The Third Manifesto

Hugh Darwen and C.J. Date[1]

All logical differences are big differences
—Wittgenstein (attrib.)

ABSTRACT

We present a manifesto for the future direction of data and database management systems. The manifesto consists of a series of prescriptions, proscriptions, and "very strong suggestions."

INTRODUCTION

This is a manifesto regarding the future of data and database management systems. It is intended to follow and, we hope, supersede two previous manifestos [1,25]—hence our choice of title. Reference [1], in spurning the Relational Model of Data, ignores its importance and significance and also, we think, fails to give firm direction. Reference [25], while correctly espousing the Relational Model, fails to mention and emphasize the hopelessness of continuing to follow a commonly accepted perversion of that model, namely SQL, in fond pursuit of the Relational Model's ideals. By contrast, we feel strongly that any attempt to move forward, if it is to stand the test of time, must *reject SQL unequivocally*. However, we do pay some attention to the question of what to do about today's SQL legacy.

BACK TO THE FUTURE

We seek a firm foundation for the future of data. We do not believe that the database language SQL is capable of providing such a foundation. Instead, we believe that any such foundation must be firmly rooted in the **Relational Model of Data**, first presented to the world in 1969 by E. F. Codd in reference [6].

We fully acknowledge the desirability of supporting certain features that have been much discussed in more recent times, including some that are commonly regarded as aspects of **Object Orientation**. We believe that these features are orthogonal to the Relational Model, and therefore that the Relational Model needs no extension, no correction, no subsumption, and, above all, no perversion, in order for them to be accommodated in some database language that could represent the foundation we seek.

Let there be such a language, and let its name be **D**[2].

D shall be subject to certain prescriptions and certain proscriptions. Some prescriptions arise from the Relational Model of Data, and we shall call these **Relational Model Prescriptions**, abbreviated to **RM Prescriptions**. Prescriptions that do not arise from the Relational Model we shall call **Other Orthogonal Prescriptions**, abbreviated to **OO Prescriptions**. We similarly categorize D's proscriptions.

We now proceed to itemize D's prescriptions and proscriptions. The RM Prescriptions and Proscriptions are not negotiable[3]. Unfortunately, the same cannot quite be said of the OO Prescriptions and Proscriptions, as there is not, at the time of writing, a clear and commonly agreed model for them to be based on. We do believe that OO has significant contributions to make in the areas of **user-defined data types** and **inheritance**, but there is still no consensus on an abstract model, even with respect to these important topics; thus, we have been forced to provide our own definitions in these

---

[1] Authors' addresses: Hugh Darwen, IBM United Kingdom Limited, P.O. Box 31, Warwick CV34 5JL, England (email: darwen@vnet.ibm.com); C.J. Date, PO Box 1000, Healdsburg, CA 95448, USA (fax: (1)-707-433-7322). Comments are welcome; in particular, readers are invited to write indicating briefly whether they support or oppose the ideas expressed herein.

[2] No special significance attaches to this choice of name; we use it merely to refer generically to any language that conforms to the principles laid down in subsequent sections.

[3] Some might feel this statement to be excessively dogmatic. What we mean is that prescriptions and proscriptions that arise from the Relational Model are only as negotiable as the features of the Relational Model themselves are.

areas. And it is only fair to warn the reader that inheritance, at least, raises a number of questions that still do not seem to have been satisfactorily answered in the open literature. As a result, our proposals in this area must necessarily be somewhat tentative at this time (see OO Prescriptions 2 and 3).

As well as prescriptions and proscriptions, this manifesto includes some **Very Strong Suggestions**, also divided into RM and OO categories.

Three final preliminary remarks:

1. The version of the Relational Model that we espouse is, very specifically, that version first described in reference [16] (Chapter 15) and further refined (slightly) in reference [11] (Part II). Note, however, that the definitions given herein for *tuple* and *relation* represent a small improvement over the definitions given in those earlier publications.

2. In what follows, we deliberately do not go into a lot of detail on the various prescriptions, proscriptions, and suggestions. (We do sometimes offer a few explanatory comments on certain points, but all such commentary could be deleted without affecting the technical substance of our proposal.) It is our intention to follow this manifesto with a series of more specific papers describing various aspects of our proposal in more depth.

3. In case it might not be obvious, we would like to make it crystal clear that our overriding concern in what follows is with an *abstract model*, not with matters of implementation (though the explanatory comments do sometimes touch on such matters for clarification reasons).

RM PRESCRIPTIONS

1. A **domain** is a named set of values. Such values, which shall be of arbitrary complexity, shall be manipulable *solely* by means of the operators defined for the domain(s) in question (see RM Prescription 3 and OO Proscription 3)—i.e., domain values shall be **encapsulated** (except as noted under RM Prescription 4). For each domain, a notation shall be available for the explicit specification (or "construction") of an arbitrary value from that domain.

   Comments:

   - We treat the terms *domain* and *data type* (*type* for short) as synonymous and interchangeable. The term *object class* is also sometimes used with the same meaning, but we do not use this latter term.

