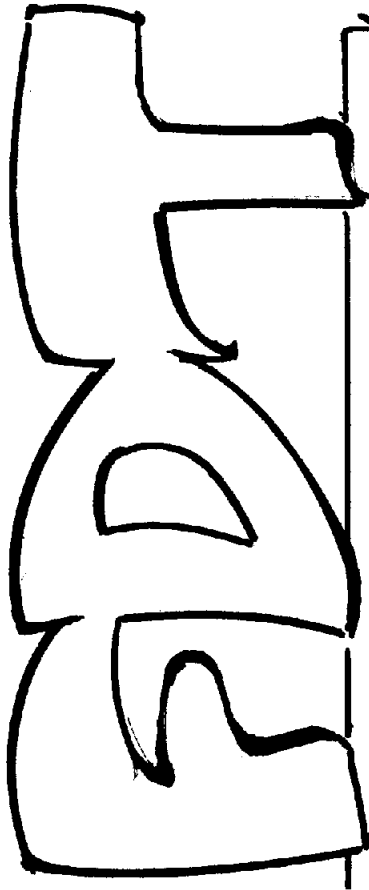




FDT ... BULLETIN OF ACM - SIGMOD  
THE SPECIAL INTEREST GROUP ON  
MANAGEMENT OF DATA .....  
VOLUME 7, NUMBER 1 .....1975

<i>Understanding Relations (E. G. Codd)</i>	1
<i>Report on IFIP TC-2 Conference: A Technical In-Depth Evaluation of the DDL (Robert W. Taylor)</i>	5
<i>A Graduate Course in Database Management (J. Gerry Purdy)</i>	10
<i>For FDT Reader's Information: National Computer Conference</i>	16
<i>IFIP TC-Working Conference: Modelling in Database Management Systems</i>	18
<i>Conference on Data: Abstraction Definition and Structure</i>	20





is a publication of ACM's Special Interest Group on File Description and Translation. It is published irregularly three times a year in Oakland, California. Technical papers appearing in this issue are unrefereed working papers. Additional copies at \$2.00 each may be obtained from ACM Headquarters, 1133 Avenue of the Americas, New York, NY 10036 (Telephone (212) 265-6300.)

Inquiries concerning SIGFIDET membership should be sent either to ACM Headquarters or to SIGFIDET's Chairman, Kendall R. Wright, IBM Development Laboratories, Monterey and Cottle Roads, San Jose, CA 95193.

(A form included inside the back cover of this issue may be used for inquiry, or for change of address.)

Material submitted for publication in FDI should be sent to the current Editor's successor (e.g., "New FDI Editor, ...") in care of our Chairman -- whose address is given above. Manuscripts should be submitted in camera-ready form, in duplicate. The usual conventions about typing, references, illustrations and footnotes will apply.

Editor

Daniel O'Connell,  
Oakland, California

SIGFIDET  
Chairman

Kendall R. Wright,  
San Jose, California

## UNDERSTANDING RELATIONS

(Installment #6)

E. F. Codd

Q1 Some critics of the relational model claim that it is useful for query and report generation only. In particular, they claim that this model cannot represent natural insertion and deletion properties of data (see for example the Proceedings of the ACM-SIGMOD Debate, May 1974). What is your answer to these critics?

A1 Let us focus on the rather specific criticism concerning insertion and deletion. It is easy to overlook the significance of primary, candidate, and foreign keys in the relational model. There are important integrity principles associated with these concepts, and these principles in turn determine broadly-applicable rules governing insertion and deletion in the relational model -- rules which, of course, may have to be augmented in various specialized ways to meet the needs of specific installations.

The candidate key concept, you may recall, is defined as follows: each candidate key  $K$  of relation  $R$  is a combination of attributes (possibly a single attribute) of  $R$  with the following properties:

unique identification -- in each tuple of  $R$  the value of  $K$  uniquely identifies that tuple;

non-redundancy -- no attribute in  $K$  can be discarded without destroying the unique identification property.

A basic integrity principle associated with candidate keys is that, for every base relation, at least one of the candidate keys is prohibited from taking on null values. Suppose this integrity principle were not applied. Suppose further that an EMPLOYEE relation has been established with the intent that for every employee there should be one and only one EMPLOYEE tuple describing him. The EMPLOYEE relation has attributes  $E\#$  (employee identification number),  $SOC\#$  (social security number),  $LAST\ NAME$ , and other attributes. The only candidate keys are  $E\#$  and  $SOC\#$ . An employee named Jones is hired. He has 1234 as his social security number. The Personnel Department for some reason does not immediately assign an employee identification number. Initial information about Jones is entered into the data base relation EMPLOYEE by inserting a tuple having a null value (denoted by "?") for  $E\#$ , 1234 for  $SOC\#$ , and JONES for  $LAST\ NAME$ . Now, suppose Jones is to be assigned 7732 as his  $E\#$ , but by mistake a second tuple is inserted

in which E# is 7732, SOC# is null, and LAST\_NAME is JONES. The resulting data base state is:

```
EMPLOYEE ( E#  SOC#  LAST_NAME  . . . )
          ?    1234   JONES      . . .
          7732   ?    JONES      . . .
```

The problem with this data base state is that it is not clear now whether these two tuples represent one and the same person or whether they represent two distinct employees who happen to have the same name. If a count of the tuples in EMPLOYEE were now executed, it could yield an incorrect answer to the query "How many employees are there?". Normally, it will not be necessary to prohibit null values in more than one candidate key -- hence, the common practice of designating precisely one such key as the primary key: i.e., the only candidate key for which null values are prohibited. The consequences for insertion and deletion are immediately obvious: no tuple can be inserted into a relation unless its primary key value (possibly a compound value) is not null and not equal to that of the primary key value in any other tuple in that relation.

