

# A critical reading of the Third Manifesto

Maurice Gittens <maurice at gittens dot nl>

November 15, 2005

## Abstract

According to the authors, Hugh Darwen and C.J. Date, of the book entitled "Foundation for Future Database Systems; The Third Manifesto" the maxim: *All logical difference are big differences* and its corollary *All logical mistakes are big mistakes* has been central to their work on this book. Respecting the standard set by this maxim and its corollary, this paper will proceed to identify a number of issues with the logical consistency of the dissertation presented in The Third Manifesto, using maxims such as: *logical conclusions should only be drawn from premises which are both valid and relevant.*

The copyright of this document belongs to its author. Making complete and unmodified copies of this document is allowed.

**Status: draft**

## Revision history

- Nov 15 2005; Fix an unfortunate misspelling of Hugh Darwen's name. For this I sincerely apologize.
- July 14 2003; Fix typo in the title of the document
- April 7 2003; More cleanups
- February 26 2003; Based on comments by Hugh Darwen I reworded a few sentences which seemed to cause confusion; I also fixed a few typographical errors
- January 8 2003; Rene Jansen made me aware of another reason for the dismissal of ObjectIDs provided by The Third Manifesto. Add this to the section about the alleged second great blunder. Thanks Rene.
- January 6 2003; A first draft of this document

## Contents

<b>1 Introduction</b>	<b>2</b>
1.1 Background information . . . . .	2
1.2 On a personal note . . . . .	3
<b>2 About the claims made by the author</b>	<b>3</b>
2.1 Regarding the first great blunder . . . . .	3
2.2 Regarding the second great blunder . . . . .	4
<b>3 About Predicates, Relations and their identity</b>	<b>6</b>
3.1 Some examples . . . . .	6
3.2 Introduction to predicate logical models . . . . .	7
3.3 Why is identity deemed a necessity? . . . . .	8
3.4 Summary . . . . .	9
<b>4 On the expressive equivalence of relation values and tuple values</b>	<b>10</b>
4.1 Introduction . . . . .	10
4.2 Defining tuple values . . . . .	11
4.3 Defining relation values . . . . .	11
4.4 Showing that all relation values are tuple values . . . . .	11
4.5 Showing that all tuple values are relation values . . . . .	12
4.6 Summary . . . . .	12
<b>5 Conclusions</b>	<b>12</b>

## 1 Introduction

### 1.1 Background information

A web page at <http://www.gittens.nl/OOR.html> raised a number of issues the author found with the logical consistency of the dissertation presented in the second edition of the book “Foundation for Future Database Systems”<sup>1</sup> by C.J. Date and Hugh Darwen. In a personal communication Mr. Hugh Darwen, requested I clarify my use of certain English words and also that I be more specific as to the issues I found with the dissertation presented in The Third Manifesto. This paper is written as an attempt to comply with the request of Mr. Darwen.

### The main issue and my primary claim

The main issue I perceive, with the logical consistency of the dissertation presented in the Third Manifesto, follows from the maxim Date and Darwen presented as central to their work in The Third Manifesto. Date and Darwen presented the maxim: *All logical difference are big differences* and its corollary *All logical mistakes are big mistakes* as a

<sup>1</sup>This paper will refer to this book as “The Third Manifesto”.

guiding principle in their work on the Third Manifesto. The Third Manifesto proceeded to identify what it refers to as the Two Great Blunders :

- Equating relvars and classes
- Mixing pointers and relations (or more specifically allowing database relvars to contain object IDs)

However, in my humble opinion, the argumentation used to substantiate the claim that the alleged blunders are truly to be viewed as blunders is somewhat weak.<sup>2</sup> This *opinion* is based on the following maxim: *logical conclusions should only be drawn from premises which are both logically valid<sup>3</sup> and relevant*. Put another way, keeping in mind the maxim, *All logical differences are big differences*, logically valid conclusions are conclusions based on premises free of fallacies, including fallacies of relevance.<sup>4</sup>

I think it important to state explicitly, that my claim in this regard, is not that Date and Darwen are *wrong* in their opinions. My claim, in this regard, is that the *substantiation* they provide as justification for their statement that the alleged great blunders are indeed great blunders is rather weak, relative to the high standard they claim to be central to their work.

## Secondary issues and my secondary claims

In this regard the question is asked whether or not the alleged two great blunders are indeed blunders. This subject matter will be addressed in subsequent sections of this paper.

