A New Normal Form for the Design of Relational Database Schemata

CARLO ZANIOLO
Sperry Research Center

This paper addresses the problem of database schema design in the framework of the relational data model and functional dependencies. It suggests that both Third Normal Form (3NF) and Boyce-Codd Normal Form (BCNF) supply an inadequate basis for relational schema design. The main problem with 3NF is that it is too forgiving and does not enforce the separation principle as strictly as it should. On the other hand, BCNF is incompatible with the principle of representation and prone to computational complexity. Thus a new normal form, which lies between these two and captures the salient qualities of both is proposed. The new normal form is stricter than 3NF, but it is still compatible with the representation principle. First a simpler definition of 3NF is derived, and the analogy of this new definition to the definition of BCNF is noted. This analogy is used to derive the new normal form. Finally, it is proved that Bernstein's algorithm for schema design synthesizes schemata that are already in the new normal form.

Categories and Subject Descriptors: H.2.1 [**Database Management**]: Logical Design—*normal forms*

General Terms: Algorithms, Design, Theory

Additional Key Words and Phrases: Relational model, functional dependencies, database schema

1. INTRODUCTION

The concept of normal form has supplied the cornerstone for most of the formal approaches to the design of relational schemata for database systems. Codd's original Third Normal Form (3NF) [9] was followed by a number of refinements. These include Boyce-Codd Normal Form (BCNF) [10], Fourth Normal Form [12], and Fifth Normal Form [13]. The referenced works have greatly contributed to bettering our understanding of the properties of relational schemata and of semantic constraints in database relations. However, the proposed normal forms suffer from a number of drawbacks. In particular, they do not seem conducive to efficient algorithms for schema design [3], and their main motivation is the elimination of update anomalies, which constitutes an elusive [18] and possibly unattainable [7] objective.

In contrast, other formal approaches [6, 18] have proposed, as a formal objective for schema design, the complete *representation* of semantic constraints. When

Author's address: Bell Laboratories, Holmdel, NJ 07733.

dealing with formal constraints, such as functional or multivalued dependencies, this objective can be formulated with mathematical rigor [4, 18]. Moreover, Bernstein [6] proposed an algorithm to design schemata consisting of 3NF relations such that the set of functional dependencies of interest is completely represented by the keys of the schema relations.

Unlike 3NF, BCNF (and therefore Fourth Normal Form and Fifth Normal Form) is not well suited for designing schemata according to the representation principle. Indeed, as we see later, there are cases in which a given set of FD's cannot be represented by the keys of a BCNF schema. Moreover, even when such a BCNF schema exists, there are good reasons to believe that finding it is computationally very hard [3]. Therefore, as noted in [3] and [4], the historical transition from 3NF to BCNF cannot be simplistically labeled an improvement, but must instead be regarded as a step which, along with some advantages, has brought significant problems.

In this paper we introduce a simpler definition of 3NF, which clarifies the relationships between 3NF and BCNF. Then we introduce a normal form which is between 3NF and BCNF and possesses some of the better qualities of both. In particular, this new normal form is well suited for designing schemata according to the representation principle. We prove that Bernstein's schema design algorithm [6] produces schemata that satisfy this new definition.

The new normal form is developed in the framework of functional dependencies. Its extension to the more general framework of multivalued dependencies [17, 12] and join dependencies [13, 15, 16] will be discussed in future reports.

2. RELATIONS AND FUNCTIONAL DEPENDENCIES

In the relational approach the database is viewed as a set of *relations* of time-varying content [9, 11]. A relation R with attribute set $\{A_1, A_2, \ldots, A_n\}$ is denoted $R(A_1, A_2, \ldots, A_n)$. To each attribute A_i there corresponds an underlying domain denoted $\mathrm{DOM}(A_i)$. An instance of R, that is, its content at a certain instant in time, is defined as a subset of the Cartesian product $\mathrm{DOM}(A_1) \times \cdots \times \mathrm{DOM}(A_n)$. This instance can be represented as a table having the elements of this subset as rows and the attributes of R as columns. If r is a row of this table (i.e., an element in an instance of R), then $r[A]$ denotes the value of this row in the A-column; $r[A]$ is called the A-value of r. Likewise, if X is a subset of the attribute set of R, then the X-value of r is the subrow of r, of length $|X|$, whose B-value, for each $B \in X$, is equal to $r[B]$. We use the letters A, B, and C to denote single attributes and the letters X, Y, W, and Z to denote sets of attributes.