   - We refer to domain values generically as *scalar values* (*scalars* for short). Note, therefore, that we explicitly permit "scalar" values to be arbitrarily complex; thus, e.g., an array of stacks of lists of ... (etc.) might be regarded as a scalar value in suitable circumstances.

2. Scalar values shall always appear (at least conceptually) with some accompanying identification of the domain to which the value in question belongs. In other words, scalar values shall be **typed**.

3. For each ordered list of n domains, not necessarily distinct ($n \geq 0$), D shall support the definition of the valid n-adic **operators** that apply to corresponding ordered lists of n values, one from each of those n domains. Every such operator definition shall include a specification of the domain of the **result** of that operator. Such operator definitions shall be logically distinct from the definitions of the domains to which they refer (instead of being "bundled in" with those definitions).

   Comments:

   - We treat the terms *operator* and *function* as synonymous and interchangeable. The term *method* is also sometimes used with the same meaning, but we do not use this latter term.

   - A function that directly or indirectly assigns to one of its arguments is known as a **mutator**, or simply as a *function with side-effects*. Such functions are generally deprecated, but they cannot be prohibited and they may be needed in connection with inheritance.

4. Let V be a domain. The operators defined for V must necessarily include operators whose explicit purpose is to expose the **actual representation** of values from V. Observe that these operators—but no others—thus violate the encapsulation of values from V.

   Comments:

   - It is our intention (a) that such encapsulation-violating operators be used only in the implementation of other operators, and (b) that this effect be achieved by means of the system's authorization mechanism. In other words, the actual representation of domain values should be hidden from most users.

   - Let V be a domain. We remark that it will often be desirable to define a set of operators whose effect is to expose *one possible representation* (not necessarily the actual representation) for values from V; given such operators,

the user will effectively be able to operate on values from *V* just as if the actual representation were exposed.

- Although the actual representation of domain values is not relevant to the specifications of this manifesto, it might be helpful to point out that if *v1* and *v2* are distinct values from domain *V*, nothing in the D language requires the actual representations of *v1* and *v2* to be the of the same form. For instance, *V* might be the domain "text," and *v1* and *v2* two documents prepared using different word processors.

5. D shall come equipped with certain builtin domains, including in particular the domain of **truth values** (*true* and *false*). The usual operators (NOT, AND, OR, IF ... THEN ..., IFF, etc.) shall be supported for this domain.

6. Let *H* be some tuple heading (see RM Prescription 9). Then it shall be possible to define a domain whose values are tuples with heading *H*—in other words, **TUPLE** shall be a valid type constructor. The operators defined for such a domain shall be, precisely, the set of tuple operators supported by D. Those operators shall include one for constructing a tuple from specified scalars and another for extracting specified scalars from a tuple. They shall also include tuple "nest" and "unnest" capabilities analogous to those described for relations in reference [16] (Chapter 6).

7. Let *H* be some relation heading (see RM Prescription 10). Then it shall be possible to define a domain whose values are relations with heading *H*—in other words, **RELATION** shall be a valid type constructor. The operators defined for such a domain shall be, precisely, the set of relational operators supported by D. Those operators shall include one for constructing a relation from specified tuples and another for extracting specified tuples from a relation. They shall also include relational "nest" and "unnest" capabilities along the lines described in reference [16] (Chapter 6).

   Comment: Note that from the perspective of any relation that includes an attribute defined on such a domain, the "scalar" values in that domain are still (like all domain values) encapsulated. (An analogous remark applies to RM Prescription 6 also.) We explicitly do not espouse NF^2 ("NF squared") relations as described in, e.g., reference [24], which involve major extensions to the classical Relational Algebra.

8. The **equals** comparison operator ("=") shall be defined for every domain. Let *v1* and *v2* each denote some value from some domain, *V*. Then *v1* = *v2* shall be *true* if and only if *v1* and *v2* are the same member of *V*.

9. A **tuple**, *t*, is a set of ordered triples of the form <*A,V,v*>, where:

   - *A* is the name of an **attribute** of *t*. No two distinct triples in *t* shall have the same attribute name.
   - *V* is the name of the (unique) **domain** corresponding to attribute *A*.
   - *v* is a value from domain *V*, called the **attribute value** for attribute *A* within tuple *t*.

   The set of ordered pairs <*A,V*> that is obtained by eliminating the *v* (value) component from each triple in *t* is the **heading** of *t*. Given a tuple heading, a notation shall be available for the explicit specification (or "construction") of an arbitrary tuple with that heading.

10. A **relation**, *R*, consists of a *heading* and a *body*. The **heading** of *R* is a tuple heading *H* as defined in RM Prescription 9. The **body** of *R* is a set *B* of tuples, all having that same heading *H*. The attributes and corresponding domains identified in *H* are the **attributes** and corresponding **domains** of *R*. Given a relation heading, a notation shall be available for the explicit specification (or "construction") of an arbitrary relation with that heading.

