A FORMAL DEFINITION OF THE RELATIONAL MODEL

C.J.Date

IBM General Products Division
555 Bailey Avenue, San Jose
California, USA 95150

September 1981

Abstract

The relational model of data, originally introduced by Codd in [1], has three components: (1) a set of objects (relations, domains, etc.); (2) a set of operators (union, project, etc.); (3) a set of general integrity rules. The purpose of this paper is to provide a formal definition of each of these three components.

Note: This paper consists of material extracted from the author's forthcoming book "Developments in Database Technology" (tentative title), scheduled for publication in 1982.

INTRODUCTION

The relational model of data, first introduced by Codd in 1970 [1], has undergone a certain amount of revision and refinement since its original formulation. It is true, and indeed significant, that the changes have been evolutionary, not revolutionary, in nature; nevertheless, the situation is now that (to this writer's knowledge) there does not exist any particularly formal definition of the relational model as it currently stands. It therefore seems worthwhile to attempt such a definition, to provide a convenient source of reference for the material and to pave the way for an understanding of some of the more recent work on extending the model to incorporate additional meaning [2]. (Note: Reference [2] does include a definition of the "basic" relational model as well as of numerous extensions thereto, but that definition is considerably less formal than the one given in the present paper.)

The relational model, like any other data model, consists of three components [3]:

(1) a set of objects;

(2) a set of operators;

(3) a set of general integrity rules.

We examine each of these components in turn, under the headings "Relational database", "Relational operations", and "Relational rules", respectively. Our definitions cover the following concepts, among others:

Relational database

- domain
- relation
  (i.e., relation variable)
- real relation
  (base relation)
- virtual relation
  (view)
- degree
- attribute
- tuple
- candidate key
- primary key
- alternate key
- named relation
- unnamed relation

Relational operations

- union
- difference
- product
  (i.e., extended Cartesian product)
- theta-selection
- projection
- theta-join
- equijoin
- natural join
- relational-assignment

Relational rules

- entity integrity
- referential integrity

As already stated, we are concerned in this paper with formal definitions of these concepts; we will not generally be discussing their intended interpretation (the reader is assumed to be familiar with such informal aspects already; see, e.g., reference [5] for a tutorial treatment). Also, it should be stressed at the outset that the precise scope of "the relational model" as defined herein reflects to some extent a personal choice on the part of the writer. Other definitions are possible, and may either include additional concepts or exclude some concepts that are included here. Thus, for example, we choose to exclude the operators introduced in [2] for dealing with null values, for reasons given in [7].

We begin with the notion of relational-system. (Note: For the purposes of this paper we shall use certain typographical conventions whenever we wish to be formal. First, all terms to be formally defined will be hyphenated, if necessary, to exclude any intervening blanks. Second, terms defined by means of the production rules of Figs. 1 and 2, such as <relational-database>, will additionally be enclosed in angle brackets.) A relational-system is a database system constructed in accordance with the relational model. More formally, a relational-system consists of three components: a <relational-database>, a collection of <relational-operation>s, and two <relational-rule>s.

As already indicated, it is convenient to summarize the structure of a <relational-database> by means of a set of production rules (Fig. 1). Please note that Fig. 1 is not a proposal for a concrete syntax for relational database declarations; rather, it is an abstract syntax, whose function is merely to indicate the abstract structure of the various objects being defined. Observe in particular that the syntax does not include any mention of the parentheses or other separators that would be needed in practice to avoid ambiguity.

In presenting the syntax we make use of the following convenient shorthand. Let <xyz> be an arbitrary syntactic category. Then the expression <xyz-set> is also a syntactic category, representing an unordered, possibly empty set of objects of type <xyz>.