A schema consists of a set of relations, where each relation is defined by its attribute sets and some semantic constraints. In this paper we restrict our attention to constraints which can be expressed as *functional dependencies* (FDs). Let R be a relation and let X and Y be nonempty subsets of its attributes. An instance of R is said to obey the FD $f\colon X \rightarrow Y$, when every two rows of this instance that have identical X-values also have identical Y-values. The FD of R, $f\colon X \rightarrow Y$, is a statement which specifies that every instance of R must obey f. If f is not an FD of R, we write $X \not\rightarrow Y$.

The existence of certain FDs in a relation implies the existence of others (i.e., no instance of R can exist which obeys the former FDs but not the others).

Inference rules are means to construct these implied dependencies. A system of inference rules is said to be *complete* when every implied dependency can be derived by (repeated) applications of these rules.

The following is a complete system of inference rules for functional dependencies [2, 5]:

$F1$ (Reflexivity): If $Y \subseteq X$, then $X \rightarrow Y$.
$F2$ (Augmentation): If $Z \subseteq W$ and $X \rightarrow Y$, then $XW \rightarrow YZ$.
$F3$ (Transitivity): If $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$.

Other useful rules can be derived from these. For instance,

$F4$ (Pseudotransitivity): If $X \rightarrow Y$ and $YW \rightarrow Z$, then $XW \rightarrow Z$.
$F5$ (Decomposition): If $X \rightarrow Y$, then $X \rightarrow A$ for every A in Y.

Because of $F5$, we assume, without loss of generality [2], that all the FDs have only one attribute at their right side. The FD $X \rightarrow A$ is called *nontrivial* when A is not an element of X. Note that a trivial FD is obeyed by all instances.

Let F be a set of functional dependencies. The set of FDs which consists of the FDs in F, plus those which are derivable from these by repeated applications of the previous inference rules, is called the *closure* of F denoted F^+. Thus, f is implied by F if and only if it belongs to F^+.

Following [3], we define a *derivation* of f from F to be a sequence of FDs $[f_1, \ldots, f_n]$ such that $f_n = f$ and for each i, $1 \leq i \leq n$, one of the following holds:

(1) f_i is in F, or
(2) f_i is the result of invoking $F1$, or
(3) f_i is the result of applying $F2$ to one of the $f_1, \ldots, f_{i-1}$, or
(4) f_i is the result of applying $F4$ to two of the FDs $f_1, \ldots, f_{i-1}$.

The occurrence of an FD, f, in the derivation of an FD, g, is said to be *redundant* if it is possible to eliminate this occurrence of f (and possibly some other FDs) from the derivation and obtain a derivation of the same FD, g. Since $F1$, $F2$, and $F4$ constitute a complete set of inference rules it follows that $f \in F^+$ if and only if there exists some derivation of f from F. The following lemma, proved in [6] and [2], is used later.

LEMMA 1. *Let F be a set of FDs, and let f: $X \rightarrow Y$ be in F. If f is used nonredundantly in some derivation from F of an FD g: $Z \rightarrow W$ in F^+, then $Z \rightarrow X$ is also in F^+.*

A *cover* of F is any set of FDs that has the same closure as F. An FD, $f \in F$, is *redundant* if it is implied by $(F - \{f\})$. A cover is redundant if and only if it contains a redundant FD. Thus the algorithm for constructing a nonredundant cover for F simply eliminates redundant FDs from F, until no more such FDs can be found. The core of this algorithm is a test to decide whether a member $f \in F$ is redundant, that is, to check whether f belongs to $(F - \{f\})^+$. A linear-time algorithm to perform this test is given in [3]. Thus the previous algorithm for finding a nonredundant cover is quadratic-time.

