ARTICLE

A normal form for relational databases that is based on domains and keys

RONALD FAGIN, IBM Research - Almaden, San Jose, CA, United States

ACM Transactions on Database Systems (TODS), Volume 6, Issue 3 (September 1981)
https://doi.org/10.1145/319587.319592
EISSN: 1557-4644

A Normal Form for Relational Databases That Is Based on Domains and Keys

RONALD FAGIN
IBM Research Laboratory

A new normal form for relational databases, called domain-key normal form (DK/NF), is defined. Also, formal definitions of insertion anomaly and deletion anomaly are presented. It is shown that a schema is in DK/NF if and only if it has no insertion or deletion anomalies. Unlike previously defined normal forms, DK/NF is not defined in terms of traditional dependencies (functional, multivalued, or join). Instead, it is defined in terms of the more primitive concepts of domain and key, along with the general concept of a "constraint." We also consider how the definitions of traditional normal forms might be modified by taking into consideration, for the first time, the combinatorial consequences of bounded domain sizes. It is shown that after this modification, these traditional normal forms are all implied by DK/NF. In particular, if all domains are infinite, then these traditional normal forms are all implied by DK/NF.

Key Words and Phrases: normalization, database design, functional dependency, multivalued dependency, join dependency, domain-key normal form, DK/NF, relational database, anomaly, complexity
CR Categories: 3.73, 4.33, 4.34, 5.21

1. INTRODUCTION

Historically, the study of normal forms in a relational database has dealt with two separate issues. The first issue concerns the definition of the normal form in question, call it Q normal form (where Q can be "third" [10], "Boyce-Codd" [11], "fourth" [16], "projection-join" [17], etc.). A relation schema (which, as we shall see, can be thought of as a set of constraints that the instances must obey) is considered to be "good" in some sense if it is in Q normal form. The second issue concerns the normalization process, that is, how one might, if possible, convert a relation schema that is not in Q normal form into a collection of relation schemata, each of which is in Q normal form, and where the new set of schemata is somehow equivalent to the original schema. In this paper, our primary focus is the first issue. We show that the second issue leads to a host of research questions.

We define a new normal form for relational databases that is based only on the primitive concepts of domain and key, and we call it *domain-key normal form*, or *DK/NF*. Intuitively, a relation schema is in DK/NF if every constraint can be inferred by simply knowing the set of attribute names and their underlying domains, along with the set of keys. We also define insertion and deletion

Author's address: IBM Research Laboratory, San Jose, CA 95193.

anomalies. (Our definition of "anomaly" is different from that of Codd [10]. Codd's anomalies have been adequately resolved, that is, prevented, by Boyce-Codd normal form [11].) Intuitively, an insertion anomaly (in our sense) occurs when a seemingly legal insertion of a single tuple into a valid instance of the schema causes the resulting relation to violate one of the constraints of the schema. Here, "seemingly legal" means that every entry of the new tuple is in the appropriate domain, and that the new tuple differs from all previous tuples on every key. Similarly, a deletion anomaly occurs when the deletion of a single tuple from a valid instance of the schema causes a constraint to be violated. We show that a satisfiable relation schema (a schema that has at least one valid instance) is in DK/NF if and only if it has no insertion or deletion anomalies.

Unlike previous normal form definitions, our definition of DK/NF is *operator free*. Our viewpoint is that in the best of all possible worlds, every relation schema in a database would be in DK/NF. (Ideally, there would be no interrelational constraints either. We discuss the issue of interrelational constraints more in Section 4.) The very important practical question of exactly when DK/NF can be obtained (and how to obtain it) is open. It certainly depends on what the allowed transformations are. We discuss this issue more in Section 4. In this paper, we deal mainly with the issue of what DK/NF is, that is, with what it means for a relation schema to be "good." We consider it to be an important research problem to deal with the other question, of how and when DK/NF is obtainable under various transformations.

In Section 2, we present a number of definitions. In Section 3, we show that a satisfiable relation schema is in DK/NF if and only if it has no insertion or deletion anomalies. In Section 4, we present some schemata as examples, and discuss the issue of converting them into DK/NF. We also discuss the general problem of transformation into DK/NF. In Section 5, we summarize some results from [17] about Boyce-Codd normal form, fourth normal form, and projection-join normal form. These results expose a strong analogy that holds between them. In Section 6, we consider the effect of modifying these previously defined normal forms to take account of the combinatorial effects of bounded domain sizes. This effect has been largely ignored in the past. Previous research proceeded under the assumption that domain sizes were infinite (or, perhaps, so large that no unexpected combinatorial effects would appear). We show that after this modification, these previously defined normal forms are all implied by DK/NF. It follows, in particular, that if all domains are infinite, then DK/NF implies all of these normal forms. We also consider a formalization of Smith's "(3,3) normal form" [28] and show that it, too, is implied by DK/NF. In Section 7, we discuss approaches toward weakening DK/NF by allowing other simple constraints besides domain and key constraints. In Section 8, we present our conclusions.

2. DEFINITIONS

Let X be a finite set of distinct symbols called *attributes* (or *column names*). In the spirit of Armstrong [3] and of Aho, Beeri, and Ullman [1] we define an *X-tuple* (or simply a *tuple* if X is understood) to be a function with domain X. Thus a tuple is a mapping that associates a value with each attribute in X. We call the value associated with the attribute A the *A entry* of the tuple. Note that

under this definition, the often-made statement that "the order of the columns does not matter" is true, since "columns" correspond to attributes, and we have imposed no ordering on the attributes. If Y is a subset of X, and if t is an X-tuple, then $t[Y]$ denotes the Y-tuple obtained by restricting the mapping to Y. An *X-relation* (or simply a *relation* if X is understood) is a finite set of X-tuples. Note that we are explicitly assuming the finiteness of relations (that is, relations are assumed to be *finite* sets of tuples). This corresponds to custom, and, of course, to practice. We must make this assumption explicit since the concept of logical consequence (defined later) would otherwise be subtly different. The proof of Theorem 3.13, for example, requires the finiteness assumption. If R is an X-relation, and if Y is a subset of X, then by $R[Y]$, the *projection* of R onto Y, we mean the set of all tuples $t[Y]$, where t is in R.

An *X-constraint* (or simply a *constraint* if X is understood) is a mapping from the collection of all X-relations into {TRUE, FALSE}. Constraints are often restricted to those that can be specified by sentences written in a given language, such as first-order predicate calculus. (We make no such restriction.) We say that relation R *obeys* constant σ, or that σ *holds* in R, if under this mapping, relation R takes on the value TRUE. Otherwise, we say that σ *fails* in R. Note that for us, constraints are static, not dynamic, in that they deal with relations, but not with transitions between relations. That is, we are not considering "dynamic constraints," such as "an employee's salary cannot decrease." Later in this section we discuss certain constraints of special interest to us. These include (1) traditional dependencies (functional, multivalued, and join dependencies), (2) domain dependencies (or DDs), and (3) key dependencies (or KDs).

Let X be a set of attributes, let σ be a constraint, and let Σ be either a constraint or a set of constraints. When we say that Σ *logically implies* σ (*in the context of* X), or that σ *is a logical consequence of* Σ (*in the context of* X), we mean that whenever Σ holds for a relation with attributes X, then so does σ. That is, there is no "counterexample relation" R with X as its set of attributes such that every constraint in Σ holds in R, but such that σ fails in R. We write $\Sigma \models_X \sigma$, or, if the context X is understood, simply $\Sigma \models \sigma$. As a simple example,

$$\{A \rightarrow B,\ B \rightarrow C\} \models A \rightarrow C.$$

Following Cadiou [7], we define a *relation schema* to be a set of attributes, along with a set of constraints. So that we can always speak of constraints *in* the schema, rather than constraints that are *logical consequences* of those in the schema, it is convenient to assume that the set of constraints of the schema is closed under logical consequence. That is, if Σ is the set of constraints of the schema, and if σ is a constraint such that $\Sigma \models \sigma$, then σ is also a constraint of the schema. A relation is a *valid instance* (or simply an *instance*) of the schema if it has the same attributes as the schema, and if it obeys every constraint of the schema.

We now define certain dependencies of special interest.

(1) *Functional, multivalued, and join dependencies.* Let U and V be sets of attributes. The *functional dependency* [9] $U \rightarrow V$ is a constraint that says that every pair of tuples that agree in the U entries must also agree in the V entries.

If S is a UV-relation and T is a UW-relation (where the attributes in common are the attributes in U), then the join $S * T$ of S and T is the relation consisting of the tuples (u, v, w), where (u, v) is in S and (u, w) is in T. (*Note*: When we write UV, where U and V are sets of attributes, we mean the set $U \cup V$. When we write AB, where A and B are attributes, we mean the set $\{A, B\}$; similarly for $ABCD$ and so on.) The concept of multivalued dependency (see Fagin [16]) is intimately related to that of join. Specifically, if U and V are subsets of attributes of a relation R, and if W is the set of attributes of R not in U or V, then the *multivalued dependency* $U \rightarrow\!\!\!\!\rightarrow V$ holds in R if and only if R is the join of its projections $R[UV]$ and $R[UW]$. (This characterization of multivalued dependencies appears in [16] as Theorem 1.)