1. <relational-database>
   ::= <domain-set> <relation-set>

2. <domain>
   ::= <domain-name> <domain-value-set> <ordering-indicator>

3. <domain-name>
   ::= <name>

4. <domain-value>
   ::= <atom>

5. <ordering-indicator>
   ::= YES | NO

6. <relation>
   ::= <named-relation> | <unnamed-relation>

7. <named-relation>
   ::= <real-relation> | <virtual-relation>

8. <real-relation>
   ::= <relation-name> <attribute-set>
       <primary-key> <alternate-key-set> <tuple-set>

9. <relation-name>
   ::= <name>

10. <attribute>
    ::= <attribute-name> <domain-name>

11. <attribute-name>
    ::= <name>

12. <primary-key>
    ::= <candidate-key>

13. <candidate-key>
    ::= <attribute-name-set>

14. <alternate-key>
    ::= <candidate-key>

15. <tuple>
    ::= <attribute-value-set>

16. <attribute-value>
    ::= <attribute-name> <domain-value>

17. <virtual-relation>
    ::= <relation-name> <relational-expression>

formal definition of RDM

18. <unnamed-relation>
::= <relational-expression>

Fig. 1: The structure of a relational-system (Part 1 of 3):
the <relational-database>

Now we discuss the constructs of Fig. 1 in detail. The following paragraphs are numbered to correspond to the production rules of the figure.

1. A <relational-database> consists of a set of <domain>s and a set of <relation>s. Two points arise immediately. First, note that we are using the term <relational-database> to mean, specifically, a variable, in the programming language sense; the value of this variable is assumed to change with time. Similar remarks apply to <relation> (but not to <domain>; a <domain> is not a variable, and its value does not change with time). Second, our definition of relational-system is still rather static, the previous point notwithstanding; that is, the <relational-operation>s (defined in Fig. 2, later) make no provision for adding new <named-relation>s to the database, nor for destroying existing <named-relation>s, nor for any other changes to the database declaration. Extension of the definitions to handle these aspects is beyond the scope of this paper.

2. A <domain> consists of a <domain-name>, a (fixed, nonempty) set of <domain-value>s, and an <ordering-indicator>.

3. A <domain-name> is simply a <name>, representing the name of the <domain> concerned. (We do not define <name>s any further in this syntax.) Every <domain> in a given <relational-database> must have a unique <domain-name>.

4. A <domain-value> is an <atom> - i.e., a value that is nondecomposable so far as the relational-system is concerned. All <domain-value>s for a given <domain> are of the same data-type. We choose not to formalize the notion of "data-type" in this presentation.

Note: We assume for the present that <domain>s may include null values - i.e., that the null value is a legal <atom>. However, we do not consider null values in any real depth in this paper. In particular, we do not consider the special treatment required by null values in the definition of <relational-operation>s such as <union>. See [7] for such a discussion.

5. The <ordering-indicator> indicates whether the <domain> concerned is ordered - i.e., whether the operator "greater than" is applicable between pairs of <atom>s from the <domain>.

6. A <relation> is either a <named-relation> or an <unnamed-relation>.

7. A <named-relation> is either a <real-relation> or a <virtual-relation>.

8. A <real-relation> consists of a <relation-name>, a nonempty set of <attribute>s, a <primary-key>, a set of <alternate-key>s, and a (time-varying) set of <tuple>s.

9. A <relation-name> is a <name> (it is the name of the <named-relation> concerned). Every <named-relation> in a given <relational-database> must have a unique <relation-name>.

10. An <attribute> consists of an <attribute-name> and a <domain-name>. The <domain> identified by <domain-name> is said to correspond to the <attribute> identified by <attribute-name> (see paragraph 11); equivalently, the <attribute> identified by <attribute-name> is said to be defined on the <domain> identified by <domain-name>. The number of <attribute>s in the <attribute-set> of a given <relation> is the degree of the <relation> concerned.

Note: As we shall see later, all <relation>s (<real-relation>s, <virtual-relation>s, and <unnamed-relation>s) possess an <attribute-set>. For reasons to be explained, however, our formalism shows <attribute-set> as an explicit component of <real-relation>s only. Similar remarks apply

to <primary-key>, <alternate-key-set>, and <tuple-set>; see paragraphs 12-16.

11. An <attribute-name> is a <name> (it is the name of the <attribute> concerned). No two <attribute>s of a given <relation> can have the same <attribute-name>.

12. A <primary-key> is a <candidate-key> (see paragraph 13). Every <relation> has exactly one <primary-key>.

13. A <candidate-key> is a nonempty set of <attribute-name>s. Every <attribute-name> appearing in a given <candidate-key> must be the <attribute-name> for some <attribute> of the <relation> concerned. Every <candidate-key> of a <relation> R satisfies the following two properties:

- Uniqueness property

  At any given time, no two <tuple>s of the current <tuple-set> of R have the same combination of <attribute-value>s for the set of <attribute>s designated by the specified <attribute-name-set> (see paragraphs 15 and 16).