Let R be a relation with attribute set U. $X \subseteq U$ is called a *key* for R when U is functionally dependent on X but not on any subset of X. The importance of the

concept of key in the relational approach cannot be overemphasized. First of all, keys serve as unique identifiers of rows in relations, thus ensuring content-addressability of the database. This has a strong bearing on the relational data manipulation languages and also on the storage structures used to support data access at the implementation level. Finally, most data definition languages will include constructs to allow explicit declaration of keys but not explicit declaration of FDs. Thus, keys are means to define and represent to a user the presence of FD constraints. If X is a key for a relation R, and A is an attribute of R not in X, then $X \rightarrow A$ is said to be *embodied* in R. The set of FDs *represented* by a schema is defined as the closure of the FDs embodied in schema relations. An attribute participating in some key will be called a *key attribute.*[1] Normal form definitions are based on the concept of keys. Following [6], we refer to any superset, proper or otherwise, of a key of R as a *superkey* of R.

3. NORMAL FORMS

Let R be a relation with attribute set U, let $X \subseteq U$, and let $A \in U$. A is *transitively dependent* on X if there exists a $Y \subseteq U$ such that $X \rightarrow Y$, $Y \not\rightarrow X$, $Y \rightarrow A$, and $A \notin Y$. The definition of Third Normal Form can be stated as follows [6]:

Definition 1. A relation is 3NF if every attribute transitively dependent on a key is a key attribute.

Note that inherent in the definition of 3NF is a distinction between key attributes and nonkey attributes. This distinction was removed with the definition of Boyce-Codd Normal Form [10].

Definition 2. A relation R is BCNF if for every nontrivial FD of R, $X \rightarrow A$, X is a superkey for R.

Every relation which is BCNF is also 3NF, but not vice versa. To illustrate the difference between these two definitions, we next consider two simple examples taken from [17].

The first example concerns departments (denoted by their number D#), their managers (denoted by their badge ID, MGID), and accounts (denoted by their number ACC#). We assume that there is a one-to-one correspondence between MGID and D# (one distinct manager per department). Moreover, there is a many-to-many correspondence between ACC# and D# (or MGID), as a department will have many accounts, some shared among departments (e.g., those relating to overhead costs). Then, we have a relation, let us call it DA, with attributes D#, MGID, and ACC#, having the following FDs:

$f1$: D# $\rightarrow$ MGID
$f2$: MGID $\rightarrow$ D#.

For this example, one might choose the following schema (the key attributes are underlined and different styles of underlining denote distinct keys):

$$\text{DA}(\underline{\text{D\#}}, \underline{\text{MGID}}, \underline{\text{ACC\#}}) \tag{3.1}$$

[1] Key attributes are often called prime attributes.

Thus, DA has two keys: one is (D#, ACC#), the other is (MGID, ACC#). Since every attribute is a key attribute, DA is 3NF. Nevertheless, it presents some obvious problems. First of all, (3.1) does not represent $f1$ and $f2$. This schema only embodies the weaker constraints (D#, ACC#) → MGID and (MGID, ACC#) → D#. Therefore, the constraints which follow from the one-to-one correspondence between D# and MGID are not represented by this schema. Moreover, those readers who like to work in terms of update anomalies [9] will find quite a few of them here (e.g., if keys cannot be assigned null values, then information on departments and their managers cannot be stored before these are assigned some accounts). Most remarkably, these problems disappear once (3.1) is replaced by (3.2) and (3.3) below:

$$\text{DA1}(\underline{\text{D\#}}, \underline{\text{MGID}}) \tag{3.2}$$

$$\text{DA2}(\underline{\text{D\#}}, \underline{\text{ACC\#}}) \tag{3.3}$$

Note that this new schema represents $f1$ and $f2$ and all the relationships of interest. If BCNF is used, then the designer would have to replace (3.1), which is not BCNF, by the pair (3.2), (3.3). This is a much better schema, separately representing the one-to-one relationship between D# and MGID and the many-to-many relationship between D# and ACC# by two independent relations. Thus, this example typifies the case in which BCNF performs better than 3NF because it implements the principle of *separation* to a stricter degree than 3NF does [4].

Our second example consists of a regional list of telephone numbers. We have a relation TEL with attributes AREA, NUMBER, and PLACE, where AREA and NUMBER denote, respectively, the area code and the telephone number, and PLACE denotes the location (e.g., the town) of this telephone. Obviously, the area code and number identify the location, but also the location identifies the area code. Thus, we have the following FDs:

$f3$: (AREA, NUMBER) → PLACE
$f4$: PLACE → AREA.

