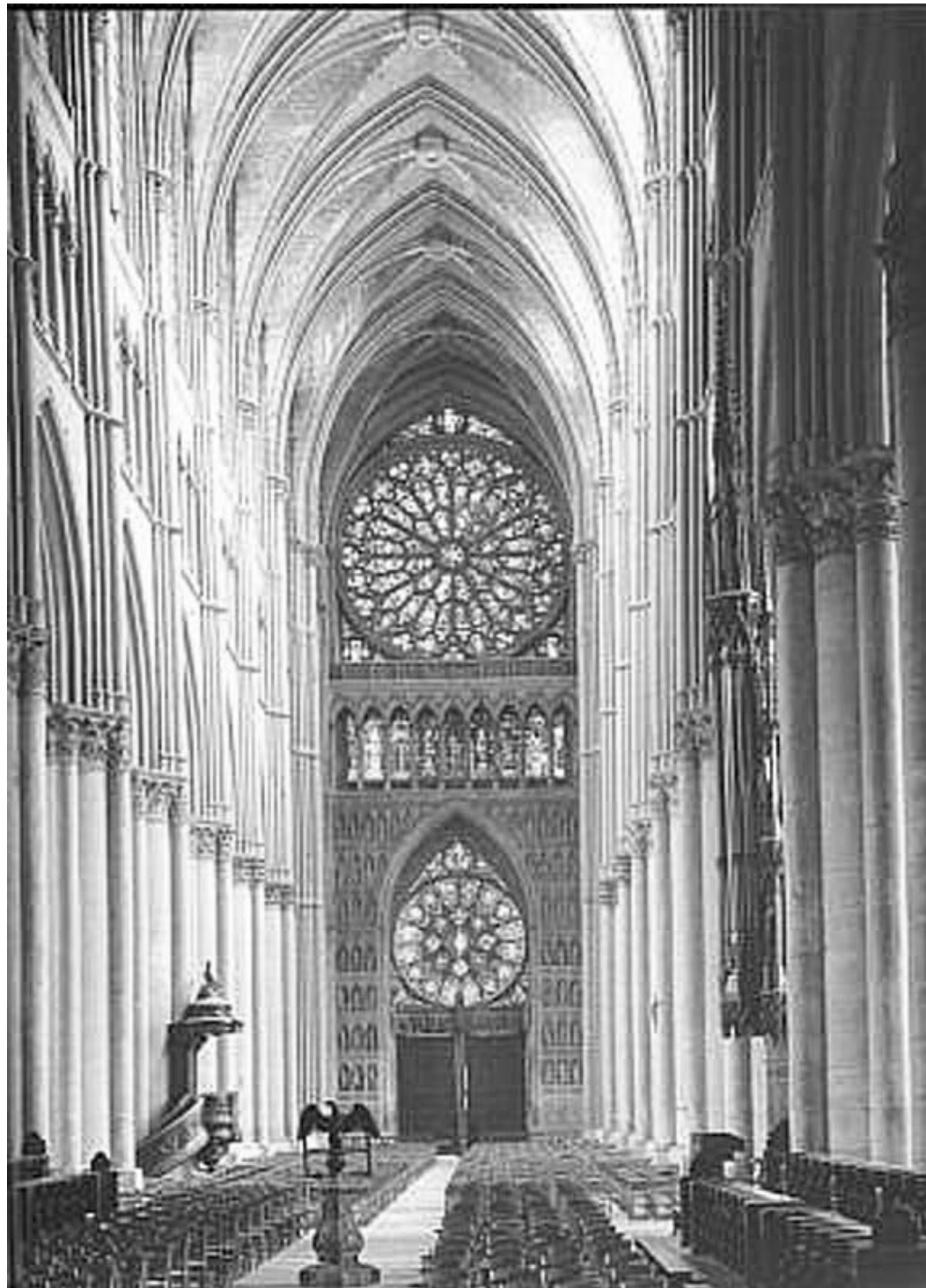
(Fred Brooks, 1975)

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

Reims Cathedral



Conceptual Integrity

Principle #7 (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)

The End