    Comments:

    - Note that each tuple in *R* contains exactly one value *v* for each attribute *A* in *H*; in other words, *R* is in *First Normal Form*, 1NF.
    - We draw a sharp distinction between relations *per se* and relation *variables* (see RM Prescription 13). An analogous distinction applies to databases also (see RM Prescription 15). We recognize that these terminological distinctions will, regrettably, be unfamiliar to most readers; we adopt them nevertheless, in the interests of precision.

11. A **scalar variable of type *V*** is a variable whose permitted values are scalars from a specified domain *V*, the **declared domain** for that scalar variable. Creating a scalar variable *S* shall have the effect of initializing *S* to some scalar value—either a value specified explicitly as part of the operation that creates *S*, or some implementation-dependent value if no such explicit value is specified.

12. A **tuple variable of type *H*** is a variable whose permitted values are tuples with a specified tuple

heading *H*, the **declared heading** for that tuple variable. Creating a tuple variable *T* shall have the effect of initializing *T* to some tuple value—either a value specified explicitly as part of the operation that creates *T*, or some implementation-dependent value if no such explicit value is specified.

13. A **relation variable—relvar** for short—**of type *H*** is a variable whose permitted values are relations with a specified relation heading *H*, the **declared heading** for that relvar.

14. Relvars are either *base* or *derived.* A **derived relvar** is a relvar whose value at any given time is a relation that is defined by means of a specified relational expression (see RM Prescriptions 18-20); the relational expression in question shall be such that the derived relvar is updatable according to the rules and principles described in references [11] (Chapter 17) and [18-19]. A **base relvar** is a relvar that is not derived. Creating a base relvar shall have the effect of initializing that base relvar to an empty relation.

    Comment: Base and derived relvars correspond to what are known in common parlance as "base relations" and "updatable views," respectively. Note, however, that we consider many more views to be updatable than have traditionally been so considered [18-19].

15. A **database variable—dbvar** for short—is a named set of relvars. Every dbvar is subject to a set of **integrity constraints** (see RM Prescriptions 23 and 24). The value of a given dbvar at any given time is a set of ordered pairs <*R,r*> (where *R* is a relvar name and *r* is the current value of that relvar), such that (a) there is one such ordered pair for each relvar in the dbvar, and (b) together, those relvar values satisfy the applicable constraints. Such a dbvar value is called a **database** (sometimes a *database state*, but we do not use this latter term).

    Comment: It is worth pointing out that we explicitly do not regard domains as belonging to any particular dbvar.

16. Each **transaction** interacts with exactly one dbvar. However, distinct transactions can interact with distinct dbvars, and distinct dbvars are not necessarily disjoint. Also, a transaction can dynamically change its associated dbvar by adding and/or removing relvars (see RM Prescription 17).

    Comments:

    - One purpose of the dbvar concept is to define a scope for relational operations. That is, if dbvar *X* is the dbvar associated with transaction *T*, then *T* shall not mention any relvar *R* that is part of some distinct dbvar *Y* and not part of dbvar *X*.

    - The set of all base relvars might be regarded as the "base" dbvar. Individual transactions, however, interact with a "derived" or "user" dbvar that consists (in general) of a mixture of base and derived relvars.

    - The mechanism for making and breaking the connection between a transaction and its unique corresponding dbvar is not specified in this manifesto.

17. D shall provide operators to **create** and **destroy** domains, variables (including in particular relvars), and integrity constraints. Every explicitly created domain, variable, or integrity constraint shall be named. Every base relvar shall have at least one **candidate key**, specified explicitly as part of the operation that creates that base relvar.

    Comment: The creation and destruction of dbvars (which we assume to be "persistent") is performed outside the D environment.

18. The **Relational Algebra** as defined in reference [11] (Part II) shall be expressible without excessive circumlocution.

    Comment: "Without excessive circumlocution" implies among other things that:

    - Universal and existential quantification shall be equally easy to express. For example, if D includes a specific operator for relational **projection**, then it should also include a specific operator for the general form of relational **division** described (as DIVIDEBY PER) in reference [16] (Chapter 11).

    - Projection over specified attributes and projection over all but specified attributes shall be equally easy to express.

19. Relvar names and explicit ("constructed") relation values shall both be legal relational expressions.

20. D shall provide operators to create and destroy named **functions** whose value at any given time is a relation that is defined by means of a specified relational expression. Invocations of such functions shall be permitted within relational expressions wherever explicit relation values are permitted.

    Comment: Such functions correspond to what are known in common parlance as "read-only views," except that we permit the relational expressions defining such "views" to be parameterized. Such

parameters represent scalar values and are permitted within the defining relational expression wherever explicit scalar values are permitted. (It might be possible to support tuple and relation parameters also. See RM Very Strong Suggestion 7.)

21. D shall permit:

    a. (The value of) a tuple expression to be **assigned** to a tuple variable, and

    b. (The value of) a relational expression to be **assigned** to a relvar,

    provided in both cases that the requirements of *type compatibility* as described in reference [11] (Chapter 19) are satisfied.