The following 3NF schema could be used for this example:

$$\text{TEL}(\text{PLACE}, \underline{\text{AREA}}, \underline{\text{NUMBER}}). \tag{3.4}$$

This schema is not satisfactory since it represents $f3$ but not $f4$. If BCNF schemata are used, then (3.4), which is not BCNF, will have to be replaced by the pair

$$\text{TEL1}(\underline{\text{PLACE}}, \text{AREA}) \tag{3.5}$$

$$\text{TEL2}(\underline{\text{PLACE}}, \underline{\text{NUMBER}}). \tag{3.6}$$

This second schema is not acceptable either, since while it represents $f4$ it does not represent $f3$. Thus, neither a schema consisting of (3.4) nor one consisting of (3.5) and (3.6) is satisfactory according to the representation principle. A solution compatible with the representation principle was proposed in [6]. This includes both (3.4) and (3.5) in the schema, thus embodying both $f3$ and $f4$.

The previous discussion illustrates that BCNF, unlike 3NF, is incompatible with the representation principle. 3NF is also superior to BCNF with respect to algorithmic design of relational schemata. While an efficient algorithm to design a 3NF schema which represents a given set of FDs is available [6], no efficient algorithm is known for finding a BCNF schema which represents these FDs (even when such a schema exists). Moreover, there are strong indications that the problem is computationally difficult [3]. Therefore, it has been suggested in [3] that 3NF, and not BCNF, should be used in the design of relational schemata. However, as the first example shows, 3NF is often too forgiving and will pass bad relations which, instead, should be further decomposed. In this paper we propose a new normal form which is stricter than 3NF but preserves its qualities with respect to the representation principle and computational simplicity.

4. A NEW NORMAL FORM

Say that G denotes a set of FDs and that $f: X \rightarrow A$ is an element of G. Then f will be said to be *elementary* with respect to G, when A is not in X and G^+ does not contain an FD, $X' \rightarrow A$, with $X' \subset X$. An elementary FD for a relation R is one which is elementary with respect to the FDs of R. Elementary FDs have a simple structure and a number of formal properties useful in schema design [18]. The definition of BCNF can be reformulated in terms of elementary FDs.

LEMMA 2. *A relation R is BCNF iff for every elementary FD of R, say, $X \rightarrow A$, X is a key of R.*

PROOF. Easy.

We now have the following two equivalent formulations of 3NF.

LEMMA 3. *A relation R is 3NF iff for every nontrivial FD of R, $X \rightarrow A$,*

(*a*) *X is a superkey for R, or*
(*b*) *A is a key attribute for R.*

PROOF. Let $X \rightarrow A$ be a nontrivial FD and let A be a nonkey attribute. Also let Y be a key of R. Then, $Y \rightarrow X$. Therefore, A is not transitively dependent on Y iff $X \rightarrow Y$, that is, iff X is a superkey of R. Q.E.D.

LEMMA 4. *A relation R is 3NF iff for every elementary FD of R, say, $X \rightarrow A$,*

(*a*) *X is a key for R, or*
(*b*) *A is a key attribute for R.*

PROOF. Easy.

The preceding lemmas provide new and simpler definitions of 3NF. Most important, they reveal an analogy between the definitions of 3NF and BCNF which was not previously known: the two definitions are identical except for (b), which waives condition (a) for key attributes. Thus, if we keep condition (a) unchanged but relax (b), we obtain a normal form definition which is stricter than 3NF but weaker than BCNF.

Observe now schema (3.4) with keys (AREA, NUMBER) and (PLACE, NUMBER). While the first key defines the elementary FD, (AREA, NUMBER) → PLACE, the second does not define any elementary FD. The only nontrivial FD represented by this key is (PLACE, NUMBER) → AREA, which is *not* elementary. This key does not contribute much to definitions of the semantic constraints of this relation since it does not represent the elementary FD, PLACE → AREA. This elementary FD must be represented separately by including (3.5) in the schema. Once this is done, the key (PLACE, NUMBER) becomes expendible in terms of data definition, since every FD represented in the schema can be derived without it. Therefore, let us define the concept of *elementary key.* A key X of a relation R is said to be elementary if, for some attribute A in U, $X \rightarrow A$ is an elementary FD of R. An attribute which belongs to some elementary key is called an *elementary key attribute.* We can now define our new normal form by making a distinction between elementary keys and nonelementary keys (which, as shown above, are expendable in terms of data definition.)