It is convenient for us to use a generalized definition of join, due to Aho, Beeri, and Ullman [1], that gives the join of an arbitrary collection of relations. If $R_1, \ldots, R_r$ are relations, then the *join* of the set $\{R_1, \ldots, R_r\}$ is the set of all tuples $(a_1, \ldots, a_n)$, where $A_1, \ldots, A_n$ are the attributes appearing in at least one of $R_1, \ldots, R_r$, and where for each i, the projection of the tuple $(a_1, \ldots, a_n)$ onto the attributes of R_i is a tuple in the relation R_i. It is straightforward to verify that the old and new definitions of join agree when there are only two relations to be joined. Furthermore, this generalized join can be "built up" out of binary joins. For example, the join $*\{R_1, R_2, R_3, R_4\}$ is equal to $(((R_1 * R_2) * R_3) * R_4)$, that is, the result of joining R_1 and R_2, joining the result with R_3, and then joining this result with R_4. By the associativity and commutativity of the binary join, we can take the binary joins in any order we wish, and parenthesize however we wish, and still get the same answer. For example, $*\{R_1, R_2, R_3, R_4\}$ also equals $(R_2 * R_3) * (R_4 * R_1)$.

Let $X_1, \ldots, X_r$ each be subsets (not necessarily disjoint) of the attributes of relation R, where each attribute of R is contained in at least one of $X_1, \ldots, X_r$. Following Rissanen [26], we say that R obeys the *join dependency* $*\{X_1, \ldots, X_r\}$ if R is the join of its projections $R[X_1], \ldots, R[X_r]$.

As noted in [1], the join of the projections $R[X_1], \ldots, R[X_r]$ of R equals

$$\{t: \text{ there are tuples } w_1, \ldots, w_r \text{ of } R \text{ such that } w_i[X_i] = t[X_i] \text{ for each } i\,(1 \leq i \leq r)\}.$$

It follows that the join dependency $*\{X_1, \ldots, X_r\}$ holds for the relation R if and only if R contains each tuple t for which there are tuples $w_1, \ldots, w_r$ of R (not necessarily distinct) such that $w_i[X_i] = t[X_i]$ for each i $(1 \leq i \leq r)$.

If R and U, V, W are as in the definition above of multivalued dependency, then the multivalued dependency $U \rightarrow\!\!\!\!\rightarrow V$ holds in R if and only if the join dependency $*\{UV, UW\}$ holds in R. Thus it is possible to represent each multivalued dependency by a join dependency. We note that Nicolas [24] defined the *mutual dependency*, which is a special case of the join dependency where there are exactly three relations to be joined. Delobel [15] and Dayal and Bernstein [14] also explore properties of join dependencies.

From this point on, we abbreviate functional dependency, multivalued dependency, and join dependency by FD, MVD, and JD, respectively.

(2) *Domain dependencies* (*DDs*). The domain dependency IN(A,S), where A is an attribute and S is a set, means that the A entry in each tuple must be a member of the set S. For example, let A be the attribute SALARY, and let S be the set of all integers between 10,000 and 100,000. If A is one of the attributes of relation R, then R obeys IN(A,S) if and only if the SALARY entry of every tuple of R is an integer between 10,000 and 100,000.

(3) *Key dependencies* (*KDs*). The key dependency KEY(K), where K is a set of attributes, says that K is a *key*, that is, that no two tuples have the same K entries. Thus if R is an X-relation, that is, if X is the set of attributes of relation R, then KEY(K) holds for R if and only if the FD $K \rightarrow X$ does. For notational convenience, we may write KEY($A_1, \ldots, A_m$) for KEY($\{A_1, \ldots, A_m\}$). We remark that keys are usually defined at the schema level, not at the instance level. At the schema level, for K to be a key, it is usually assumed that K is minimal, that is, that no proper subset of K is a key. However, this minimality assumption is not useful at the instance level. Thus we wish a relation in which a proper subset of K happens to uniquely identify tuples to be a possible instance of a schema in which K is a key. So our definition, in which minimality is not assumed, is more convenient for our purposes.

3. DEFINITION OF INSERTION ANOMALY, DELETION ANOMALY, AND DK/NF

In this section, we present a definition of an insertion anomaly and of a deletion anomaly. We define DK/NF, and we show that a satisfiable relation schema is in DK/NF if and only if it has no insertion or deletion anomalies. We present some schemata as examples. We assume throughout this paper that every relation schema we consider is already in Codd's first normal form (1NF), that is, each entry of each instance is atomic.

Definition 3.1. Let R^* be a relation schema, and let R be a valid instance of R^*. Let t be an arbitrary tuple not in R. We say that tuple t is *compatible* with R (in the context of R^*) if

(1) t has the same attributes as R;
(2) the A entry of t is in set S for each attribute A and each DD IN(A,S) of R^*; and
(3) $t[K] \neq s[K]$ for each KD KEY(K) of R^* and for each tuple s of R.

Thus a compatible tuple is "insertible" into R without violating any DDs or KDs.

Definition 3.2. Relation schema R^* has an *insertion anomaly* if there is a valid instance R of R^* and there is a tuple t compatible with R such that $R \cup \{t\}$, the relation obtained by inserting t into R, is not a valid instance of R^* (i.e., violates a constraint of R^*).

Example 3.3. Let R^* be a relation schema with attributes EMP, DEPT, and MGR, and with the following constraints (where $\mathscr{CHAR}20$ is the set of all

EMP	DEPT	MGR
Hilbert	Math	Gauss
Pythagoras	Math	Gauss
Turing	Computer Science	von Neumann

Figure 1

EMP	DEPT	MGR
Hilbert	Math	Gauss
Pythagoras	Math	Gauss
Turing	Computer Science	von Neumann
Cauchy	Math	Euler

Figure 2

character strings of at most 20 characters):

$$\text{IN}(\text{EMP}, \mathscr{CHAR}\,20)$$

$$\text{IN}(\text{DEPT}, \mathscr{CHAR}\,20)$$

$$\text{IN}(\text{MGR}, \mathscr{CHAR}\,20)$$

$$\text{KEY}(\text{EMP})$$

$$\text{DEPT} \rightarrow \text{MGR}$$

As we noted earlier, we tacitly close the set of constraints under logical consequence, that is, under $\models$. The schema is not in Boyce-Codd normal form, or even in third normal form [10], because of the "transitive dependence" of MGR on EMP via EMP $\rightarrow$ DEPT and DEPT $\rightarrow$ MGR. We now show that R^* has an insertion anomaly. Let R be the relation in Figure 1. This is a valid instance of the schema R^*. Let t be the tuple (Cauchy, Math, Euler). Tuple t is compatible with relation R. However, if we were to insert tuple t, then we would obtain the relation in Figure 2, which is *not* a valid instance of the schema, since it violates the FD DEPT $\rightarrow$ MGR. Thus there is an insertion anomaly. One intuitive viewpoint on what is happening here is that by the insertion of the tuple (Cauchy, Math, Euler), it is not clear whether the intent is simply to insert information about a new employee (Cauchy), or whether the intent is also to update the name of the chairman of the math department.

Example 3.4. Let R^* be a relation schema with attributes ABC, and with the following constraints (where $\mathscr{NAT}$ is the set $\{0, 1, 2, \ldots,\}$ of natural numbers):

$$\text{IN}(A, \mathscr{NAT})$$

$$\text{IN}(B, \mathscr{NAT})$$

$$\text{IN}(C, \mathscr{NAT})$$

$$\text{KEY}(ABC)$$

$$A \twoheadrightarrow B$$

A	B	C
0	0	0
1	0	0

Figure 3

A	B	C
0	0	0
1	0	0
0	1	1

Figure 4

The schema R^* is not in fourth normal form (although it is in Boyce-Codd normal form), since the MVD $A \rightarrow\rightarrow B$ holds, and A is not a key. We now show that R^* has an insertion anomaly. Let R be the relation in Figure 3. This is a valid instance of the schema R^*. Let t be the tuple $(0, 1, 1)$. Tuple t is compatible with relation R. However, if we were to insert tuple t, then we would obtain the relation in Figure 4, which is *not* a valid instance of the schema, since it violates the MVD $A \rightarrow\rightarrow B$.