### 1.2 On a personal note

I think it appropriate to state my appreciation for the fact that Mr. Hugh Darwen has thought it appropriate to spend time communicating with me about the issues I raised.

## 2 About the claims made by the author

### 2.1 Regarding the first great blunder

The first alleged *great blunder* identified in The Third Manifesto follows:

*Equating relvars and classes*

Now please consider the question: *What arguments that adhere to the strict discipline of logic does The Third Manifesto provide for the claim that this equation is indeed a blunder?* In considering an answer to this question it is noted that this claim<sup>5</sup> is

---

<sup>2</sup>I invite the reader to verify for herself or himself, that The Third Manifesto provides *no* logically valid arguments justify the labeling the presented propositions as “*blunders*”.

<sup>3</sup>A logically valid premise is one that is substantiated in terms of mathematical logic.

<sup>4</sup>or layering violations, if you prefer.

<sup>5</sup>Also much of the argumentation which supports this claim is in this section of the book

first made on page 15 of second edition of the Third Manifesto, which by its own admission (on page 14) is informal in nature. Still lacking a mathematically sound definition of what an *object class* is, page 21 decides that object classes and domains are, I quote: “*the same thing*”. Since the statement is made in an informal context, one wonders if classes and domains are informally the same thing or also formally<sup>6</sup> the same thing. The reason given to substantiate the claim that object classes and domains are determined to be the same thing is presented as the fact that for both, domains and object classes, it holds that their values are manipulated by operators defined for the type in question. However, the same argument can be made for relations. *Is it not the case that relations values are manipulated by a set of operators defined specifically for their types*. Yes, relation types have a set of pre-defined operators, does this make them logically different from a specific class of domains which have a pre designated set of operators? No, it does not! OK, since this argumentation is said to be informal, the author proceeded to seek out, the formal arguments presented, for labeling the first great blunder as such. Apart from reiterations of the alleged first great blunder, no such argument has currently been found by the author<sup>7</sup>. Additionally, Date and Darwen, seem to assume<sup>8</sup> that there is one so-called *right way*, in which objects and relations should be integrated. Does not the discipline of logic dictate that one must prove, that there exists only one right way, before one could even claim to provide *the* one right way? Is it not possible that there are different ways, each with its own merit, to achieve the integration between objects and relations?

## 2.2 Regarding the second great blunder

The Third Manifesto identifies the second great blunder as:

*Mixing pointers and relations (or more specifically allowing database relations to contain object IDs).*

Using the index of The Third Manifesto I have found the following reasons why object IDs or references<sup>9</sup> are unwanted by Date and Darwen.

1. Codd’s information principle: *All information in the database at any time must be cast explicitly in terms of values in relations and in no other way or All inter-relating between different parts of a database must be achieved by comparison of value.*
2. The reason Codd removed pointers from the relational model is stated as: *It is safe to assume that all kinds of users [including end users in particular] understand the act of comparing values, but that relatively few understand the complexities of pointers [including the complexities of referencing and dereferencing*

---

<sup>6</sup>as in mathematically equivalent abstractions

<sup>7</sup>I will gracefully, acknowledge being in error if such arguments do exist.

<sup>8</sup>on page 14 of the second edition of the Third Manifesto

<sup>9</sup>There seems to be some confusion that object IDs, references and pointers represent one and the same thing. It should for example, be recognized that identity is a property of every element of a mathematical set. Confusing the identity of an object and the notion of a pointer is a logical error

*in particular]. The relational model is based on this fundamental principle... [The] manipulation of pointers is more bug-prone than is the act of comparing values, even if the user happens to understand the complexities of pointers.*

3. On page 417 of the second edition, the paragraph entitled : “OBJECT IDS UNDERMINE INHERITANCE “

Concerning the first two points I ask the question: *Of what logical value are these arguments?*<sup>10</sup>The fact that Codd rejects pointers on the grounds that they are "difficult to understand" and "bug-prone" is of no logical value, and as such in the context of providing a justification for the so-called *second great blunder*, these arguments represent a fallacy of relevance. This is not to say that the statement is not true by some measure. It is only to say that such statements do not provide logically valid grounds for the dismissal of object-IDs.

Concerning the third point, the following issues:

- First, a fallacy is exposed by this quote from the page referenced: “*Pointers can lead to a serious problem if type inheritance is also supported*”. This provides the in-site that Date and Darwen confuse object identity and pointers.
- Second, they proceed to describe a problem with object identity and *their* inheritance model. It is a fallacy to assume that this problem would exist with some other inheritance model.