Definition 3. A relation R is *Elementary Key Normal Form* (EKNF) if for every elementary FD of R, say, $X \rightarrow A$,

(a) X is a key for R, or
(b) A is an elementary key attribute for R.

This definition illustrates the importance of elementary FDs and elementary keys for normal forms. Note that if $X \rightarrow A$ is elementary and condition (a) above is satisfied, then X is an elementary key. Similar considerations can be made regarding 3NF (Lemma 4) and BCNF (Lemma 2). It is also easy to define EKNF using the concept of superkey:

Lemma 5. *A relation R is EKNF iff for every nontrivial FD in R,* $X \rightarrow A$,

(*a*) *X is a superkey of R, or*
(*b*) *A is an elementary key attribute for R.*

Clearly, every relation which is BCNF is EKNF, and every relation which is EKNF is 3NF. The previous examples illustrate the differences between these normal forms more concretely. Take, for instance, our relation DA (3.1). This relation has no elementary key and, therefore, no elementary key attribute. Now, if we consider the elementary FD: D# → MGID, we find that D# is not a key, nor is MGID an elementary key attribute. Thus (3.1) is not EKNF. Therefore, if EKNF is used in schema design, (3.1) will be replaced by the pair (3.2) and (3.3), which is a much better schema. In relation TEL, instead, we find the elementary FDs, *f*3: (AREA, NUMBER) → PLACE and *f*4: PLACE → AREA, and the elementary key (AREA, NUMBER). Thus, the left side of *f*3 is an (elementary) key for TEL. Moreover, the right side of *f*4 is an elementary key attribute. Thus (3.4) is EKNF and so is the combined schema (3.4), (3.5). As we know, relation (3.4) is not BCNF.

In conclusion, we see that the new normal form is stricter than 3NF and implements the principle of separation [4] much better than does 3NF. At the same time, EKNF is, unlike BCNF, compatible with the representation principle;

in the next section, we prove that Bernstein's algorithm [6], which is known to produce 3NF schemata, does actually produce EKNF schemata.

5. SCHEMA DESIGN

Bernstein proposed an algorithm where, given a set of FDs, G, a relational schema S is produced having the following properties [6]:

(1) *Complete Representation.* The schema S represents all the FDs in G.
(2) *Minimality.* No schema with fewer relations represents G.
(3) *3NF property.* Every relation in S is 3NF.

This algorithm, given in Figure 1, executes in a time bound of $O(n^2)$, where n denotes the length of the string used to represent G. As previously stated, we assume that all the FDs in G have only one attribute at the right side (using the decomposition property of FDs, it is easy to reduce any set of FDs to this form). Then we prove that the algorithm does actually produce EKNF relations. Observe first that we have the following property:

LEMMA 6. *Every FD in the minimal cover H, produced by Step 1 of Bernstein's algorithm, is elementary with respect to G^+.*

PROOF. Easy.

We can now prove the following:

THEOREM 1. *Every key synthesized by Bernstein's algorithm is elementary.*

PROOF. If X is a synthesized key for a relation R, then H must contain some elementary FD with left side X, say, $f\colon X \to A$. There are two possibilities:

1. $f \in (H' + J)^+$. Then A is in R and, by definition, X is an elementary key.
2. $f \notin (H' + J)^+$. In this case, f must have been deleted from H at Step 4 since

$$f \in ((H' + J) - \{f\})^+. \tag{5.1}$$

The cover H is a minimal. Thus, every nonredundant derivation of $f\colon X \to A$ from $(H' + J) - \{f\}$ uses some $g\colon Y \to B$, where $g \in J$, and g is not derivable from $H - \{f\}$. But if g is not derivable from $H - \{f\}$, then each derivation of g from H must use f. Thus, Lemma 1 yields $X \leftrightarrow Y$. Therefore, X and Y are both synthesized keys for R which then contains attribute B. We now complete the proof by showing that $h\colon X \to B$ is elementary. For this purpose, observe that, since g is not implied by $H - \{f\}$, neither is $h\colon X \to B$ (otherwise, one could use it in place of $Y \to B$ to derive f from $H - \{f\}$.) Thus,