Example 3.5. This example, which is discussed in [17], is a slight modification of an example due to Nicolas [24]. Assume that there are only three attributes A, P, C, where the tuple (a, p, c) means, intuitively, that agent a represents product p produced by company c. Let R^* be a relation schema with these attributes, and with the following constraints:

$$\text{IN}(A, \mathscr{CHAR}20)$$

$$\text{IN}(P, \mathscr{CHAR}20)$$

$$\text{IN}(C, \mathscr{CHAR}20)$$

$$*\{AP, AC, PC\}$$

Let us denote the JD $*\{AP, AC, PC\}$ above by σ. In English, the JD σ says precisely the following:

> Assume that agent a represents product p produced by company c', that agent a represents product p' produced by company c, and that agent a' represents product p produced by company c. Then agent a represents product p produced by company c.

It is shown in [17] that the schema R^* is in fourth normal form but not in projection-join normal form. We now show that R^* has an insertion anomaly. Let R be the relation in Figure 5. It is a valid instance of the schema. Let t be the tuple (Smith, bolt, Acme). Tuple t is compatible with relation R. However, if we were to insert tuple t, then we would obtain the relation in Figure 6, which is *not* a valid instance of the schema, since it violates the JD σ. (If we were to add the tuple (Jones, bolt, Acme) to the relation in Figure 6, then it would obey σ.)

A	P	C
Jones	bolt	Ace
Jones	screw	Acme

Figure 5

A	P	C
Jones	bolt	Ace
Jones	screw	Acme
Smith	bolt	Acme

Figure 6

EMP#	STATUS	SALARY
329461	1	73000
141592	0	37000
271828	1	46000

Figure 7

EMP#	STATUS	SALARY
329461	1	73000
141592	0	37000
271828	1	46000
141421	0	57000

Figure 8

Example 3.6. Let R^* be a relation schema with attributes EMP#, STATUS, and SALARY, and with the following constraints:

$$\text{IN}(\text{EMP\#}, \{n\colon\ n \text{ is a six-digit integer}\})$$

$$\text{IN}(\text{STATUS}, \{0, 1\})$$

$$\text{IN}(\text{SALARY}, \{n\colon\ 10{,}000 \leq n \leq 100{,}000\})$$

$$\text{KEY}(\text{EMP\#})$$

$$\forall t((t[\text{STATUS}] = 0) \Rightarrow (t[\text{SALARY}] \leq 50{,}000))$$

Intuitively, our constraints say that EMP# is a six-digit integer, and is the key; there are two statuses, represented by 0 and 1; and salaries are between \$10,000 and \$100,000, except that employees with status 0 have a salary of at most \$50,000. This schema is in fourth normal form, and even in projection–join normal form. However, it has the following insertion anomaly. Let R be the relation in Figure 7. It is a valid instance of the schema. Let t be the tuple (141421, 0, 57000). Tuple t is compatible with relation R. However, if we were to insert tuple t, then we would obtain the relation in Figure 8, which is *not* a valid instance of the

A	B	C
1	0	0
1	0	1
1	1	0
1	1	1

Figure 9

A	B	C
1	0	0
1	0	1
1	1	0

Figure 10

schema, since it violates the constraint that no employee with status 0 can have a salary greater than $50,000.

Example 3.7. This example is due to Smith [28]. Let P__ROLE be a relation schema with attributes PERSON and ROLE. A PERSON is assumed to have two types of ROLEs: sex role (male or female) and professional role. It is assumed that a person can have several professions, but only one sex. This schema is in projection-join normal form, but not in Smith's (3,3) normal form [28] (which we discuss briefly at the end of Section 6). The schema has an insertion anomaly since it is possible to insert a tuple that gives the same person two different sex roles, without violating any DDs or KDs. Smith notes that the schema can be *split* [17] into two schemata, SEX__ROLE and PROF__ROLE. A tuple about a person's sex role goes into the first relation, and tuples about his professional roles go into the second. The schema SEX__ROLE obeys the FD PERSON → ROLE, while the schema PROF__ROLE does not. As Smith notes, an advantage of splitting is that the system's key maintenance mechanism can then prevent the insertion of two sex roles for one person. In our terminology, a schema that is not in DK/NF has been split into two schemata, each of which is in DK/NF. Our intent is *not* that, at time of insertion, the database management system check the tuple and decide which of the two relations it should go into. Instead, our intent is that the *user view* of the database be two relations, one involving sex roles and one involving professional roles, and that the *user* should decide which relation to insert a tuple into.

We now define a deletion anomaly.

Definition 3.8. Relation schema R^* has a *deletion anomaly* if there is a valid instance R of R^* and a tuple t in R such that the relation obtained by removing t from R is not a valid instance of R^* (i.e., violates a constraint of R^*).

Example 3.9. Let R^* be the same relation schema as in Example 3.4. We now show that R^* has a deletion anomaly. Let R be the relation in Figure 9. This is a valid instance of the schema R^*. Let t be the tuple $(1, 1, 1)$. If we delete tuple t from relation R, then we obtain the relation in Figure 10, which is *not* a valid instance of the schema, since it violates the MVD $A \rightarrow\!\!\!\rightarrow B$.

EMP#	MGR#
414213	732050
236067	732050
732050	449489
449489	—

Figure 11

EMP#	MGR#
414213	732050
236067	732050
449489	—

Figure 12

Example 3.10. Let R^* be a relation schema with attributes EMP# and MGR#, and with the following constraints:

$$\text{IN}(\text{EMP\#}, \{n\colon n \text{ is a six-digit integer}\})$$

$$\text{IN}(\text{MGR\#}, \{n\colon n \text{ is a six-digit integer}\})$$

$$\text{KEY}(\text{EMP\#})$$

$$\text{MGR\#} \subseteq \text{EMP\#}$$

By MGR# $\subseteq$ EMP#, we mean the constraint that says that every MGR# entry must also be an EMP# entry. We call such constraints *inclusion dependencies*. They have also been used by Smith and Smith [29], Zaniolo [34], and Codd [12]. Probably their greatest importance is as an *interrelational* constraint, in which we might say that each A entry of relation R appears as a B entry of relation S. More generally, an inclusion dependency can say that the projection onto a given m columns of relation R are a subset of the projection onto a given m columns of relation S. Inclusion dependencies are a special case of extended embedded implicational dependencies (see Fagin [18]). We discuss them more in Section 4.

This schema is in projection-join normal form. However, we now show that it has a deletion anomaly. Let R be the relation in Figure 11. This is a valid instance of the schema (we have a null entry corresponding to the manager of employee number 449489 since this employee is the head of the organization and has no manager). Let t be the tuple (732050, 449489). If we delete tuple t from relation R, then we obtain the relation in Figure 12, which is *not* a valid instance of the schema since it violates the inclusion dependency MGR# $\subseteq$ EMP#.

Example 3.11. Let R^* be a relation schema with only one attribute A, and with the following constraints:

$$\text{IN}(A, \mathcal{NAT})$$

$$\exists t_1 \exists t_2 \exists t_3((t_1 \neq t_2) \wedge (t_1 \neq t_3) \wedge (t_2 \neq t_3))$$

The predicate calculus expression simply says that there are at least three distinct tuples. It is easy to see that this schema has a deletion anomaly.

We now define DK/NF.

Definition 3.12. Let R^* be a 1NF relation schema, and let Γ be the set of DDs and KDs of the schema. R^* is in *domain-key normal form* (*DK/NF*) if $\Gamma \models \sigma$ for every constraint σ of R^*.

That is, a 1NF relation schema is in DK/NF if every constraint can be inferred by simply knowing the DDs (domain dependencies) and the KDs (key dependencies). Putting it another way: a 1NF relation schema is in DK/NF if, by enforcing the DDs and KDs, every constraint of the schema is automatically enforced. This is good, since DDs ("this entry must lie in this set") and KDs ("this entry is a unique identifier") are quite fundamental and easily understood. By constrast, each of Boyce–Codd normal form, fourth normal form, and projection-join normal form (all discussed in the next section) depend heavily on the concept of FD; the latter two normal forms also depend on the concepts of MVD and JD. Although these three traditional dependencies are quite useful from a practical point of view, it is difficult to justify that they are inherently fundamental (from a mathematical point of view) to the relational model. Our definitions of DK/NF and of anomaly, however, depend only on the primitive concepts of domains (via DDs) and keys (via KDs), along with the general concept of a "constraint."

Bernstein [6, p. 280] also discusses the basic nature of the key concept. He notes that data definition languages tend to allow the definition of keys, but not of FDs. He defines an FD to be *embodied* in a schema if its left-hand side is a key of the schema (and if its right-hand side is an attribute of the schema). He thereby allows one to replace the specification of FDs by the more primitive notion of specification of keys. Another point about the basic nature of the key concept: relations, being sets (of tuples), are unordered. Hence it is desirable to refer to a tuple in a content-addressable manner; keys provide a uniform mechanism for doing so.

If all relation schemata that correspond to stored relations in a database are in DK/NF (and if there are no interrelatonal dependencies), then the database management system need only have a mechanism for supporting DDs and KDs, rather than a mechanism for supporting more general constraints. We remark that while System R [5] does not even support general FDs, it does support keys (via "unique indices").