These two points identify the third reason supplied for the dismissal of ObjectIDs as a fallacy.

## **More on identity**

Please consider the position of Hugh Darwen on identity as it was presented in a personal communication [ref 3].

We do not recognize any concept of identity of a value  $v$  other than  $v$  itself. A truth-valued expression of the form  $x = y$  is true if and only if the values denoted by the expressions  $x$  and  $y$  are identical are in fact one and the same value. Given equality, we do not need any other concept to do with distinction of values. In case the distinction you are referring to is the one found in some OO programming languages, I remark that in such languages equality is as in our definition (though "=" is sometimes sacrificed, with unpleasant consequences, in favor of an operator with the same name but meaning "approximately equal to"), whereas what you call identity is equality of pointers (usually called object identifiers), and a pointer points to a variable, not a value. As you note in your very next section, we do not admit pointers.

---

<sup>10</sup>The reference to *logical value of an argument* refers to the degree in which an argument can be used to draw logically valid conclusions

Identity is a fundamental property of all things by which they can be counted. If elements of mathematical sets did not have identity they would be not be countable<sup>11</sup>. To put this another way, *Identity can be viewed as a property of an element of a set*. Equality on the other hand, is a *correspondence* between two or more elements of a set. For example, if  $\{v_1, v_2, \dots, v_n\}$  represents a set of relation variables. Each variable in this set has a distinct identity, otherwise it would not be possible to distinguish it from other variables in this example set. The identity of these variables is orthogonal to the issue of whether or not some of these relation variables are equal or not. And for the sake of completeness I wish to state that it should be evident that the fundamental concept of *identity* has nothing to do with the concept of *pointers* as they are known in different programming languages<sup>12</sup>.

### 3 About Predicates, Relations and their identity

This section presents what is in the opinion of the author a logically sound motivation for the support of identity in future databases. To comply with Hugh Darwen's request for specific examples I start with some examples.

#### 3.1 Some examples

Consider some example functionality. Let  $rv_1, rv_2, \dots, rv_k$  be relation variables of the same relation type. Let  $v_1, v_2, \dots, v_k$  be the relation values of  $rv_1, rv_2, \dots, rv_k$ . I would like to be able to ask the equivalent of following questions in the query language of the database: *Which set of relation variables has the value  $v_2$ . Or which relation variable has the greatest number of tuples with a particular property?* The database system would in turn respond with a *properly typed* set of identities corresponding with the result of the query. The catalog of common SQL-databases might be used for such purposes however, in current relational database systems the types of the objects returned would, as you know, be incorrect. This forces people working on business-repositories, data-mining applications etc, to build much logic into their applications which, in my opinion, should be gracefully handled by databases of the future. If the result of a query can be an entity representing a relation variable, or a type, or a tuple variable etc, the logical expressiveness<sup>13</sup> of the database is increased. If the information in the catalog of relational databases were properly typed much of the necessary machinery would be present in database systems.

---

<sup>11</sup>Similarly, symbols in mathematical strings also have identity. In the string "aaa" there are three instances of the symbol *a* each with their own identity. The fact that the symbols are all the same, is not relevant to their identity in this string.

<sup>12</sup>Of course, based on the identity of objects, pointers can distinguish between them. But this does not equate pointers to identity.

<sup>13</sup>In this regard a formalism  $f_1$  is said to be *more expressive* than a formalism  $f_2$  when the set of statements that can be represented using  $f_1$  is a super set of the statements that can be represented using  $f_2$ .

## A new operator

An operator which in my opinion is necessary, in one form or the other, in future database is the *foreach* operator. This would be the database counter part of the universal quantifier operator known from predicate logic. <sup>14</sup>Hopefully self-explanatory, informal<sup>15</sup> examples, using this operator in an SQL like language follow:

Example statement	Description
foreach relation $r$ select * from $r$ ;	select all tuples in the default schema
foreach relation $r$ in schema example_schema delete from $r$ ;	remove all tuples from relation variables in a schema
foreach relation $r$ in schema example_schema delete $r$ ;	drop all relations from a schema
foreach schema $s$ foreach relation $r$ in $s$ select * from $r$	select all tuples in the database
foreach relation $r$ select * from attributes( $r$ )	select the attributes of all relations
foreach relation $r$ where $r.someProperty() == true$ select * from $r$	select all attributes of relations with some property

It is important to note that the *type* of  $r$  in a statement like: *foreach relation  $r$  ...* is a *relation type*. The logical variable  $r$  is said to be bound to a predicate constant, representing the *identity* but not the propositional value of the predicate<sup>16</sup>.