- Minimality property

  No <attribute-name> can be discarded from the specified <attribute-name-set> without destroying the uniqueness property.

Conversely, any <attribute-name-set> possessing these two properties is a <candidate-key>.

For a given <relation>, exactly one <candidate-key> is (arbitrarily) designated as the <primary-key>, and all other <candidate-key>s are designated as <alternate-key>s (for that <relation>). Note: The distinction between <primary-key> and <alternate-key> is significant in practice only for <real-relation>s, not for <virtual-relation>s and not for <unnamed-relation>s.

14. An <alternate-key> is a <candidate-key>. Every <relation> has zero or more <alternate-key>s. See paragraph 13.

15. A <tuple> is a set of <attribute-value>s. The <tuple-set> of a given <relation> represents, informally, the value of that <relation>. Let R be a <relation> and let T be the <tuple-set> within R at some particular time. Then every <tuple> of T conforms to the <attribute-set> of R; that is, it contains exactly one <attribute-value> for each distinct <attribute> of R, and the <attribute-name> within that <attribute-value> is the name of that <attribute> (see paragraph 16).

16. An <attribute-value> consists of an <attribute-name> and a <domain-value>. The <domain-value> is a value from the (unique) <domain> corresponding to the <attribute> identified by <attribute-name>. (Informally, we often consider the <domain-value> alone as the "attribute-value," and ignore the <attribute-name> component.)

17. A <virtual-relation> consists of a <relation-name> and a <relational-expression>. The <relational-expression> is an expression of the relational algebra, whose function is to specify the definition of the <virtual-relation> in terms of other <named-relation>s in the <relational-database> (see Fig. 2). The <virtual-relation>, like a <real-relation>, possesses both <attribute>s and a <tuple-set>; however, the <attribute>s and <tuple-set> of a <virtual-relation> are derived, in a manner to be explained later, from the <relational-expression>, and are therefore not shown as separate syntactic components. We also omit any mention of <candidate-key> (and <primary-key> and <alternate-key>); while these concepts do apply to <virtual-relation>s, they are not particularly significant in this context, and we exclude them from our formalism accordingly.

18. An <unnamed-relation> consists of a <relational-expression>. Informally, the <unnamed-relation> represents the result of evaluating that <relational-expression>. The remarks in paragraph 17 concerning <attribute>s, <tuple-set>, and <candidate-key>s (and <primary-key> and <alternate-key>s) apply to <unnamed-relation>s also, mutatis mutandis.

The operations of the relational algebra are an integral component of the relational model. A particular relational implementation may support the operators of the algebra directly, or it may provide some alternative set of constructs, such as those of the relational calculus (tuple version or domain version [4,9]). Here we consider just the algebraic operators, since they are in a sense the most fundamental. Again we present our definitions in terms of a set of production rules (Fig. 2); again this syntax should not be construed as a proposal for a concrete language.

We also define a <relational-assignment> operator (not part of the relational algebra per se). The function of this operator is to assign the result of evaluating some specified <relational-expression> to some specified <real-relation>.

```
19. <relational-operation>
      ::= <relational-algebra-operation>
        | <relational-assignment>

20. <relational-algebra-operation>
     ::= <union>
       | <difference>
       | <product>
       | <theta-selection>
       | <projection>

21. <union>
     ::= UNION <relation-name> <relation-name> <correspondence>

22. <correspondence>
     ::= <attribute-name-pair-set>

23. <attribute-name-pair>
     ::= <attribute-name> <attribute-name>

24. <difference>
     ::= DIFFERENCE <relation-name> <relation-name> <correspondence>

25. <product>
     ::= PRODUCT <relation-name> <relation-name>

26. <theta-selection>
     ::= THETA_SELECT <relation-name> <theta-comparison>

27. <theta-comparison>
     ::= <attribute-name> <theta> <comparand>

28. <theta>
     ::= = | ¬= | < | <= | > | >=

29. <comparand>
     ::= <atom> | <attribute-name>

30. <projection>
     ::= PROJECT <relation-name> <attribute-name-set>

31. <relational-assignment>
     ::= ASSIGN <relation-name>
              <relational-expression> <correspondence>

32. <relational-expression>
     ::= <relation-name>
       | <relational-algebra-operation>
       | <relation-literal>

33. <relation-literal>
     ::= <attribute-set> <tuple-set>
```

Fig. 2: The structure of a relational-system (Part 2 of 3): the <relational-operation>s

formal definition of RDM

Before explaining Fig. 2 in detail, we introduce a convenient simplifying notation, which we define as follows.