We now need to define the concept of satisfiability. A relation schema is *satisfiable* if it has at least one valid instance. Thus a relation schema is satisfiable if its set of constraints is noncontradictory. An example of a nonsatisfiable schema would be one in which one of the constraints is $\exists t(t \neq t)$. Of course, the only schemata of any interest are satisfiable. We remark that if the DD $\mathrm{IN}(A,\varnothing)$, where $\varnothing$ is the empty set, holds for a relation schema, then this does not necessarily mean that the schema is unsatisfiable. It simply means that the only possible valid instance for the schema is the empty relation (with no tuples). In particular, every set of KDs and DDs alone (or, more exactly, the schema determined by a set of attributes along with this set of constraints) is satisfiable,

since the empty relation (with the appropriate attributes) is always a valid instance.

The following theorem, which is not deep, shows that there are two equivalent ways of defining what it means for a satisfiable 1NF schema to be in DK/NF. The first is the definition we gave, which could be called "static." The second possible definition could be called "dynamic," since it discusses the effects of insertions and deletions (in particular, it says that there are no insertion or deletion anomalies).

THEOREM 3.13. *A satisfiable 1NF relation schema is in DK/NF if and only if it has no insertion or deletion anomalies.*

PROOF. Assume first that R^* is a satisfiable 1NF relation schema in DK/NF. We shall show that R^* has no insertion or deletion anomalies. Let R be a valid instance of R^*. We must show that R', which is obtained from R by either inserting a tuple compatible with R, or by deleting a tuple from R, is also a valid instance of R^*. Let Γ be the set of DDs and KDs of R^*. It is easy to verify that since R obeys Γ, so does R'. To show that R' is a valid instance of R^*, we must show that R' obeys every constraint σ of R^*. But if σ is an arbitrary constraint of R^*, then $\Gamma \models \sigma$, since R^* is in DK/NF. Hence since R' obeys Γ and since $\Gamma \models \sigma$, it follows that R' obeys σ, as desired.

Conversely, we now show that if R^* is a satisfiable 1NF relation schema that is not in DK/NF, then it has either an insertion or a deletion anomaly. Assume that R^* is a satisfiable 1NF relation schema that is not in DK/NF. Let Γ be the set of DDs and KDs of R^*. Since R^* is not in DK/NF, it has a constraint σ that is not a logical consequence of Γ. By definition of logical consequence, this means that there exists a "counterexample relation" S for which Γ holds and σ fails. In particular, S is not a valid instance of the schema R^*, since σ fails. Since R^* is satisfiable, it has a valid instance. Let us call this valid instance R. Since, by our definitions, a "relation" is a *finite* set of tuples, each of R and S contain a finite number of tuples. Assume that R contains p distinct tuples $r_1, \ldots, r_p$, and that S contains q distinct tuples $s_1, \ldots, s_q$. We now define $p+q+1$ relations $T_i (-p \leq i \leq q)$, as follows:

$$
\begin{aligned}
T_{-p} &= \{r_1, \ldots, r_p\} = R \\
T_{-p+1} &= \{r_1, \ldots, r_{p-1}\} \\
&\vdots \\
T_{-i} &= \{r_1, \ldots, r_i\} \qquad (1 \leq i \leq p) \\
&\vdots \\
T_0 &= \varnothing \\
T_1 &= \{s_1\} \\
T_2 &= \{s_1, s_2\} \\
&\vdots \\
T_i &= \{s_1, \ldots, s_i\} \qquad (1 \leq i \leq q) \\
&\vdots \\
T_q &= \{s_1, \ldots, s_q\} = S
\end{aligned}
$$

Thus the sequence of relations T_i is obtained as follows. The sequence begins with the relation R; then one tuple at a time is deleted until the empty relation is obtained; then one tuple at a time is inserted until the relation S is obtained. Now the first relation ($T_{-p} = R$) in the sequence is a valid instance of the schema, and the last relation ($T_q = S$) in the sequence is not a valid instance of the schema. Therefore, for some k ($-p \leq k \leq q-1$), we know that T_k is a valid instance of the schema and T_{k+1} is not. If $k<0$, that is, if T_{k+1} is obtained from T_k by deleting a tuple, then R^* has a deletion anomaly. If $k \geq 0$, that is, if T_{k+1} is obtained from T_k by inserting a tuple t, then R^* has an insertion anomaly, since t is compatible with T_k. (The reason that t is compatible with T_k is that t and all of the tuples of T_k are in S, and S obeys Γ.) Thus R^* has either an insertion or a deletion anomaly, which was to be shown. □

Note that the (non-DK/NF) schema in Example 3.3 has an insertion anomaly but no deletion anomaly, and the (non-DK/NF) schema in Example 3.11 has a deletion anomaly but no insertion anomaly.

Although, as we just noted, it is possible for a satisfiable schema to violate DK/NF without having an insertion anomaly, we do have the following result.

THEOREM 3.14. *Let R^* be a 1NF relation schema. Then R^* is in DK/NF if and only if* (1) *it has no insertion anomaly, and* (2) *the empty relation (with the appropriate attributes) is a valid instance of the schema.*

PROOF. Assume first that R^* is in DK/NF. As in the proof of the "only if" direction of Theorem 3.13, it follows that R^* has no insertion anomaly. So (1) holds. And, (2) holds, since the empty relation obeys every DD and KD. We have now proved the "only if" direction of the theorem. The "if" direction can be proved by a modification of the proof of the "if" direction of Theorem 3.13, where we let the relation R in the proof be the empty relation. □

Note that the empty relation (with a given set of attributes) obeys every DD, KD, FD, MVD, and JD (with the same attributes). In particular, all of our examples that are not in DK/NF (except Example 3.11) obey (2) but not (1) of Theorem 3.14. The schema in Example 3.11 is one for which the empty relation is *not* a valid instance. This schema, which is not in DK/NF, obeys (1) but not (2).

4. TRANSFORMING INTO DK/NF

As we noted in the introduction, the primary focus of this paper is on *defining* DK/NF (and on giving some of its properties), rather than on *telling how to obtain* DK/NF (and on determining when this is possible). Thus in this paper, the main issue is what it means for a relation schema to be "good," rather than on how to *make* it good. We now briefly discuss this latter issue. We begin by considering the normalization of the examples in the previous section.

The schema in Example 3.3 (with attributes {EMP, DEPT, MGR}) is not in DK/NF, but can be decomposed into two DK/NF schemata R_1^* and R_2^*, with attributes {EMP, DEPT} and {DEPT, MGR}, respectively. This is classic normalization, as introduced by Codd [10]. We note that sometimes (but not always), it might be desirable in practice to have an interrelational constraint that says

that if R_1 and R_2 are the instances at a given time of R_1^* and R_2^*, respectively, then R_1[DEPT] = R_2[DEPT], that is, that the DEPT entries must be the same. This issue is a subtle one, which we return to later in this section.

The schema in Example 3.4 is not in DK/NF. By decomposing the schema into two schemata, with attributes AB and AC, respectively, we obtain two DK/NF schemata.

The schema in Example 3.5 is not in DK/NF. However, by decomposing the schema into three schemata, with attributes AP, AC, and PC, respectively, we obtain three DK/NF schemata.

The schema in Example 3.6 is not in DK/NF, and the decomposition approach (the use of projections) is not powerful enough to obtain DK/NF. However, the split operator (as discussed in [17] and in Example 3.7) does do the job. That is, we replace the original schema R^* of Example 3.6 by two schemata R_1^* and R_2^*, where instances of schema R_1^* deal with employees of status 0, and where instances of schema R_2^* deal with employees of status 1. Thus R_1^*, which deals with status 0 employees, has attributes EMP# and SALARY, and the following constraints:

$$\text{IN(EMP\#}, \{n\text{: } n \text{ is a six-digit integer}\})$$

$$\text{IN(SALARY}, \{n\text{: } 10{,}000 \leq n \leq 50{,}000\})$$

$$\text{KEY(EMP\#)}$$

Note that a STATUS attribute is no longer necessary or desirable since all STATUS values would be identically 0. Schema R_2^*, which deals with status 1 employees, is just like schema R_1^*, except that 50,000 is replaced by 100,000. We shall return to this example later.

Similarly, as discussed in Example 3.7, the split operator converts the schema of Example 3.7, which is not in DK/NF, into two schemata, each of which is in DK/NF.

The schema of Example 3.10 is not in DK/NF, and there is no obvious, natural way to convert it into several schemata, each of which is in DK/NF. The same comment applies to the schema of Example 3.11.

In order to discuss the possibility or impossibility of converting a schema into DK/NF, we must, of course, define the class of legal transformations. We now discuss this issue.