$$h \notin (H - \{f\})^+. \tag{5.2}$$

We prove that h is elementary by contradiction: say that $h' \in H^+$, where $h'\colon X' \to B$ with X' properly contained in X. Now, since $h' \in H^+$ and h' is derivable from H without the use of f (otherwise $X' \to X$ by Lemma 1, contradicting the fact that f is elementary), then h' is implied by $(H - \{f\})$. Since this contradicts (5.2), our proof is complete. Q.E.D.

Synthesizing a Relational Schema from a Set of FDs

Step 1: (Eliminate extraneous attributes and find nonredundant cover.) For each $f: X \rightarrow A \in G$ and each $B \in X$ do, if f': $(X - \{B\}) \rightarrow A$ then replace f by f' in G. Then find a nonredundant cover H for G.

Step 2: (Partition.) Partition H into groups such that all the FDs in each group have identical left sides.

Step 3: (Merge Equivalent Keys.) Set J equal to the empty set. For each pair of groups, say, $H1$ and $H2$ with left sides X and Y, respectively, merge $H1$ and $H2$ together if there is a bijection $X \leftrightarrow Y$ in H^+. For each $A \in Y$, add $f1$: $X \rightarrow A$ to J, and if $f1$ is in H delete it from H. Likewise, for each $B \in X$, add $f2$: $Y \rightarrow B$ to J, and if $f2$ is in H delete it from H.

Step 4: Find $H' \subseteq H$ such that $(H' + J)^+ = (H + J)^+$ and no proper subset of H' has this property. Add each FD of J into the corresponding group of H'.

Step 5: (Construct Relations.) For each group, construct a relation consisting of the attributes appearing in that group. Each set of attributes that appears on the left side of any FD in the group is a key of the relation (each key so constructed is called *synthesized*). The set of constructed relations constitutes a schema for the given set of FDs, G.

Fig. 1. Bernstein's algorithm.

THEOREM 2. *Every relation produced by Bernstein's algorithm is EKNF.*

PROOF. Indeed, if a relation R produced by the algorithm were to contain an elementary FD, say, $X \rightarrow A$, where X were not a key, nor A belonged to any synthesized key, then A would be transitively dependent on the synthesized keys of R. And this would contradict the main result of [6]. Q.E.D.

The principles of representation, minimal redundancy, and separation were proposed in [4] as a general basis for schema design and as a yardstick for evaluating schema design algorithms. Since they satisfy properties (1) and (2), the schemata produced by Bernstein's algorithm satisfy the principles of representation and minimal redundancy. Moreover, we now find that these schemata satisfy the principle of separation to an extent which was not previously known. Indeed, they satisfy the EKNF definition, which is significantly stricter than 3NF. This indicates that the algorithm supplies a sound basis for schema design. This conclusion is corroborated by recent results which extend and improve the original algorithm. For instance, it has been shown in [18] that the algorithm can often be used, following a decomposition driven by both functional and multivalued dependencies, to ensure a better modeling of nonfunctional relationships. Moreover, with the addition presented in [8], the algorithm produces schemata that, along with the properties previously discussed (including the EKNF property), also have the lossless join property [1]. Moreover, the algorithms proposed in [14] can be used to improve these schemata further, by removing from each relation every attribute that can be eliminated without compromising representation, thus enforcing a stronger notion of minimal redundancy. These results seem to confirm the basic soundness and robustness of Bernstein's algorithm and stress the importance of EKNF which, more accurately, identifies the class of schemata produced by this algorithm.

6. CONCLUSION

In this paper we have improved the current understanding of the properties of normal forms and their design algorithms. We have also proposed a new normal form which, according to the principles of representation and separation, supplies a better basis for schema design than do 3NF or BCNF.