Just as the primary and candidate key concepts provide basic insertion and deletion constraints within each relation, the foreign key concept provides such constraints between relations. Now, attribute combination C of relation R is a foreign key with respect to attribute combination D of relation S if:

1. D is a null-free candidate key for S; and
2. The projection of R on C (ignoring any null values which may be present in C) is included in the projection of S on D, regardless of time.

In such a case we shall call D in S the target of C in R. Incidentally, this definition differs from the original one mainly in regard to paying more attention to null values. As an example consider a personnel data base containing the following two relations:

```
ASSIGN ( E#  D#  ASSIGNED_DATE  )
DEPT   ( D#  HEADCOUNT  )
```

where E# denotes employee identification number, D# denotes department number, and an employee may be assigned to at most one department at a time (temporarily to none). All of the non-null values in the ASSIGN.D# column must appear in the DEPT.D# column. Null values may appear in ASSIGN.D#, but not in DEPT.D#. Thus, in this example, ASSIGN.D# is a foreign key with respect to DEPT.D#. The integrity rule associated with foreign keys is simply that, at no time,

can a foreign key value exist which is both non-null and fails to appear in the target attribute(s). The resulting insertion-deletion constraints are obvious: in the example above, a new ASSIGN tuple with a non-null D# component may not be inserted into the relation ASSIGN unless and until its D# component has been established in DEPT.D#. Likewise, no tuple may be deleted from DEPT.D# unless and until all references to it by foreign keys have been removed (either by modification or deletion of the tuples containing those references).

Clearly, there is a good deal more to be investigated and reported about insertion-deletion semantics in the relational model -- and in other models. An interesting contribution will be presented by Schmid and Swenson of the University of Toronto at the 1975 ACM-SIGMOD Workshop in San Jose.

- Q2 Is it necessary that every relation have at least one null-free candidate key?
- A2 No. For the integrity reason cited above (see A1) it is important that every base relation have at least one candidate key from which null values are prohibited. Generally speaking, however, such a constraint would be unnecessary for derived relations, whether they are materialized as static snapshots or manipulated as views with dynamically changing extensions.
- Q3 Due to the absence of explicit links in the relational model, it appears that, whenever a tuple is deleted from a relational data base, it becomes necessary to scan the entire data base to see if there are any references still extant to the primary key of this tuple. Doesn't this result in intolerably poor performance?
- A3 The confusion on this point seems to be widespread. It arises from a failure to keep the relational model clearly distinguished from its many possible implementations. An implementation can exploit various types of fast access paths between foreign key values and their counterparts in primary keys (IDS chaining is but one example of an access path that might be used for this purpose). The fact that these access paths are invisible to the relational user does not make these paths ineffective! They still obviate the need for widespread searching for foreign key cross-references.
- Q4 Can insertion-deletion dependencies other than those implied by keys be expressed in the relational model? Can integrity constraints of more general kinds than insertion-deletion dependencies be handled?

- A4 Yes, in two ways: by extending a high-level, relationally complete query language to permit integrity assertions or by employing data base procedures written in a general purpose procedural language. The first of these approaches is discussed in the Boyce-Chamberlin paper entitled "Using a Structured English Query Language as a Data Definition Facility" (IBM Research RJ1318) and in the new book "An Introduction to Database Systems" by C. J. Date (Addison-Wesley). It has been implemented in the experimental system SEQUEL. It can cope with a wide variety of integrity constraints and offers, in contrast to using a procedural language, a clear, concise, and reliable way to specify such constraints. Nevertheless, for true generality the system must provide the computational completeness of data base procedures. These are procedures that are not part of any application program, neither are they explicitly invoked from any application program. Execution of a data base procedure is triggered whenever a transaction of some specified type is attempted.
- Q5 When the data base administrator (DBA) is saddled with the development of data base procedures, isn't this tantamount to relabelling the application programmers "DBA staff"? There seems to be no saving of effort.
- A5 The whole purpose of data base procedures and any other facilities for specifying integrity constraints is to express and enforce these constraints in a centralized manner, not relying upon voluntary compliance by individual application programs or terminal activities. While there could be a saving of effort in specifying these constraints once only, any such saving is of minor importance compared with the increase in dependability of the data.

Answer to Problem for Readers (see Installment #3,4,5)

The converse of Heath's theorem is false and this suggests that there is no neat way of characterizing the non-decomposability property in terms of functional dependencies. However, we may state this property as follows: for each of the cited time-varying relations (BETWEEN, FLIGHT, and SUPPLY) there exists no pair of binary projections which are re-joinable to yield the original ternary relation regardless of time.

Correspondence should be addressed to:

Dr. E. F. Codd  
IBM Research Laboratory K51-282  
Monterey & Cottle Roads  
San Jose, California 95193

All rights reserved