Define a *database schema* to be a collection of relation schemata (each of which is a set of attributes and a set of constraints), possibly along with interrelational constraints. A *transformation* from one database schema into another is a mapping from the valid instances of the first database schema into the valid instances of the second. (We sometimes say loosely that the transformation maps one *schema* into another.) In classic decomposition (as would take place, for example, in Example 3.3), the transformation consists of mapping a relation into a collection of certain of its projections. One natural demand on a transformation from one schema into another is that the transformation be 1–1. For, if D is a valid instance of the original schema, and if $f(D)$ is the transformed version (i.e., the corresponding instance of the second schema), then it is possible, in theory, to reconstruct D from $f(D)$, since the fact that f is 1–1 says precisely

that $f(D)$ uniquely determines D. This is what corresponds in the database literature to a "lossless" transformation, or a transformation "without loss of information" [1, 16]. Classical decomposition (using projections) has this 1-1 property, in the presence of the appropriate FDs, MVDs, or JDs. The whole point of Rissanen's paper on independent components of relations [25] is to consider the situation in which the transformation is not only 1-1, but also onto. Interestingly enough, one of Codd's original motivations for normalization is in direct contradiction to the "onto" property. Thus one of the reasons that Codd gives for normalizing the {EMP, DEPT, MGR} schema of Example 3.3 (with the FD DEPT $\rightarrow$ MGR) into an {EMP, DEPT} schema and a {DEPT, MGR} schema is to be able to maintain manager information for a department with no employees. Thus Codd wants it to be possible for the DEPT entries in the {EMP, DEPT} relation to be different from the DEPT entries in the {DEPT, MGR} relation. In this case, the transformation map is not onto, since no {EMP, DEPT, MGR} relation has as a pair of projections an {EMP, DEPT} relation and a {DEPT, MGR} relation in which the DEPT entries are different. Intuitively, when a transformation map is not onto, there are "less constraints" (i.e., "more possible database instances").

This last observation gives us a clue to a trivial 1-1 transformation from an arbitrary database schema into a database schema in which every relation schema is in DK/NF, and in which there are no interrelational constraints. We simply let the transformation f be the identity map (in which $f(D) = D$), and we let the "target" database schema (the result of the transformation) be the same as the "source" (i.e., the original) database schema, but with all constraints removed. Thus to every relation schema in the source database schema, there corresponds a relation schema in the target database schema with exactly the same attributes, but with no constraints (and, further, the target database schema has no interrelational constraints).

Even if we were to demand that the transformation map be both 1-1 and onto, there is still always a trivial transformation of every satisfiable database schema into a single DK/NF relation schema. For ease in exposition, we describe a transformation that "almost" works; its only flaw is that the transformation is not quite onto, since it misses one valid instance of the target schema (the empty instance). We remark that by a trivial finite modification of the transformation, the transformation can be made onto. There are exactly two attributes A and B in the target schema. Let S be the set of all valid database instances of the source database schema. The target schema has precisely the following constraints.

$$\text{IN}(A, \{0\})$$

$$\text{IN}(B, S)$$

$$\text{KEY}(A)$$

The transformation f maps an instance D to the one-tuple relation with 0 as the A entry and with D as the B entry. Although DK/NF has been attained, it is clear that nothing at all has been gained by this transformation. All the difficulties of maintaining the constraints of the source schema have simply been converted into difficulties in maintaining a very complex domain dependency. Furthermore,

the target schema is clearly unacceptable in every way, both as a user view and as a way for data to be actually stored.

In order to adequately deal with the theoretical issues involved in the problem of when it is desirable and how it is possible to convert a database schema into a "DK/NF database schema," a number of problems must be addressed. The first issue is: what is a "DK/NF database schema"? We have so far only defined a DK/NF *relation* schema. We certainly insist that a DK/NF database schema contain only DK/NF relation schemata. What about interrelational constraints? We might insist that a DK/NF database schema contain no interrelational constraints. A less strict demand, that is also natural, is to allow only inclusion dependencies (see Example 3.10) as interrelational constraints. Allowing inclusion dependencies as interrelational constraints is in the spirit of our definition of a DK/NF *relation* schema, in that inclusion dependencies are very important constraints that a database management system should have the capability of enforcing anyway. Note that an "equality dependency," which says, for example, that the set of A entries of relation R must equal the set of B entries of relation S, is equivalent to two inclusion dependencies ($R[A] \subseteq S[B]$ and $S[B] \subseteq R[A]$). One efficient mechanism for enforcing inclusion dependencies in a *double index*. Thus to maintain the inclusion dependency which says, for example, that the A entries of relation R must always be a subset (not necessarily proper) of the B entries of relation S, it is possible to keep an index on both $R[A]$ and $S[B]$ (possibly in the same physical index, as in ADABAS [30]), and thereby to enforce the constraint by checking the indices whenever there is an insertion, deletion, or update affecting these entries.

It is interesting to note that any relation schema in which the only constraints are FDs can always be transformed in a simple manner into a DK/NF database schema, in which the only interrelational constraints are inclusion dependencies. We first form one relation schema R^* that contains all of the attributes in the source relation schema, but with no constraints. For each FD $X \rightarrow Y$ of the source schema, we also form a new relation schema with attributes XY, and with the single constraint KEY(X). The interrelational constraints say that each relation is a projection of the instance R of R^* (this can be said with a collection of inclusion dependencies). This process can be best understood by an example.

Example 4.1. Assume that the source relation schema contains exactly four attributes $ABCD$, and that the constraints are the FDs $AB \rightarrow C$ and $C \rightarrow A$. This schema is not in DK/NF (it is not even in Boyce-Codd normal form). We transform this schema into a new database schema with three relation schemata R_1^*, R_2^*, and R_3^*. Schema R_1^* contains all four attributes $ABCD$, and no constraints. Schema R_2^* contains attributes ABC and the single constraint KEY(AB). Schema R_3^* contains attributes AC and the single constraint KEY(C). The interrelational constraints are

$$R_2 \subseteq R_1[ABC]$$

$$R_1[ABC] \subseteq R_2$$

$$R_3 \subseteq R_1[AC]$$

$$R_1[AC] \subseteq R_3$$

The first two interrelational constraints say that R_2 is the projection of R_1 onto ABC, and the second two interrelational constraints say that R_3 is the projection of R_1 onto AC. It is easy to verify that there is a 1-1 correspondence between the source and the target database instances. This technique is a special case of a more general approach by Armstrong and Delobel [4], in which they project onto saturated sets and antiroots. For details, see [4].

We shall come back to the issue of what interrelational dependencies to allow in Section 7.

An important practical issue is: how complicated a domain dependency can we expect (or demand) that the relational database management system be able to enforce? System R [5] and ADABAS [30] currently have the capability of enforcing domain constraints such as "each A entry of relation R must be exactly n bytes," for arbitrary fixed n. Query-by-example [21] and INGRES [32] also have this capability, but n must be at most 6. In IBM's GIS [19], it is possible to completely specify any finite domain (e.g., it is possible to say that every entry in column B of relation R must be a member of the finite domain {nail, bolt, screw}).

Another important practical issue that may arise relates to the complexity of the transformation f. (This would be an issue if the target schema is how the data are to be stored, and the source schema is how the data are to be viewed.) The complexity measure might involve computation time and memory requirements, as well as the number of page faults. Possibly, the time complexity of f and its inverse should be linear. Furthermore, if the database instance D' is obtained from D by a simple operation (such as an insertion or a deletion of a single tuple), then it should be easy to convert $f(D)$ into $f(D')$; here, "easy" might mean "constant time." Clearly, this is a fertile area for research.

We close this section with a warning. *In the general case*, it is not useful to think in terms of mechanical procedures for conversion to DK/NF, since we immediately run into undecidability results. For example, it is not even decidable as to whether a sentence of first-order logic is a tautology (this is Church's theorem [8]). One approach to circumventing undecidability is to restrict the class of constraints. For example, there are known decision procedures [23] for determining when traditional dependencies (FDs, MVDs, and JDs) are consequences of other traditional dependencies. We shall not pursue this matter further in this paper.

5. BCNF, 4NF, AND PJ/NF

In this section, we present definitions of some previously defined normal forms that are based on the projection and join operators: Boyce-Codd normal form [11], fourth normal form [16], and projection-join normal form [17]. We hereafter refer to these latter three normal forms as BCNF, 4NF, and PJ/NF, respectively. The definitions that we give are not the same as the original definitions, but, as shown in Fagin [17], are equivalent to the original definitions.

Definition 5.1. A 1NF relation schema R^* is in BCNF if $\Delta \models \sigma$ for each FD σ of R^*, where Δ is the set of KDs of R^*. Thus every FD is the result of keys.

Definition 5.2. A 1NF relation schema R^* is in 4NF if $\Delta \models \sigma$ for each MVD σ of R^*, where Δ is the set of KDs of R^*. Thus every MVD is the result of keys.

Definition 5.3. A 1NF relation schema R^* is in PJ/NF if $\Delta \models \sigma$ for each JD σ of R^*, where Δ is the set of KDs of R^*. Thus every JD is the result of keys.