    Comment: Of course, this prescription does not prohibit the additional provision of convenient shorthands such as INSERT, UPDATE, and DELETE as described in reference [11] (Part II).

22. D shall support certain **comparison operators**. The operators defined for comparing tuples shall be "=" and "≠" (only); the operators defined for comparing relations shall include "=", "≠", "is a subset of" (etc.); the operator "∈" for testing membership of a tuple in a relation shall be supported. In all cases, the requirements of *type compatibility* as described in reference [11] (Chapter 19) shall be satisfied.

23. Any expression that evaluates to a truth value is called a **conditional expression**. Any conditional expression that is (or is logically equivalent to) a closed WFF of the Relational Calculus [11] (Part II) shall be permitted as the specification of an **integrity constraint**. Integrity constraints shall be classified according to the scheme described in references [11] (Chapter 16) and [18-19] into **domain, attribute, relation,** and **database** constraints, and D shall fully support the *constraint inference* mechanism required by that scheme.

24. Every relvar has a corresponding **relation predicate** and every dbvar has a corresponding **database predicate**, as explained in references [11] (Chapter 16) and [18-19]. Relation predicates shall be satisfied at statement boundaries. Database predicates shall be satisfied at transaction boundaries.

    Comments:

    - These concepts, which we believe to be both crucial and fundamental, have unfortunately been very much overlooked in the past, and we therefore amplify them slightly here. Basically, a relation predicate is the logical AND of all integrity constraints that apply to the relvar in question, and a database predicate is the logical AND of all integrity constraints that apply to the dbvar in question. The point cannot be emphasized too strongly that it is *predicates*, not *names*, that represent data semantics.

    - To say that relation predicates shall be satisfied at statement boundaries is to say, precisely, that no relational assignment shall leave any relvar in a state in which its relation predicate is violated. To say that database predicates shall be satisfied at transaction boundaries is to say, precisely, that no transaction shall leave the corresponding dbvar in a state in which its database predicate is violated.

    - This prescription further implies that it shall not be possible to update an "updatable view" (i.e., derived relvar) in such a way as to violate the definition of that view. In other words, "updatable views" shall always be subject to what SQL calls CASCADED CHECK OPTION [17].

25. Every dbvar shall include a set of relvars that constitute the **catalog** for that dbvar. It shall be possible to assign to relvars in the catalog.

    Comment: This prescription implies that the catalog must be what is commonly known as "self-describing."

26. D shall be constructed according to well-established principles of good language design as documented in, e.g., reference [3].

    Comment: Arbitrary restrictions such as those documented in references [8] (Chapter 12), [14] and [17], and all other *ad hoc* concepts and constructs, shall thus be absolutely prohibited.

RM PROSCRIPTIONS

The observant reader will note that many of the proscriptions in this section are logical consequences of the RM Prescriptions. In view of the unfortunate mistakes that have been made in SQL, however, we feel it is necessary to write down some of these consequences by way of clarification.

1. D shall include no construct that depends on the definition of some ordering for the attributes of a relation. Instead, for every relation *R* expressible in D, the attributes of *R* shall be distinguishable by *name*.

    Comment: This proscription implies no more anonymous columns, as in SQL's SELECT X + Y FROM T, and no more duplicate column names, as

in SQL's SELECT X, X FROM T and SELECT T1.X, T2.X FROM T1, T2.

2. D shall include no construct that depends on the definition of some ordering for the tuples of a relation.

   Comment: This proscription does not imply that such an ordering cannot be imposed for, e.g., presentation purposes; rather, it implies that the effect of imposing such ordering is to convert the relation into something that is not a relation (perhaps a sequence or ordered list).

3. For every relation *R*, if *t1* and *t2* are distinct tuples in *R*, then there must exist an attribute *A* of *R* such that the attribute value for *A* in *t1* is not equal to the attribute value for *A* in *t2*.

   Comment: In other words, "duplicate rows" are absolutely, categorically, and unequivocally outlawed. What we tell you three times is true.

4. Every attribute of every tuple of every relation shall have a value that is a value from the applicable domain.

   Comment: In other words—no more nulls, and no more many-valued logic!

5. D shall not forget that relations with zero attributes are respectable and interesting, nor that candidate keys with zero components are likewise respectable and interesting.

6. D shall include no constructs that relate to, or are logically affected by, the "physical" or "storage" or "internal" levels of the system (other than the functions that explicitly expose the actual representation of domain values—see RM Prescription 4). If an implementer wants or needs to introduce any kind of "storage structure definition language," the statements of that language, and the mappings of dbvars to physical storage, shall be cleanly separable from everything expressed in D.