- The expression R(A1:D1,...,An:Dn) denotes a <relation> called R, of degree n, with <attribute-set> (A1:D1,...,An:Dn) - i.e., with <attribute>s A1, ..., An, defined on <domain>s D1, ..., Dn, respectively. When the <domain>s are irrelevant to the purpose at hand, we shall abbreviate the expression to just R(A1,...,An). If the <attribute>s are also irrelevant, we shall abbreviate the expression still further to simply R.

- Similarly, we shall use (A1:a1,...,An:an) to denote a <tuple> of R(A1,...,An). Here a1, ..., an are values of <attribute>s A1, ..., An, respectively (and are therefore drawn from <domain>s D1, ..., Dn, respectively).

We shall also need the notion of union-compatibility.

- Two <relation>s of degree n, say R(A1,...,An) and S(B1,...,Bn), are said to be union-compatible with respect to a correspondence C, if and only if C is a set of exactly n ordered pairs of <attribute-name>s (Ai,Bj) (i,j = 1,...,n), and the following three conditions hold:

  (a) each <attribute-name> for R is some Ai (i = 1,...,n);

  (b) each <attribute-name> for S is some Bj (j = 1,...,n);

  (c) within each pair (Ai,Bj) of the set, the <attribute>s designated by Ai and Bj have the same corresponding <domain>.

  Note that, for given R and S, there may exist more than one such C - i.e., more than one correspondence satisfying conditions (a), (b), and (c).

We now continue with our detailed explanations.

19. A <relational-operation> is either a <relational-algebra-operation> or a <relational-assignment>.

20. A <relational-algebra-operation> is a <union>, a <difference>, a <product>, a <theta-selection>, or a <projection>. The result of evaluating a <relational-algebra-operation> is to generate an <unnamed-relation>. As explained earlier (paragraph 18), that <unnamed-relation> possesses both an <attribute-set> and a <tuple-set>; moreover, each <tuple> in that <tuple-set> conforms to that <attribute-set>, in the sense of paragraph 15. The rules for generating the <attribute-set> and <tuple-set> are explained in paragraphs 21-30 below.

21. The <union> operator is defined as follows. Let R(A1:D1,...,An:Dn) and S(B1:D1,...,Bn:Dn) be two <named-relation>s that are union-compatible with respect to the correspondence C = ((A1,B1),...,(An,Bn)). The <union> of R and S over this particular correspondence, denoted UNION(R,S,C), is an <unnamed-relation> in which:

(a) the <attribute-set> is (U1:D1,...,Un:Dn), where each Ui is (arbitrarily) that one of the pair (Ai,Bi) that precedes the other in lexicographic ordering;

(b) the <tuple-set> is defined as follows. The <tuple> (U1:u1,...,Un:un) appears in the <tuple-set> if and only if the <tuple> (A1:u1,...,An:un) appears in R or the <tuple> (B1:u1,...,Bn:un) appears in S (or both).

22. See the definition of union-compatibility, earlier.

23. See the definition of union-compatibility, earlier.

24. The <difference> operator is defined as follows. Let R(A1:D1,...,An:Dn) and S(B1:D1,...,Bn:Dn) be two <named-relation>s that are union-compatible with respect to the correspondence C = ((A1,B1),...,(An,Bn)). The <difference> of R and S over this particular correspondence, denoted DIFFERENCE(R,S,C), is an <unnamed-relation> in which:

(a) the <attribute-set> is (A1:D1,...,An:Dn);

(b) the <tuple-set> is defined as follows. The <tuple> (A1:a1,...,An:an) appears in the <tuple-set> if and only if it appears in R and the <tuple> (B1:a1,...,Bn:an) does not appear in S. Note that the <difference> of S and R over C, DIFFERENCE(S,R,C), is also defined.