The following theorem is proved in [17].

THEOREM 5.4. *PJ/NF* $\Rightarrow$ *4NF* $\Rightarrow$ *BCNF.*

There is a striking analogy between the definitions, as presented above, of BCNF, 4NF, and PJ/NF. In fact, this observation was one of the author's motivations in defining DK/NF. Unlike DK/NF, none of these earlier normal forms considered the role of domains. The purpose of the next section is to remedy this neglect.

6. MODIFIED VERSIONS OF BCNF, 4NF, AND PJ/NF

In this section, we define modified versions of BCNF, 4NF, and PJ/NF, which we call BCNF′, 4NF′, and PJ/NF′, respectively. The modification consists of taking into account the effect of DDs. It is proved that the original and modified versions are equivalent, provided no domain is too small. We also show that DK/NF $\Rightarrow$ PJ/NF′ $\Rightarrow$ 4NF′ $\Rightarrow$ BCNF′. One view of this section is that, for the first time, the combinatorial effects of a bounded domain size are being considered. This effect has been neglected in the past. For example, the tableaux technique of Aho, Beeri, and Ullman [1], and the constructions of Armstrong [3] implicitly assume that it is always possible to introduce an arbitrary number of new entries of new tuples. Kanellakis [22] has recently shown another effect of bounded domain sizes. Namely, he proves that the property of determining whether or not a lossless join property holds, in the presence of KDs and DDs, is NP-complete. We show that if all domains are infinite, then DK/NF implies PJ/NF (and, of course, 4NF and BCNF). We also give a formalization of Smith's (3,3) normal form, and show that it is also implied by DK/NF.

Let Ω be a set of constraints. For each attribute A, define $\mathrm{dom}_\Omega(A) = \cap \{S : \Omega \models \mathrm{IN}(A, S)\}$. Thus $\mathrm{dom}_\Omega(A)$ is the collection of all possible values that the A entry of a tuple of a relation can assume, if the relation obeys Ω. If Ω is the set of constraints of a relation schema, then $\mathrm{dom}_\Omega(A) = \cap \{S : \mathrm{IN}(A, S) \in \Theta\}$, where Θ is the set of DDs of the schema. This is because, as we stated in the introduction, the set of constraints of a schema are assumed to be closed under logical consequence. In particular, then, every DD implied by the constraints of the schema is in Θ. (Similarly, every KD implied by the constraints of the schema is in Δ, the set of KDs of the schema.) We usually write simply $\mathrm{dom}(A)$ for $\mathrm{dom}_\Omega(A)$. In what follows, we shall often be concerned with $|\mathrm{dom}(A)|$, the cardinality of $\mathrm{dom}(A)$.

The definitions of BCNF, 4NF, and PJ/NF in Section 4 ignore the effects of DDs. That is, for certain constraints σ, they deal with the issue of whether or not $\Delta \models \sigma$, rather than whether or not $\Gamma \models \sigma$ (recall that Δ is the set of KDs of a schema, while Γ is the set of KDs *and* DDs). The next two lemmas give conditions under which there is no difference.

LEMMA 6.1. *Let* Δ *be a set of KDs,* Θ *a set of DDs, and* $\Gamma = \Delta \cup \Theta$. *Assume that* $|dom(A)| \geq 2$ *for each attribute A. Let* σ *be an FD or MVD. Then* $\Delta \models \sigma$ *if and only if* $\Gamma \models \sigma$.

Before we prove Lemma 6.1, it is helpful to consider two examples.

Example 6.2. Let R^* be a relation schema with attributes ABC, and with the following constraints:

$$\text{IN}(A, \{1\})$$
$$\text{IN}(B, \mathcal{NAT})$$
$$\text{IN}(C, \mathcal{NAT})$$
$$\text{KEY}(BC)$$

Let σ be the FD $\emptyset \rightarrow A$, where $\emptyset$ is the empty set. The reader can easily verify from the formal definition of functional dependency that the special functional dependency $\emptyset \rightarrow A$, where the left-hand side is the empty set, says precisely that the A entry of every tuple is the same. Then σ is a constraint of the schema since σ is a logical consequence of the first DD above. (If the reader is not comfortable with FDs in which the left-hand side is the empty set, then he can let σ be the FD $B \rightarrow A$, and our example will work just as well.) Let Δ be the singleton set $\{\text{KEY}(BC)\}$, let Θ be the set consisting of the three DDs above, and let $\Gamma = \Delta \cup \Theta$. Then $\Gamma \models \sigma$, while it is false that $\Delta \models \sigma$. Note that $|\text{dom}(A)| = 1$. Thus Lemma 6.1 is false if we drop the assumption that $|\text{dom}(A)| \geq 2$. We make use of this example again later.

Example 6.3. Let R^* be a relation schema with attributes $ABCDE$, and with the following constraints:

$$\text{IN}(A, \{0, 1\})$$
$$\text{IN}(B, \mathcal{NAT})$$
$$\text{IN}(C, \mathcal{NAT})$$
$$\text{IN}(D, \mathcal{NAT})$$
$$\text{IN}(E, \mathcal{NAT})$$
$$\text{KEY}(AB)$$
$$\text{KEY}(CDE)$$

Let σ be the JD $*\{BCD, BCE, BDE, ACDE\}$. Let Γ be the set of all constraints listed above, and let Δ be the set containing only the two KDs above. We shall show that $\Gamma \models \sigma$. However, it follows from the membership algorithm in [17] (or from the "chase" algorithm in [23]) that it is *false* that $\Delta \models \sigma$. Thus Lemma 6.1 would be false if we were to allow σ to be a JD. We make use of this example again later.

We now show, as promised, that $\Gamma \models \sigma$. Recall that σ is $*\{BCD, BCE, BDE, ACDE\}$. Let R be a relation in which Γ holds. We must show that σ necessarily holds in R. To show that σ holds in R, we must show that R contains each tuple t for which there are tuples w_1, w_2, w_3, w_4 of R (not necessarily distinct) such that

$$w_1[BCD] = t[BCD] \tag{6.1}$$
$$w_2[BCE] = t[BCE] \tag{6.2}$$

$$w_3[BDE] = t[BDE] \tag{6.3}$$

$$w_4[ACDE] = t[ACDE]. \tag{6.4}$$

Thus assume that t, w_1, w_2, w_3, w_4 are as in (6.1), (6.2), (6.3), and (6.4), with w_1, w_2, w_3, w_4 in R; we must show that t is also in R. Now $w_1[A]$, $w_2[A]$, and $w_3[A]$ cannot all be distinct, since $|\operatorname{dom}(A)| = 2$. Assume that $w_1[A] = w_2[A]$; the proof is almost identical in the other two cases. Now $w_1[B] = w_2[B]$ since both equal $t[B]$, by (6.1) and (6.2). Thus $w_1[AB] = w_2[AB]$. Since AB is a key, it follows that $w_1 = w_2$. Denote the common value of w_1 and w_2 by w. We know that $w[BCD] = t[BCD]$ by (6.1), and that $w[BCE] = t[BCE]$ by (6.2). It follows that $w[BCDE] = t[BCDE]$. Now, $w_4[ACDE] = t[ACDE]$ by (6.4). By these last two facts, we know that both $w[CDE]$ and $w_4[CDE]$ equal $t[CDE]$. Hence $w[CDE] = w_4[CDE]$. Since CDE is a key, $w_4 = w$. But $w_4[A] = t[A]$ by (6.4). Hence $w[A] = t[A]$. Since we also already showed that $w[BCDE] = t[BCDE]$, it follows that $w[ABCDE] = t[ABCDE]$, that is, $w = t$. Thus t is indeed in R, which was to be shown. This concludes the example.

PROOF OF LEMMA 6.1. If $\Delta \models \sigma$, then $\Gamma \models \sigma$, since $\Delta \subseteq \Gamma$. So we need only show that if $\Gamma \models \sigma$, then $\Delta \models \sigma$. Assume that $\Gamma \models \sigma$, but that it is false that $\Delta \models \sigma$. We will derive a contradiction. Since it is false that $\Delta \models \sigma$, we know that there is a counterexample relation R for which Δ holds but σ fails. Since σ is either an FD or MVD, it is easy to see R contains a 2-tuple subrelation for which σ fails. That is, there are two tuples s_1 and s_2 of R, such that the relation S, which contains precisely these two tuples, does not obey σ. Clearly S, being a subrelation of R, obeys the set Δ of KDs. Thus S is a 2-tuple counterexample relation for which Δ holds but σ fails. (We note that a strong version of the result that says that there is a 2-tuple counterexample subrelation appears in [27, Lemma 9], the "2-tuple subrelation lemma.")