Please note that supporting the *foreach* operator implies that , SQL statements like ALTER and DROP statements may be replaced by appropriate uses of UPDATE and DELETE statements. Thus showing, these and similar, statements to be redundant.

## 3.2 Introduction to predicate logical models

A model  $M$  for a first order predicate logical language  $L$  is a pair  $(D, I)$  such that :

- $D$  represents the domain of discourse of the model  $M$ . This is the set of objects which can be bound to variables in  $L$ . In relational systems, objects in the domain of discourse may be viewed as domain values. In relational systems, domains partition the domain of discourse into a set of disjoint subsets. Such that the union of the set of all domain values in a RM database is exactly equal to the set of objects in  $D$ .
- $I$  represents the interpretation function of the model  $M$ . Since  $I$  is a mathematical function it by definition has a domain and a co-domain, denoted  $dom(I)$  and  $codom(I)$  respectively.

In first order predicate logic each object  $d$  in the domain of discourse  $D$  has an associated constant  $c$  in the predicate logical language  $L$  which represents it in the language  $L$ . Using the interpretation function  $I$ , each predicate  $P$  of arity  $n$  in the predicate logical language  $L$  assigns the property represented by  $P$  to a set of  $n$  tuples  $\{t_1, \dots, t_k\}$  where each  $t_i$  ( $1 \leq i \leq k$ ) can be written as  $t_i = (d_1, \dots, d_n)$  where each  $d_j$  is an object in the domain of discourse  $D$ . As an example let us consider a model  $M$  for a predicate logical language  $L$  with constants  $\{a, b, c, Mark, Jane\}$  and predicates  $\{odd, love, miss, rich\}$ . In this example the domain of discourse  $D$  of  $M$  is  $D = \{1, 2, 3, "Mark", "Jane"\}$ ,

<sup>14</sup>In the context of databases I would suggest this operator be used for quantifying objects which are elements of the domain of predicate logical interpretation functions.

<sup>15</sup>and thus appealing to the goodwill of the reader

<sup>16</sup>The following section will elaborate, so that the distinction becomes clear

while an example interpretation function  $I$  for  $M$  is presented in the following table.

$dom(I)$	$codom(I)$
$a$	1
$b$	2
$c$	3
$Mark$	"Mark"
$Jane$	"Jane"
$love$	{("Jane", "Jane"), ("Jane", "Mark")}
$miss$	{("Mark", "Jane")}
$rich$	{"Jane", "Mark"}
$odd$	{1, 3}

This example represents statements like:

- *Jane loves both herself and Mark*
- *Mark misses Jane*
- *Jane and Mark are both rich*

The *identity* of the predicate *love* captured by the predicate constant *love* which, in the example above, appears in domain of the interpretation function  $I$ . The *propositional value* or the *value* of the predicate *love* in this example, is the set of tuples {("Jane", "Jane"), ("Jane", "Mark")}. When the Third Manifesto speaks of the relation value it is referring to the propositional value of a predicate. In this example *love* and *miss* are binary predicates, so the interpretation function  $I$  maps them to sets of binary tuples. The interpretation function  $I$  maps the constants in the language  $L$  to elements of  $D$  and unary predicates are mapped to subsets of  $D$ . The information contained in the interpretation function of a predicate logical model can be viewed as the predicate logical equivalent of a database. Relational algebra can thus be viewed as an algebra defining operations on a subset of the co-domain of interpretation functions of predicate logical models, more specifically *relational algebra defines a number of operations on the propositional value of predicates*.

### 3.3 Why is identity deemed a necessity?

The predicate logical language  $L$  in the previous section was based on the object constants  $\{a, b, c, Mark, Jane\}$  and the predicate constants  $\{odd, love, miss, rich\}$ . Now please notice that the *co-domain* of the interpretation function  $I$  in the previous example contains no appearances of either object constants or predicate constants. Put another way, there are no appearances of elements of  $dom(I)$  in  $codom(I)$ . The reason for this is quite simply that First Order predicate logic<sup>17</sup> does not allow object constants and predicate constants to be part of the domain of discourse  $D$ . As far as my understanding reaches, *Codd's information principle is, at least in spirit, referring to this fact*.

<sup>17</sup>The same is true for higher order predicate logic

**When value substitution is not enough** Now please consider a modified interpretation function as an extension of the previous example. This example will attempt to illustrate that by allowing so-called predicate constants to appear in the co-domain of the interpretation function, more sophisticated logical statements can be made<sup>18</sup>.