25. The <product> of two differently-named <named-relation>s R(A1:D1,...,Am:Dm) and S(B1:E1,...,Bn:En), denoted PRODUCT(R,S), is an <unnamed-relation> in which:

(a) the <attribute-set> is (RA1:D1,...,RAm:Dm,SB1:E1,..., SBn:En) - note the attribute renaming, which guarantees that <attribute-name>s in the <attribute-set> of the <product> are unique;

(b) the <tuple-set> is defined as follows. The <tuple> (RA1:a1,...,RAm:am,SB1:b1,...,SBn:bn) appears in the <tuple-set> if and only if the <tuple> (A1:a1,...,Am:am) appears in R and the <tuple> (B1:b1,...,Bn:bn) appears in S.

Note 1: The <product> as defined above represents what is sometimes called the extended Cartesian product.

Note 2: We assume the existence of an aliasing operator, which permits the introduction of a new <relation-name> for a given <named-relation>. Such an operator is discussed informally in [5]. If it is necessary to form the <product> of some <named-relation> R with itself, the aliasing operator can be used to create an alias, S say, for R, so that the <product> can still be expressed in terms of two different <relation-name>s.

26. The <theta-selection> operator is defined as follows. Let R(A1:D1,...,An:Dn) be a <named-relation>. Let theta denote any one of the comparison operators =, ¬=, <, <=, >, and >=. Let theta-comparison denote any comparison of the form "Ai theta vi", where vi is either a <domain-value> from Di or is another <attribute-name> Aj of R also defined on Di (i,j = 1,...,n). (We assume that theta is applicable to Di.) The <theta-selection> of R with respect to this particular theta-comparison, THETA_SELECT(R, Ai theta vi), is an <unnamed-relation> in which:

(a) the <attribute-set> is (A1:D1,...,An:Dn);

(b) the <tuple-set> is defined as follows. The <tuple> (A1:a1,...,An:an) appears in the <tuple-set> if and only if it appears in R and "Ai theta vi" evaluates to true for this <tuple>.

27. See paragraph 26.

28. See paragraph 26.

29. See paragraph 26.

30. The <projection> operator is defined as follows. Let R(A1,...,An) be a <named-relation>, and let (Ai,...,Aj) be a subset of the <attribute-name>s specified in the <attribute-set> of R. The <projection> of R over this subset, denoted PROJECT(R,(Ai,...,Aj)), is an <unnamed-relation> in which:

(a) the <attribute-set> is (Ai:Di,...,Aj:Dj), where each Dk (k = i,...,j) is the Dk corresponding to Ak in the <attribute-set> of R;

(b) the <tuple-set> is defined as follows. The <tuple> (Ai:ai,...,Aj:aj) appears in the <tuple-set> if and only if a <tuple> exists in the <tuple-set> of R having ai as its Ai-value, ..., aj as its Aj-value.

31. The <relational-assignment> operator is defined as follows. The <relation-name> must be the <name> of a <real-relation>, R say. Let R have <attribute>s A1, ..., An. Let X be a <relational-expression> and let C be a <correspondence>. The <relational-assignment> of X to R in accordance with C, denoted ASSIGN(R,X,C), causes the existing <tuple-set> of R to be replaced by a new <tuple-set>, as follows:

(a) X is evaluated (see paragraph 32) and generates an <unnamed-relation>. Let us refer to this <unnamed-relation> as S, and let the <attribute>s of S be B1, ..., Bn.

(b) R and S must be union-compatible with respect to C. Let corresponding <attribute>s of R and S under C be (A1,B1),...,(An,Bn).

(c) The <tuple> (A1:a1,...,An:an) appears in the new <tuple-set> of R if and only if the <tuple> (B1:a1,...,Bn:an) appears in the <tuple-set> of S.

We remark that a given <relational-assignment> will fail if the would-be new <tuple-set> of R violates any integrity constraints that are applicable to R.

32. A <relational-expression> is a <relation-name>, a <relational-algebra-operation>, or a <relation-literal>. The result of evaluating a <relational-expression> is to generate an <unnamed-relation>, as explained earlier. In the case of a <relation-name>, the <unnamed-relation> has an <attribute-set> and a <tuple-set> identical to those of the designated <named-relation>. In the case of a <relational-algebra-operation>, it has an <attribute-set> and a <tuple-set> that are derived in accordance with paragraphs 21-30 above. In the case of a <relation-literal>, the <unnamed-relation> has an <attribute-set> and a <tuple-set> as specified by that <relation-literal> (see paragraph 33).