We define two new tuples t_1 and t_2, attribute by attribute, as follows. Let A be an arbitrary attribute. If $s_1[A] = s_2[A]$, then let $t_1[A] = t_2[A] = a$, where a is an arbitrary member of $\operatorname{dom}(A)$. If $s_1[A] \neq s_2[A]$, then let $t_1[A] = a$, and let $t_2[A] = b$, where a and b are distinct members of $\operatorname{dom}(A)$; this is possible since $|\operatorname{dom}(A)| \geq 2$. Let T be the relation containing precisely t_1 and t_2. For each attribute A, we have defined T so that the two tuples of T agree on attribute A (i.e., $t_1[A] = t_2[A]$) if and only if the two tuples of S agree on attribute A. Thus S and T are "isomorphic." In particular, since Δ holds in S and σ fails in S, also Δ holds in T and σ fails in T. Furthermore, we have constructed T so that all of the DDs in Θ hold in T. Hence Γ holds in T.

By assumption, $\Gamma \models \sigma$. But, we showed that Γ holds in T and σ fails in T. This is a contradiction. □

It is certainly possible for a set Γ of DDs and KDs to logically imply a fixed bound on the number of tuples. For example, KEY(A) and IN(A, {0, 1, 2, 3}) taken together logically imply that each instance may have no more than four tuples. One's first thought might be that the combinatorial curiosities exhibited by the schema in Example 6.3 were caused by such a phenomenon. However, this is *not* what "goes wrong" in this case. For, an instance of the schema in Example 6.3 may have an arbitrarily large number of tuples.

Example 6.3 shows us that Lemma 6.1 fails if we allow σ to be a JD. The next lemma (Lemma 6.4) gives a condition, analogous to Lemma 6.1, for the JD case. Before we can state Lemma 6.4, we need a few more definitions. Following Codd [10], we say that an attribute is *prime* if it is a member of a minimal key, that is, of a key with no proper subset that is a key. By $C(n, m)$, we mean the number of combinations of n objects, taken m at a time. ($C(n, m)$ is sometimes read as "n choose m.") Finally, by $[x]$, where x is a real number, we mean the greatest integer not exceeding x.

LEMMA 6.4. *Let Δ be a set of KDs, Θ a set of DDs, and $\Gamma = \Delta \cup \Theta$. Let n be the number of attributes. Assume that $|\, dom(A) \,| \geq 2$ for each nonprime attribute A, and that $|\, dom(A) \,| \geq C(n, [n/2])$ for each prime attribute A. Let σ be a JD. Then $\Delta \models \sigma$ if and only if $\Gamma \models \sigma$.*

The proof of Lemma 6.4 appears in the appendix. Our technique in the proof of Lemma 6.4 (and in the proof in Example 6.3) is in the spirit of the "chase" technique of Maier, Mendelzon, and Sagiv [23], except that unlike them, we must take into consideration the effects of bounded domains. We note also that by Stirling's formula, the term $C(n, [n/2])$ in Lemma 6.4 is asymptotic to $2^{n+0.5}/(\pi n)^{0.5}$.

As an example of Lemma 6.4, assume that there are five attributes, that $|\, \mathrm{dom}(A) \,| \geq 2$ for each nonprime attribute A, and that $|\, \mathrm{dom}(A) \,| \geq 10$ for each prime attribute A. If σ is an arbitrary JD, then $\Delta \models \sigma$ if and only if $\Gamma \models \sigma$.

We now present modified versions of BCNF, 4NF, and PJ/NF, in which the combinatorial effect of the DDs are explicitly taken into account.

Definition 6.5. A 1NF relation schema R^* is in BCNF′ if $\Gamma \models \sigma$ for each FD σ of R^*, where Γ is the set of DDs and KDs of R^*.

THEOREM 6.6. *Let R^* be a relation schema in which $|\, dom(A) \,| \geq 2$ for each attribute A. Then R^* is in BCNF if and only if it is in BCNF′.*

PROOF. Immediate from the definitions and from Lemma 6.1. □

Definition 6.7. A 1NF relation schema R^* is in 4NF′ if $\Gamma \models \sigma$ for each MVD σ of R^*, where Γ is the set of DDs and KDs of R^*.

THEOREM 6.8. *Let R^* be a relation schema in which $|\, dom(A) \,| \geq 2$ for each attribute A. Then R^* is in 4NF if and only if it is in 4NF′.*

PROOF. Immediate from the definitions and from Lemma 6.1. □

Definition 6.9. A 1NF relation schema R^* is in PJ/NF′ if $\Gamma \models \sigma$ for each JD σ of R^*, where Γ is the set of DDs and KDs of R^*.

THEOREM 6.10. *Let R^* be a relation schema in which $|\, dom(A) \,| \geq 2$ for each nonprime attribute A, and in which $|\, dom(A) \,| \geq C(n, [n/2])$ for each prime attribute A. Then R^* is in PJ/NF if and only if it is in PJ/NF′.*

PROOF. Immediate from the definitions and from Lemma 6.4. □

COROLLARY 6.11. *Let R^* be a relation schema in which $dom(A)$ is infinite for each attribute A. Then R^* is in PJ/NF if and only if it is in PJ/NF′.*

PROOF. Immediate from Theorem 6.10. □

COROLLARY 6.12. *Let R^* be a relation schema in which dom(A) is infinite for each attribute A. If R^* is in DK/NF, then it is in PJ/NF. Thus if all domains are infinite, then DK/NF $\Rightarrow$ PJ/NF $\Rightarrow$ 4NF $\Rightarrow$ BCNF.*

PROOF. Since R^* is in DK/NF, it follows immediately from the definitions that R^* is in PJ/NF′ (because every JD is a constraint). It then follows from Corollary 6.11 that R^* is in PJ/NF. The other implications (PJ/NF $\Rightarrow$ 4NF $\Rightarrow$ BCNF) hold in general, by Theorem 5.4. □

Example 6.2 shows that the assumption $|\text{dom}(A)| \geq 2$ is necessary in Theorems 6.6 and 6.8 since the schema in Example 6.2 is in BCNF′ and 4NF′ but not in BCNF or 4NF. Example 6.3 shows that it is not sufficient to replace the assumptions on $|\text{dom}(A)|$ in the statement of Theorem 6.10 by simply "$|\text{dom}(A)| \geq 2$ for each attribute A," since the schema in Example 6.3, in which $|\text{dom}(A)| \geq 2$ for each attribute A, is in PJ/NF′ but not in PJ/NF. We note that it is not hard to modify Example 6.3 to obtain, for each integer k, a schema in which $|\text{dom}(A)| \geq k$ for each attribute A, and which is in PJ/NF′ but not in PJ/NF.

A relation schema that violates the assumption of Theorems 6.6 and 6.8 would certainly be unreasonable. For, we would either have $|\text{dom}(A)| = 0$ for some attribute A (in which case the schema would have no valid instance that is nonempty), or else we would have $|\text{dom}(A)| = 1$ for some attribute A (in which case there is no reason to include A as an attribute). So, by Theorems 6.6 and 6.8, it follows that BCNF and BCNF′ are the same for all reasonable schemata, as are 4NF and 4NF′. However, a schema could certainly violate the assumptions of Theorem 6.10 and still be reasonable. Our viewpoint on PJ/NF versus PJ/NF′ is that PJ/NF′ is fundamentally more natural than PJ/NF since the effects of bounded domain sizes are quite real and should be taken into account. Another lesson we should learn from these results is that a reasonable restriction to place on a DK/NF schema is that $|\text{dom}(A)| \geq 2$.

THEOREM 6.13. *DK/NF $\Rightarrow$ PJ/NF′ $\Rightarrow$ 4NF′ $\Rightarrow$ BCNF′.*

PROOF. As before, it is immediate from the definitions that DK/NF $\Rightarrow$ PJ/NF′ since every JD is a constraint. The fact that PJ/NF′ $\Rightarrow$ 4NF′ follows from the fact, noted in Section 2, that each MVD can be represented by a JD. The proof that 4NF′ $\Rightarrow$ BCNF′ requires some work. Let R^* be a relation schema in 4NF′; we shall show that it is in BCNF′. Assume not; we shall derive a contradiction. If $|\text{dom}(C)| = 0$ for some attribute C, then R^* is in BCNF′. Thus for each attribute C, we can assume that $|\text{dom}(C)| \geq 1$. Let Γ be the set of DDs and KDs of R^*. Since R^* is not in BCNF′, there is an FD σ of the schema such that Γ does not logically imply σ. We can assume without loss of generality that the right-hand side of the FD σ is a single attribute. Let us write σ as $U \rightarrow A$, where A is a single attribute.

Let V be the set of those attributes C such that $|\text{dom}(C)| = 1$. We define two tuples s_1 and s_2 as follows. For each attribute C in $U \cup V$, we set $s_1[C] = s_2[C] = a$, where a is an arbitrary member of dom(C). For each remaining attribute C, we set $s_1[C] = a$ and $s_2[C] = b$, where a and b are distinct members of dom(C).