7. There shall be no tuple-at-a-time operations on relations.

   Comments:

   - INSERT, UPDATE, and DELETE statements, if provided, insert or update or delete (as applicable) a set of tuples, always; a set containing a single tuple is just a special case (though it might prove convenient to offer a syntactic shorthand for that case).
   - Tuple-at-a-time retrieval (analogous to SQL's FETCH via a cursor)—though prohibited, and generally deprecated to boot—can effectively be performed, if desired, by converting the relation to an ordered list of tuples and iterating over that list.
   - Tuple-at-a-time update (analogous to SQL's UPDATE and DELETE via a cursor) is categorically prohibited.

8. D shall not include any specific support for "composite domains" or "composite columns" (as proposed in, e.g., reference [4]), since such functionality can be achieved if desired through the domain support already prescribed. See reference [9].

9. "Domain check override" operators (as documented in, e.g., reference [4]) are *ad hoc* and unnecessary and shall not be supported.

10. D shall not be called SQL.

OO PRESCRIPTIONS

1. D shall permit **compile-time type checking**.

   Comment: By this prescription, we mean that—insofar as feasible—it shall be possible to check at compilation time that no type error can occur at run time. This requirement does not preclude the possibility of "compile and go" or interpretive implementations.

2. **(Single inheritance)** If D permits some domain *V′* to be defined as a **subdomain** of some **superdomain** *V*, then such a capability shall be in accordance with some clearly defined and generally agreed model.

   Comments:

   - It is our hope that such a "clearly defined and generally agreed" inheritance model will indeed someday be found. The term "generally agreed" is intended to imply that the authors of this manifesto, among others, shall be in support of the model in question. Such support shall not be unreasonably withheld.
   - We note that support for inheritance implies certain extensions to the definitions of *scalar variable, tuple variable, relation*, and *relvar*. It also seems to imply that OO Prescription 1 might need to be relaxed slightly. A possible model for inheritance that incorporates these points is sketched in a forthcoming appendix to this manifesto (draft version currently available from the authors).

3. **(Multiple inheritance)** If D permits some domain *V′* to be defined as a subdomain of some superdo-

main *V*, then *V′* shall not be prevented from additionally being defined as a subdomain of some other domain *W* that is neither *V* nor any superdomain of *V* (unless the requirements of OO Prescription 2 preclude such a possibility).

4. D shall be **computationally complete**. That is, D may support, but shall not require, invocation from so-called "host programs" written in languages other than D. Similarly, D may support, but shall not require, the use of other programming languages for implementation of user-defined functions.

   Comment: We do not intend this prescription to undermine such matters as D's optimizability unduly. Nor do we intend it to be a recipe for the use of procedural constructs such as loops to perform database queries or integrity checks. Rather, the point is that computational completeness will be needed (in general) for the implementation of user-defined functions. To be able to implement such functions in D itself might well be more convenient than having to make excursions into some other language—excursions that in any case are likely to cause severe problems for optimizers. Of course, we agree that it might prove desirable to prohibit the use of certain D features outside the code that implements such functions; on the other hand, such a prohibition might too severely restrict what can be done by a "free-standing" application program (i.e., one that does not require invocation from some program written in some other language). More study is needed.

5. Transaction initiation shall be performed only by means of an explicit "**start transaction**" operator. Transaction termination shall be performed only by means of a "**commit**" or "**rollback**" operator; "commit" must be explicit, but "rollback" can be implicit (if the transaction fails through no fault of its own).

   Comment: If transaction *T* terminates via commit ("normal termination"), changes made by *T* to the applicable dbvar are committed. If transaction *T* terminates via rollback ("abnormal termination"), changes made by *T* to the applicable dbvar are rolled back. In other words, dbvars (only) possess the property of "persistence."

6. D shall support **nested transactions**—i.e., it shall permit a transaction *T1* to start another transaction *T2* before *T1* itself has finished execution, in which case:

   a. *T2* and *T1* shall interact with the same dbvar (as is in fact required by RM Prescription 16).

   b. D shall not preclude the possibility that *T1* and *T2* be able to execute asynchronously. However, *T1* shall not be able to complete before *T2* completes (in other words, *T2* shall be wholly contained within *T1*).

   c. Rollback of *T1* shall include the undoing of *T2* even if *T2* was committed.

7. Let *A* be an **aggregate** operator (such as SUM) that is essentially just shorthand for some iterated dyadic operator θ (the dyadic operator is "+" in the case of SUM). If the argument to *A* happens to be empty, then:

   a. If an identity value exists for θ (the identity value is 0 in the case of "+"), then the result of that invocation of *A* shall be that identity value;

   b. Otherwise, the result of that invocation of *A* shall be undefined.

OO PROSCRIPTIONS

1. Relvars are not domains.

   Comment: In other words, we categorically reject the equation "relation = object class" (more accurately, the equation "relvar = object class") espoused in, e.g., reference [23].

2. No value (scalar or any other kind) shall possess any kind of ID that is somehow distinct from the value *per se*.

   Comments:

   - In other words, we reject the idea of *"object IDs."* As a consequence, we reject (a) the idea that "objects" might make use of such IDs in order to share "subobjects," and (b) the idea that users might have to "dereference" such IDs (either explicitly or implicitly) in order to obtain values.