33. A <relation-literal> consists of an <attribute-set> and a set of <tuple>s. Each <tuple> in the <tuple-set> must conform to the <attribute-set>. Intuitively, <relation-literal>s permit the insertion of new <tuple>s into the <relational-database> (via a <union> operation), and the deletion of existing <tuple>s from the <relational-database> (via a <difference> operation).

RELATIONAL ALGEBRA: ADDITIONAL OPERATORS

Our formal definitions have deliberately restricted themselves to a very primitive (but relationally complete [4]) set of operations. In practice, a relational implementation should provide various refinements on this primitive version, for both usability and efficiency. We consider some such refinements very briefly.

• Intersection, join, division

The relational algebra as usually defined includes not only the operators defined above, but also the operators intersection, join, and division [2]. We excluded these latter operators from our formalism because they are not true primitives - each can be defined in terms of the other operators, as shown in [5]. However, the join operator is so important in practice that we give a definition of it here, for purposes of reference. (Actually, as we shall see, the term "join" is used generically to refer to several related but distinct operators.) Let relations R(A1,...,Ai:D,...,Am) and S(B1,...,Bj:D,...,Bn) be such that attributes Ai and Bj are defined on the same domain D. Let theta be any one of the comparison operators =, ¬=, <, <=, >, and >= that is applicable to domain D. The theta-join of R on Ai with S on Bj, THETA_JOIN(R, S, Ai theta Bj), is an unnamed-relation with attribute-set as for PRODUCT(R,S). The tuple-set in this relation consists of all tuples

(RA1:a1,...,RAi:ai,...,RAm:am,SB1:b1,...,SBj:bj,...,SBn:bn)

such that (A1:a1,...,Ai:ai,...,Am:am) appears in the current tuple-set of R, (B1:b1,...,Bj:bj,...,Bn:bn) appears in the current tuple-set of S, and "ai theta bj" evaluates to true. (The theta-join of R and S is thus obtained by taking the product of R and S and then applying a theta-selection to it.)

If theta is equality, the theta-join is said to be an equijoin. In the case of the equijoin, the projections of the result on Ai and on Bj are necessarily identical. Let C be the set of all attributes of the equijoin except Ai (or except Bj). The projection of the equijoin on C is called the natural join of R and S (on Ai and Bj).

• Extended and combined operations

It is desirable to provide shorthand equivalents for commonly-occurring sequences of the primitive operations. We note some possible shorthands here:

- allow the operands of the relational-algebra-operations to be specified as arbitrary relational-expressions, enclosed in parentheses if necessary to avoid ambiguity, instead of only as relation-names;

- introduce an extended "selection" operator in which the second operand is an arbitrary Boolean combination of simple theta-comparisons;

- introduce an analogously extended "join" operator;

- allow the target for relational-assignment to be a virtual-relation as well as a real-relation (for "updatable" virtual-relations);

- introduce explicit "insert," "delete," and "update" operations, instead of relying purely on relational-assignment and appropriate use of the union and difference operators.

• Operators to handle null values

The relational algebra as defined by Codd in [2] includes certain additional operators for dealing with null values - for example, a "maybe" version of THETA_SELECT that selects tuples for which the theta-comparison evaluates to unknown, rather than to true. As indicated earlier, however, we omit detailed discussion of null values - and their effect on the relational algebra, in particular - from this paper. See reference [7].

RELATIONAL RULES

As already indicated, the basic relational model includes two general integrity rules. (A specific relational-system will typically also include some specific "local" integrity rules.) We define the two general rules here (Fig. 3). Our definitions are in the spirit of reference [2]; regarding Rule 2 ("referential integrity"), however, it has been suggested that the formalism of [2] is not entirely satisfactory [8]. We content ourselves here with pointing out that Rule 2 is not a complete control in itself on the referential integrity problem; loosely speaking, it is "necessary but not sufficient" (that is, a database could conform to the requirements of Rule 2 and yet still be in violation of some referential constraint). Reference [6] contains a detailed discussion of this topic and presents a more general proposal for expressing referential constraints - a proposal, incidentally, that (unlike Rule 2 as defined here) does not require the notion of "primary domain" at all.