We began by deriving a (surprisingly) simpler definition of 3NF. Then we noted the analogy between this new definition and the definition of BCNF. A new normal form was then proposed which combines the salient qualities of both BCNF and 3NF. Finally, we proved that the algorithm proposed in [6] for the design of 3NF schemata does in fact produce EKNF schemata.

ACKNOWLEDGMENTS

I am grateful to Phil Bernstein and Stott Parker for their valuable suggestions for improvements. I would also like to thank Dan Connelly, Murray Edelberg, Ken Lin, and the referees for their comments.

REFERENCES

1. AHO, A.V., BEERI, C., AND ULLMAN, J.D. The theory of joins in relational databases. *ACM Trans. Database Syst.* 4, 3 (Sept. 1979), 297–314.
2. ARMSTRONG, W.W. Dependency structures of database relationships. In *Proc. IFIP 74*, North-Holland, Amsterdam, 1974, pp. 580–583.
3. BEERI, C., AND BERNSTEIN, P.A. Computational problems related to the design of normal form relational schemata. *ACM Trans. Database Syst. 4*, 1 (March 1979), 30–59.
4. BEERI, C., BERNSTEIN, P.A., AND GOODMAN, N. A sophisticate's introduction to data base normalization theory. In *Proc. 4th Conf. Very Large Data Bases* (West Berlin, Germany, Sept. 1978), 113–124.
5. BEERI, C., FAGIN, R., AND HOWARD, J.H. A complete axiomatization for functional and multivalued dependencies. In *Proc. ACM SIGMOD Conf.* (Toronto, Canada, Aug. 3-5), ACM, New York, 1977, pp. 47–61.
6. BERNSTEIN, P.A. Synthesizing Third Normal Form relations from functional dependencies. *ACM Trans. Database Syst. 1*, 4 (Dec. 1976), 277–298.
7. BERNSTEIN, P.A., AND GOODMAN, N. What does Boyce-Codd Normal Form do? In *Proc. 6th Conf. Very Large Data Bases* (Montreal, Canada, Oct. 1–3 1980).
8. BISKUP, J., DAYAL, U., AND BERNSTEIN, P.A. Synthesizing independent database schemata. In *Proc. ACM SIGMOD Conf.* (Boston, Mass., May 30–June 1), ACM, New York, 1979, pp. 143–151.
9. CODD, E.F. Further normalization of the data base relational model. In *Data Base Systems*, Courant Institute Computer Science Symposia Series, vol. 6, R. Rustin, Ed., Prentice-Hall, Englewood Cliffs, N.J., 1972.
10. CODD, E.F. Recent investigations in relational database systems. In *Proc. IFIP 74*, North-Holland, Amsterdam, 1974, pp. 33–36.
11. DATE, C.J. *An Introduction to Database Systems*, 2nd ed. Addison-Wesley, Reading, Mass., 1977.
12. FAGIN, R. Multivalued dependencies and a new formal form for relational databases. *ACM Trans. Database Syst. 2*, 3 (Sept. 1977), 262–278.
13. FAGIN, R. Normal forms and relational database operators. In *Proc. 1979 ACM SIGMOD Conf.* (Boston, Mass., May 30–June 1), ACM, New York, 1979, pp. 153–160.
14. LING, T.W., TOMPA, F.W., AND KAMEDA, T. An improved Third Normal Form for relational databases. *ACM Trans. Database Syst. 6*, 2 (June 1981), 329–346.
15. MAIER, D., MENDELZON, A.O. AND SAGIV, Y. Testing implications of data dependencies. *ACM Trans. Database Syst. 4*, 4 (Dec. 1979), 455–469.

16. RISSANEN, J. Theory of relational databases—A tutorial survey. In *Proc. 7th Symp. Math.*, Found. Comp. Sci. Lecture notes in Computer Science 64, Springer-Verlag, pp. 537–551.
17. ZANIOLO, C. Analysis and design of relational schemata for database systems. Ph.D. dissertation, Tech. Rep. UCLA-Eng-7769, Dep. Computer Science, Univ. California at Los Angeles, July 1976.
18. ZANIOLO, C., AND MELKANOFF, M.A. On the design of relational database schemata. *ACM Trans. Database Syst. 6*, 1 (March 1981), 1–47.

Received November 1979; revised January 1981; accepted February 1981