   - We also reject the idea of "tuple IDs" (some writers seem to equate tuple IDs and object IDs).

   - This proscription does not prevent objects outside the dbvar from having IDs that are "somehow distinct from" the object *per se*, nor does it prevent such IDs from appearing within the dbvar. (We should stress that we use the term "object" here in its general sense, not in its specialized object-oriented sense.) Thus, for example, a domain of host operating system filenames is a legal domain.

3. Any "*public instance variable*" notation provided for operating on values in domains shall be mere

syntactic shorthand for certain special function invocations (and perhaps "pseudovariable references," if such instance variables can appear on the left-hand side of assignment operations). There shall not necessarily be any direct correlation between such instance variables and the actual representation of the domain values in question.

4. D shall not include either the concept of "*protected*" (as opposed to private) instance variables or the concept of "*friends*" (see reference [20] for an explanation of these concepts).

   Comment: We believe the problem that such concepts are intended to address is better solved by means of the system's authorization mechanism.

RM VERY STRONG SUGGESTIONS

1. It should be possible to specify one or more **candidate keys** for each derived relvar. For each relvar (base or derived) for which candidate keys have been specified, it should be possible to nominate exactly one of those candidate keys as the **primary** key.

   Comment: Every relvar does possess one or more candidate keys (of which at least one, and preferably all, must be so designated by the user in the case of base relvars, as required by RM Prescription 17). Designation of one particular candidate key as primary is optional, however, for reasons explained in reference [13].

2. D should include support for **system-generated** keys along the lines described in references [16] (Chapter 5) and [7] (Chapter 19).

3. D should include some convenient declarative shorthand for expressing (a) **referential constraints** and (b) **referential actions** such as "cascade delete."

4. It is desirable, but thought not to be completely feasible, for the system to be able to infer the (time-independent) candidate keys of every relation *R* expressible in D, such that:

   a. Candidate keys of *R* become candidate keys of *R′* when *R* is assigned to *R′*, and

   b. Candidate keys of *R* may be included in the information about *R* that is available to a user of D.

   D should provide such functionality, but without any guarantee that inferred keys are not proper supersets of actual keys, or even that some superset (proper or otherwise) is discovered for every actual key. Implementations of D can thus compete with each other in their degree of success at discovering candidate keys.

   Comment: The recommendation that candidate keys should (as far as possible) be inferred for derived relations is in fact subsumed by RM Prescription 23, which requires support for a general constraint inference mechanism. We mention candidate keys explicitly here because we regard them as an important special case, for reasons explained in reference [16] (Chapter 10).

5. D should provide some convenient (nonprocedural) means of expressing **quota queries** (e.g., "find the three youngest employees"). Such a capability should not be bundled with the mechanism that converts a relation into an ordered list (see RM Proscription 2).

6. D should provide some convenient (nonprocedural) means of expressing the **generalized transitive closure** of a graph relation, including the ability to perform generalized *concatenate* and *aggregate* operations as described in reference [21].

7. D should permit the parameters to relation-valued functions to represent tuples and relations as well as scalars.

   Comment: We make this a suggestion merely, rather than a prescription, because we believe it requires further study at this time.

8. D should provide a mechanism for dealing with "missing information" along the lines of the default value scheme described in Chapter 21 of reference [16] (but based on domains rather than attributes).

   Comment: The term "default values" is perhaps misleading, inasmuch as it suggests an interpretation that was not intended—namely, that the value in question occurs so frequently that it might as well be the default. Rather, the intent is to use an appropriate "default" value, distinct from all possible genuine values, when no genuine value can be used. For example, if the genuine values of the attribute HOURS_WORKED are positive integers, the default value "?" might be used to mean that (for some reason) no genuine value is known. Note, therefore, that the domain for HOURS_WORKED is *not* simply the domain of positive integers.

9. **SQL** should be implementable in D—not because this is desirable *per se*, but so that a painless migration route might be available for current SQL users. To this same end, existing SQL databases should be convertible to a form that D programs can operate on without error.

Comment: The foregoing does not imply that D must be a superset of SQL, but rather that it should be possible to write a frontend layer of D code on top of D's true relational functionality that:

a. Will accept SQL operations against converted SQL data, and

b. Will give the results that SQL would have given if those SQL operations had been executed against the original unconverted SQL data.

We should stress that we believe it possible to construct such an SQL frontend without contravening any of the prescriptions and proscriptions laid down in this manifesto.

OO VERY STRONG SUGGESTIONS

1. Some form of **type inheritance** should be supported (in which case, see OO Prescriptions 2 and 3). In keeping with this suggestion, D should not include: (a) the concept of implicit type conversion; (b) the concept that functions have a special "distinguished" or "receiver" parameter.

   Comment: Implicit type conversions would undermine the objective of substitutability; distinguished parameters would introduce an artificial and unnecessary degree of asymmetry. Both these points are amplified in the forthcoming appendix on inheritance mentioned in OO Prescription 2.