$dom(I)$	$codom(I)$
$a$	1
$b$	2
$c$	3
$Mark$	"Mark"
$Jane$	"Jane"
$love$	$\{("Jane", miss), ("Mark", love)\}$
$miss$	$\{("Mark", "Jane")\}$
$rich$	$\{("Jane", "Mark")\}$
$odd$	$\{1, 3\}$

In this example the predicate *love* is used to make the statement that *Mark loves to love* and also the statement *Jane loves to miss*. Notice that it would be incorrect to substitute  $\{("Mark", "Jane")\}$ , which is the propositional value of the *miss* relation, for the predicate constant *miss* in this example? Such a substitution would represent the claim *Jane loves the set*  $\{("Mark", "Jane")\}$ , which is clearly a different statement than the statement *Jane loves to miss*.

Of course one could argue that such expressiveness is not necessary. This however, does not seem prudent when the purpose is to define a foundation for *future* databases. By allowing predicate constants into the domain of discourse it now becomes possible to ask question like: *What does Mark love to do?* Or *Select all the people who like to love people or miss people* and also *What do people love to do?* I would hope that models for future databases, how ever they are called, would at least define operators which allow, the manipulation of and access to, objects in the domain  $dom(I)$  of the interpretation function  $I$ . It is also desired that predicate constants are added to the domain of discourse.<sup>19</sup>

Generic data-mining applications which search for "trends" in databases, generic business repositories, generic database applications, which automatically generate user interfaces allowing user friendly access to databases, intelligent agents which master the art of speech, etc. are examples of applications which would benefit from this.

### 3.4 Summary

The fact that The Third Manifesto rejects constants representing the identity of objects in databases is in my opinion a logical error and as a consequence a big mistake. This rejection of identity is a logical error on the following counts:

<sup>18</sup>A superset of higher order logical called *extensional type logic* is based on allowing predicate constants in the domain of discourse.

<sup>19</sup>This is to say that in my opinion future databases such be firmly based on extensional type or intensional type logic. At least by supplying the necessary primitives that allow extensional and intensional phenomena to be captured.

- The Third Manifesto rejects *identity* on grounds which are not relevant in mathematical logic<sup>20</sup>
- Key concepts like relation variables and candidate keys, are not recognized within the relational algebra of The Third Manifesto. Since these concepts are, according to The Third Manifesto, *required* in future databases, it is an error to not give them a sound mathematical foundation<sup>21</sup>.
- In the definition of a tuple value, it is evident that tuple values include an object identifier called an *attribute name*. Contrary to what is claimed by Date and Darwen<sup>22</sup>, the value of a triple, or tuple with a arity of three, representing an attribute does not define its identity. The object identifier *attribute name* defines the identity of this triple in a tuple because it is the *attribute name* that must be unique.<sup>23</sup>

Adding insult to injury, the rejection of identity also limits the *logical expressiveness* of the algebra upon which future databases are might be based. This opinion has been substantiated by illustrating the correspondence between relational and predicate logical knowledge representation models. In terms of relational database systems the following suggestions are made in this regard:

- Allow for a properly typed equivalent of predicate constants, representing the identity of a predicate. Properly typed object identifiers serve this purpose well.
- Allow for operators which provide access to, and the manipulation of, the equivalent of the domain of predicate logical interpretation functions<sup>24</sup>.

## 4 On the expressive equivalence of relation values and tuple values

### 4.1 Introduction

This section will show that every tuple value has a corresponding representation as a relation value. Conversely every relation value will be shown to have a corresponding representation as a tuple value. This exercise will be performed using liberties allowed by The Third Manifesto.

---

<sup>20</sup>For some reason unknown to the author, Date and Darwen equate identity to *pointers*.

<sup>21</sup>Otherwise, many could claim that The Third Manifesto judges Object Oriented Systems by different standards than relational ones

<sup>22</sup>See quote in section 2.2

<sup>23</sup>Also, please see the next section for an illustration that, given the liberties provided by The Third Manifesto, tuple values and relation values are appearances of one and the same thing

<sup>24</sup>No, the catalog of commercial relational databases does not get it right. Have you ever noticed that, given the operators of relational algebra, it is impossible to perform a trivial operation like selecting every tuple in a relational database?