Note that the two rules refer to <real-relation>s only, not to <virtual-relation>s and not to <unnamed-relation>s.

Integrity Rule 1 (entity integrity)

• Let <attribute> A of <real-relation> R be a component of the <primary-key> of R. Then <attribute> A cannot accept null values. That is, no <tuple> of R can include an <attribute-value> in which the <attribute-name> is A and the <domain-value> is null.

formal definition of RDM

Integrity Rule 2 (referential integrity)

- Let <domain> D be such that there exists a <real-relation> with (single-attribute) <primary-key> defined on D. Then D may optionally be designated as a primary <domain>.

- Now let R(...,A:D,...) be a <real-relation> with <attribute> A defined on primary <domain> D. Then every <tuple> of R must satisfy the constraint that the <domain-value> corresponding to <attribute> A within that <tuple> must be either (a) null, or (b) equal to k, say, where k is the <domain-value> corresponding to the <primary-key> within some <tuple> of some <real-relation> S with <primary-key> defined on D. R and S need not be distinct. The <attribute> A of R is said to be a foreign key.

Fig. 3: The structure of a relational-system (part 3 of 3): the <relational-rule>s

CONCLUDING REMARKS

We conclude by showing informally how the definitions given earlier lead to certain well-known and important relational properties.

- Within a given relation, no two tuples are identical (at any given time). This follows from the fact that the tuples within the relation at any given time are defined to constitute a set, and sets by definition do not contain duplicate elements.

- As a consequence of the previous point, the requirement that relations - in particular, real relations - always have a primary key is a reasonable (i.e., consistent) one. This follows from the fact that at least the combination of all attributes has the uniqueness property, and hence that at least one candidate key (the set of all attributes, if necessary) always exists.

- Within a given relation (more accurately, within the tuple-set of a given relation), tuples are unordered. This also follows from the fact that a relation is a set - sets by definition have no ordering.

- Within a given relation, attributes are unordered. This follows from the fact that the collection of attributes of the relation is also defined as a set.

- All attribute-values are atomic; in other words, all relations are normalized. Equivalently, all relations are in first normal form. This level of normalization is required by the relational algebra. Higher levels, such as 4NF [10], are useful for database design purposes but are not an intrinsic part of the relational model per se.

ACKNOWLEDGMENTS

I am grateful to Addison-Wesley Publishing Company for permission to base this paper on material that is due to appear in the forthcoming book "Developments in Database Technology" (tentative title). I am also grateful to my colleagues Ted Codd, Bill Kent, and Phil Shaw for numerous constructive comments on earlier drafts.

POSTSCRIPT

After the body of this paper was written, my attention was drawn to the final report of the ANSI Relational Task Group [11], which contains as an appendix a formal definition of the relational model that is similar in some respects to that presented above.

formal definition of RDM

REFERENCES

1. E.F.Codd. "A Relational Model of Data for Large Shared Data Banks." CACM 13, No. 6 (June 1970).

2. E.F.Codd. "Extending the Database Relational Model to Capture More Meaning." ACM TODS 4, No. 4 (December 1979).

3. E.F.Codd. "Data Models in Database Management." ACM SIGMOD Record 11, No. 2 (February 1981).

4. E.F.Codd. "Relational Completeness of Data Base Sublanguages." In Data Base Systems, Courant Computer Science Symposia Series, Vol. 6. Prentice-Hall (1972).

5. C.J.Date. "An Introduction to Database Systems." Addison-Wesley (3rd edition, 1981).

6. C.J.Date. "Referential Integrity." Proc. 7th International Conference on Very Large Data Bases (September 1981).

7. C.J.Date. "The Problem of Null Values." To appear.

8. W.Kent. Private communication (May 1981).

9. M.Lacroix and A.Pirotte. "Domain-Oriented Relational Languages." Proc. 3rd International Conference on Very Large Data Bases (October 1977).

10. R.Fagin. "Normal Forms and Relational Database Operators." Proc. 1979 ACM SIGMOD International Conference on Management of Data.

11. ANSI/X3/SPARC DBS-SG Relational Database Task Group. Final Report (eds., Brodie and Schmidt). September 1981.

formal definition of RDM