2. "Collection" type constructors, such as **LIST, ARRAY,** and **SET**, as commonly found in languages supporting rich type systems, should be supported. (See also RM Prescription 7.)

3. Let C be a collection type constructor other than **RELATION**. Then a conversion function, say C2R, should be provided for converting values of type C to relations, and an inverse function, say R2C, should also be provided, such that:

   a. C2R(R2C(*r*)) = *r* for every relation *r* expressible in D;

   b. R2C(C2R(*c*)) = *c* for every expressible value *c* of type C.

4. D should be based on the "single-level storage" model as described in, e.g., reference [15]. In other words, it should make no logical difference whether a given piece of data resides in main memory, secondary storage, tertiary storage, etc.

CONCLUDING REMARKS

We have presented a manifesto for the future direction of data and database management systems. Now perhaps is the time to confess that we do feel a little uncomfortable with the idea of calling what is, after all, primarily a technical document a "manifesto." According to Chambers Twentieth Century Dictionary, a manifesto is a "written declaration of the intentions, opinions, or motives" of some person or group (e.g., a political party). This particular written declaration, by contrast, is—we hope—a matter of science and logic, not mere "intentions, opinions, or motives." Given the historical precedents that led us to write this document, however, our title was effectively chosen for us.

By way of summary, we present an abbreviated mnemonic list of all of the prescriptions, proscriptions, and very strong suggestions discussed in the foregoing sections.

RM Prescriptions

1. Domains
2. Typed scalars
3. Scalar operators
4. Actual representation
5. Truth values
6. Type constructor TUPLE
7. Type constructor RELATION
8. Equality operator
9. Tuples
10. Relations
11. Scalar variables
12. Tuple variables
13. Relation variables (relvars)
14. Base *vs.* derived relvars
15. Database variables (dbvars)
16. Transactions and dbvars
17. Create/destroy operations
18. Relational algebra
19. Relvar names and explicit relation values
20. Relational functions
21. Relation and tuple assignment
22. Comparisons
23. Integrity constraints
24. Relation and database predicates
25. Catalog
26. Language design

RM Proscriptions

1. No attribute ordering
2. No tuple ordering
3. No duplicate tuples
4. No nulls
5. No nullological mistakes
6. No internal-level constructs
7. No tuple-level operations
8. No composite columns
9. No domain check override
10. Not SQL

OO Prescriptions

1. Compile-time type checking
2. Single inheritance (conditional)
3. Multiple inheritance (conditional)
4. Computational completeness
5. Explicit transaction boundaries
6. Nested transactions
7. Aggregates and empty sets

OO Proscriptions

1. Relvars are not domains
2. No object IDs
3. No "public instance variables"
4. No "protected instance variables" or "friends"

RM Very Strong Suggestions

1. Candidate keys for derived relvars
2. System-generated keys
3. Referential integrity
4. Candidate key inference
5. Quota queries
6. Transitive closure
7. Tuple and relation parameters
8. Default values
9. SQL migration

OO Very Strong Suggestions

1. Type inheritance
2. Collection type constructors
3. Conversion to/from relations
4. Single-level store

ACKNOWLEDGMENTS

We are grateful to many friends and colleagues for their encouragement and for technical discussions and helpful comments on earlier drafts of this manifesto: Tanj Bennett, Charley Bontempo, Keith Charrington, Linda DeMichiel, Vincent Dupuis, Mark Evans, Ron Fagin, Oris Friesen, Michael Jackson, John Kneiling, Adrian Larner, Bruce Lindsay, Albert Maier, Carl Mattocks, Nelson Mattos, David McGoveran, Serge Miranda, Jim Panttaja, Mary Panttaja, Stephane Parent, Fabian Pascal, Ron Ross, Arthur Ryman, Mike Sykes, Stephen Todd, Rick van der Lans, Anton Versteeg, and Fred Wright.

Hugh Darwen adds: In November, 1994, IBM's Software Solutions Division announced a consolidation of its R&D activities that resulted in the closure of its software development laboratory at Warwick in England. I had been an employee of this lab, where technical vitality thrived and prospered, since 1987, and I would like to express my appreciation for the support I was given by lab director Tony Temple and several of his managers for extracurricular pursuits such as this manifesto. My very special thanks go to my first manager, **Stuart Colvin**, and my last one, **Dick Chapman**.

REFERENCES AND BIBLIOGRAPHY

1. Malcolm Atkinson *et al.*: "The Object-Oriented Database System Manifesto," Proc. First International Conference on Deductive and Object-Oriented Databases, Kyoto, Japan (1989). New York, N.Y.: Elsevier Science (1990).

2. David Beech: "Collections of Objects in SQL3," Proc. 19th International Conference on Very Large Data Bases, Dublin, Ireland (August 1993).