## 4.2 Defining tuple values

Let us consider tuple values and relation values as they are defined in chapter 3 of The Third Manifesto. A tuple  $t$  is defined as a set of ordered triples  $(I, T, V)$  called *attributes*. Such that:

- $I$  is an identifier called the *name* of an attribute. No two attributes in  $t$  share a common name.
- $T$  is an identifier representing the type of an attribute.
- $V$  is a value of type  $T$ , called the *attribute value*.

The set of pairs obtained by eliminating the attribute value from triples in  $t$  is called the *heading* of  $t$ . The heading of a tuple  $t$  will be denoted:  $heading(t)$ . When the purpose is to show that Relation values and Domain values are basically appearances of one and the same thing, one is inclined to demonstrate that any relation value can also be represented by a set of triples. So, please read on...

## 4.3 Defining relation values

The Third Manifesto defines a relation  $r$  as a pair  $(h, b)$  where :

- $h$  represents the heading of  $r$ . The heading  $h$  is defined to be a tuple heading.
- $b$  represents the body of  $r$ .  $b$  is a set of tuples all conforming to the heading  $h$ .

In the following it will be demonstrated that every relation value<sup>25</sup> can be represented by a mathematically equivalent tuple value<sup>26</sup>

## 4.4 Showing that all relation values are tuple values

The purpose of this section is to illustrate that, by the liberties provided by The Third Manifesto, all relation values are tuple values. Let  $r = (h, b)$  be a relation value with heading  $h$  and body  $b$ . Since tuple values are sets of ordered triples it becomes necessary to demonstrate that all relation values are similarly representable as sets of ordered triples. The body  $b$  of the relation  $r$  will now be defined as a set  $ts$  of ordered triples  $(I, T, V)$ , such that:

- $I$  is an identifier called an *object identifier*
- $T$  is an identifier representing name of a type.
- $V$  is a value of type  $T$ .

To insure that this set of triples  $ts$  forms a valid tuple the following conditions must hold for all triples  $(I, T, V)$  in  $ts$  :

---

<sup>25</sup>under the definition of relation values dictated by The Third Manifesto

<sup>26</sup>under the definition of tuple values dictated by The Third Manifesto

- no two object identifiers in  $ts$  are equal
- the type  $T$  is defined to have the same type as the heading  $h$  of  $r$ , which is to say:  $T = \text{heading}(r)$ .
- $V$  is a tuple value of type  $T$ .

When these conditions hold,  $ts$  will be a valid tuple, containing the same information found in  $r$ . Can this exercise not be performed for any and every relation value?

#### 4.5 Showing that all tuple values are relation values

Let  $t$  be a tuple value the heading of which is denoted  $\text{heading}(t)$ . A relation value  $r = (\text{heading}(t), t)$  is quickly recognized as a relation value which is in no logically significant way different from  $t$ .

#### 4.6 Summary

This section illustrated that tuples and relations as defined by The Third Manifesto are appearances of one and the same thing. This implies that from a logical point of view, only one of the two concepts is a necessity. Noticing that user defined types in principle, allow types of arbitrary complexity supporting a diverse set of operators, domains, as defined by the Third Manifesto would seem to be the most general of the types supported by The Third Manifesto. Domains have been equated to object classes by The Third Manifesto, it is interesting to contemplate the logical implications these of findings.

### 5 Conclusions

With regard to the subject matter of this article the following conclusions are drawn:

- The Third Manifesto, has provided no logically valid substantiation for the claim that the alleged *first great blunder* is indeed a blunder.
- The Third Manifesto, has provided no logically valid substantiation for the claim that the alleged *second great blunder* is indeed a blunder.
- From the perspective of the relational algebra presented in The Third Manifesto, the *requirement* that each relation variable must have at least one candidate key, is an arbitrary one.
- In the relational algebra of the Third Manifesto identity is rejected by Date and Darwen, while it is reified, as a *requirement*, in the form of candidate keys in a context foreign to this relational algebra. This fact makes the rejection of identity in relational algebra, an arbitrary one.

- Domains, which have been equated to object classes by The Third Manifesto have been established to represent a more general class of types than relation types.
- By rejecting identity in the algebra of future database systems The Third Manifesto also limits the logical expressiveness of future databases.

## References

- [1] C.J. Date, Hugh Darwen [2000] Foundation for Future Database Systems, Addison-Wesley Publishing Company.
- [2] J. van Eijck, E. Thijsse, [1989], Logica voor alfa's en informatici, Academic Service
- [3] Hugh Darwen [2002] "Gittens000.pdf", a personal communication
- [4] Maurice Gittens [2002] An anatomy of knowledge representation and a theory of meaning, A document available at <http://www.gittens.nl>