3. Jon Bentley: "Little Languages," *CACM 29*, 8 (August 1986).

   Illustrates and discusses the following "yardsticks of language design": *orthogonality, generality, parsimony, completeness, similarity, extensibility,* and *openness*.

4. E. F. Codd: *The Relational Model for Database Management Version 2*. Reading, Mass.: Addison-Wesley (1990).

   Codd spent much of the late 1980s revising and extending his original model (which he now refers to as "the Relational Model Version 1" or RM/V1), and this book is the result. It describes "the Relational Model Version 2" or RM/V2. (We include this reference primarily so that we can make it clear that the version of the Relational Model espoused in the present manifesto is *not* "RM/V2," nor indeed "RM/V1" as Codd currently defines it. Rather, it is the version described in Part II of reference [11] and Chapter 15 of reference [16].)

5. E. F. Codd: "A Relational Model of Data for Large Shared Data Banks," *CACM 13*, 6 (June 1970). Republished in *Milestones of Research—Selected Papers 1958-1982* (CACM 25th Anniversary Issue), *CACM 26*, 1 (January 1983).

   The first widely available description of the original Relational Model, by its inventor.

6. E. F. Codd: "Derivability, Redundancy, and Consistency of Relations Stored in Large Data Banks," IBM Research Report RJ599 (August 19th, 1969).

   A preliminary version of reference [5].

7. Hugh Darwen (writing as Andrew Warden): "Adventures in Relationland", a special contribution to reference [8].

8. C. J. Date, *Relational Database Writings 1985-1989.* Reading, Mass.: Addison-Wesley (1990).

9. C. J. Date, "We Don't Need Composite Columns," *Database Programming & Design 8*, 5 (May 1995, to appear).

10. C. J. Date: "Oh Oh Relational ...," *Database Programming & Design 7*, 10 (October 1994).

    Explains why the equation "domain = object class" is right and the equation "relvar = object class" is wrong, and describes the benefits that would accrue from a true rapprochement between relational and

OO principles as advocated in the present manifesto. References [2] and [23] are examples of papers that advocate the "relvar = object class" equation.

11. C. J. Date: *An Introduction to Database Systems* (6th edition). Reading, Mass.: Addison-Wesley (1995).

    There are a few discrepancies (mostly minor) between the version of the Relational Model described in Part II of this book and that described in Chapter 15 of reference [16]. Where such discrepancies occur, this book should be regarded as superseding.

12. C. J. Date: "How We Missed the Relational Boat" (published under the title "How SQL Missed the Boat"), *Database Programming & Design 6*, 9 (September 1993).

13. C. J. Date: "The Primacy of Primary Keys: An Investigation," *InfoDB 7*, 3 (Summer 1993).

14. C. J. Date: "A Critique of the SQL Database Language," *ACM SIGMOD Record 14*, 3 (November 1984). Republished in C. J. Date, *Relational Database: Selected Writings*. Reading, Mass.: Addison-Wesley (1986).

15. C. J. Date: "An Architecture for High-Level Language Database Extensions," Proc. ACM SIGMOD International Conference on Management of Data, Washington, D.C. (June 1976).

16. C.J. Date and Hugh Darwen: *Relational Database Writings 1989-1991*. Reading, Mass.: Addison-Wesley (1992).

17. C. J. Date and Hugh Darwen: *A Guide to the SQL Standard* (3rd edition). Reading, Mass.: Addison-Wesley (1993).

    This book is a tutorial reference to the current SQL standard ("SQL/92"). It contains numerous examples of violations of good language design principles. In particular, it includes an appendix (Appendix D) that documents "many aspects of the standard that appear to be inadequately defined, or even incorrectly defined, at this time."

18. C. J. Date and David McGoveran: "Updating Joins and Other Views," *Database Programming & Design 7*, 8 (August 1994).

19. C. J. Date and David McGoveran: "Updating Union, Intersection, and Difference Views," *Database Programming & Design 7*, 6 (June 1994).

20. Margaret A. Ellis and Bjarne Stroustrup: *The Annotated C++ Reference Manual*. Reading, Mass.: Addison-Wesley (1990).

21. Nathan Goodman: "Bill of Materials in Relational Database," *InfoDB 5*, 1 (Spring/Summer 1990).

22. P. A. V. Hall, P. Hitchcock, and S. J. P. Todd: "An Algebra of Relations for Machine Computation," Conference Record of the 2nd ACM Symposium on Principles of Programming Languages, Palo Alto, Calif. (January 1975).

23. William Kelley and Won Kim, "Observations on the Current SQL3 Object Model Proposal (and Invitation for Scholarly Opinions)," available from UniSQL, Inc., 9390 Research Blvd., Austin, Texas 78759 (1994).

24. Mark A. Roth, Henry F. Korth, and Abraham Silberschatz: "Extended Algebra and Calculus for Nested Relational Databases," *ACM TODS 13*, 4 (December 1988).

25. Michael Stonebraker *et al.*: "Third Generation Database System Manifesto," *ACM SIGMOD Record 19*, 3 (September 1990).