Note that A is not in $U \cup V$, or else $\Gamma \vDash U \rightarrow A$ would hold. Thus $s_1[A] \neq$

$s_2[A]$. We now show that there is some other attribute B, distinct from A, which is not in $U \cup V$. Assume not. Then, since $U \rightarrow A$ and $U \rightarrow V$ are constraints in R^*, it would follow that KEY(U) is a constraint of R^*. But then, since KEY(U) would be in Γ, it would follow that $\Gamma \models U \rightarrow A$, a contradiction. Thus we have shown that B, as described above, exists. Then $s_1[B] \neq s_2[B]$.

Let S be a 2-tuple relation containing precisely s_1 and s_2. We know that

$$\begin{aligned} s_1[U] &= s_2[U] \\ s_1[A] &\neq s_2[A] \\ s_1[B] &\neq s_2[B]. \end{aligned} \tag{6.5}$$

Since A and B are distinct attributes not in U, it follows easily from (6.5) that the MVD $U \twoheadrightarrow A$ fails in S. Let KEY(K) be an arbitrary KD in Γ. Then $s_1[K] \neq s_2[K]$, or else, by definition of s_1 and s_2, we would have $K \subseteq U \cup V$, which would imply that KEY(U) is in Γ, which would imply that $\Gamma \models \sigma$, a contradiction. Hence every KD in Γ holds in S. Also, by construction, every DD in Γ holds in S. So, Γ holds in S. We already showed that $U \twoheadrightarrow A$ fails in S. So, S is a counterexample relation that shows that $\Gamma \models U \twoheadrightarrow A$ is false. But on the other hand, $U \twoheadrightarrow A$ is a constraint of R^* (since $U \rightarrow A$ is). So, since $U \twoheadrightarrow A$ is an MVD of R^*, and since R^* is in 4NF′, it follows that $\Gamma \models U \twoheadrightarrow A$. This is a contradiction. □

We close this section with some remarks on Smith's (3,3) normal form ((3,3)NF) [28]. One formalization of the intent of (3,3)NF is to say that a 1NF schema is in (3,3)NF if those "embedded" FDs that hold in a subrelation obtained by splitting (in exactly the same sense as in Example 3.7) are all logical consequences of Γ. That is, (3,3)NF means that Γ logically implies each of the schema's constraints that are of the form

$$\forall t_1 \forall t_2((\psi(t_1) \wedge \psi(t_2) \wedge (t_1[U] = t_2[U])) \Rightarrow ((t_1[V] = t_2[V])) \tag{6.6}$$

for arbitrary ψ, U, and V. It is not hard to see that (6.6) says that "the relation consisting of those tuples t that satisfy $\psi(t)$ obeys the FD $U \rightarrow V$." It follows from what we have said that DK/NF $\Rightarrow$ (3,3)NF, since (6.6) is simply another constraint.

7. A PARADIGM FOR NORMAL FORMS

Earlier, we discussed the fundamental nature of domains and keys for relational databases. Thus it is reasonable to assume that a database management system should have the power to enforce our KDs and at least a limited version of our DDs (such as those DDs that state that the domain of a given attribute is, say, strings of six characters). Since domains and keys are already being supported for other reasons, it is very good if *all* of the constraints of a schema can be enforced by simply enforcing domains and keys (which is precisely what DK/NF is all about). That is, if a schema is in DK/NF, then there is *no extra overhead* in supporting the constraints of a DK/NF schema. In fact, the database management system may well not have the capability for supporting constraints *other than* those that are automatically enforced by enforcing domains and keys.

We might define new classes of normal forms (for relation schemata and for database schemata) by selecting a certain class $\mathscr{C}$ of constraints (including

interrelational constraints) that are somehow primitive, or "easy to enforce." To properly define "easy to enforce," we need a complexity theory for database constraints. Unlike traditional complexity theory, the primary cost is *not* the computation time or memory, but rather the number of page faults. A schema is in $\mathscr{C}$ *normal form* if the set of constraints of the schema is precisely the set of logical consequences of some set of constraints, each of which is in the class $\mathscr{C}$. For example, if $\mathscr{C}$ is the class of all DDs and KDs, then $\mathscr{C}$ normal form is the same as DK/NF. It is easy to see that if $\mathscr{C}_1 \subseteq \mathscr{C}_2$, then $\mathscr{C}_1$ normal form implies $\mathscr{C}_2$ normal form.

If the database management system has the capability of enforcing inclusion dependencies as interrelational constraints, then it is reasonable to consider allowing such dependencies as constraints *within* a relation schema (as in the schema of Example 3.10).

Another natural class of simple constraints are *single-tuple dependencies.* These are constraints of the form $\forall t \phi(t)$, where $\phi(t)$ is quantifier free in some language. Single-tuple dependencies involve only the tuple to be inserted and no other tuples. Thus whatever the language of $\phi(t)$ is, $\phi(t)$ should be computable, without referencing any other tuples. In Example 3.6, the last constraint (which says that employees with status 0 have a salary of at most \$50,000) is an example of a single-tuple dependency. If we allow $\phi(t)$ to be an arbitrary computable predicate, then computable domain dependencies are special cases of single-tuple dependencies. INGRES [32] and MAGNUM [33] are designed to enforce single-tuple dependencies. IMS [20] supports "exits," where a side computation can be done before an insertion or an update; the side computation cannot look at any other tuples other than the one to be inserted, but the computation can be arbitrarily complex. By this means, IMS can enforce arbitrary single-tuple dependencies.

Another example of a constraint that might be easy to enforce is a cardinality constraint on the number of tuples in a relation, such as "there are at least 3 tuples," or "there are at most 117 tuples." A special case that one might want is the constraint that says "the relation is not empty" (i.e., "there is at least one tuple"). Depending on the database management system, these cardinality constraints might be easily implemented by associating a tuple counter with each relation; the counter is updated whenever the relation is.

8. CONCLUSIONS

We have defined a new normal form, called domain-key normal form (DK/NF), which is based only on the primitive concepts of domain and key, along with the general concept of a "constraint," We have also presented a formal definition of an insertion anomaly and a deletion anomaly. We have shown that a satisfiable 1NF relation schema is in DK/NF if and only if it has no insertion or deletion anomalies. We have shown how traditional normal forms might be modified "in the spirit of DK/NF," to take into consideration the combinatorial consequences of bounded domain sizes. The effect of the modifications, as we have shown, is that the original normal form and its modified counterpart are equivalent provided no domain size is too small. Each of these modified normal forms is implied by DK/NF.

Our results indicate that it is reasonable to insist that a relation schema obey other properties besides being in DK/NF. These restrictions include:

(1) The schema is in 1NF (we have discussed nothing but 1NF schemata in this paper, but DK/NF could be defined without this assumption).
(2) The schema is satisfiable (i.e., it has at least one valid instance).
(3) $|\text{dom}(A)| \geq 2$ for each attribute A.
(4) All domain constraints are "simple," that is, easily enforceable.

We consider it an important research problem to find other natural restrictions that should be imposed.

We have discussed paradigms for normal forms (both for *relation* schemata and for *database* schemata). All constraints (including interrelational constraints) should be "simple." A good example of a simple and important interrelational constraint is the inclusion dependency.

A "database complexity theory" is needed, to clarify what a "simple" constraint is and what a "simple" transformation from one database schema into another is. (The concept of a simple transformation is needed for dealing with the question of converting a database schema into a DK/NF database schema.) "Simple" should somehow mean "fast for a database management system to perform." Such a complexity notion might lead to the isolation of a set of primitive operations that a database management system can and should support easily.

Our viewpoint on DK/NF is that in the best of all possible worlds, every relation schema in a database would be in DK/NF (and there would be no interrelational constraints, or perhaps only easily enforceable interrelational constraints, such as inclusion dependencies). In practice, this goal is unlikely to be attained. A good research problem might be to define reasonable ways in which we can "set our sights lower"; one possibility is to consider weaker normal forms than DK/NF. From a purely practical point of view, we feel that in the database design process, the designer should *strive* to obtain DK/NF.

APPENDIX

The purpose of this appendix is to prove Lemma 6.4, which is as follows.

Lemma 6.4. *Let* Δ *be a set of KDs,* Θ *a set of DDs, and* $\Gamma = \Delta \cup \Theta$. *Let* n *be the number of attributes. Assume that* $|\,dom(A)\,| \geq 2$ *for each nonprime attribute* A, *and that* $|\,dom(A)\,| \geq C(n, \lceil n/2 \rceil)$ *for each prime attribute* A. *Let* σ *be a JD. Then* $\Delta \models \sigma$ *if and only if* $\Gamma \models \sigma$.

If σ is a JD $*\{X_1, \ldots, X_r\}$, where no two X_i are the same, then we say that σ has r *components.* We begin by proving the following lemma.

LEMMA A1. *Let* Δ *be a set of KDs,* Θ *a set of DDs, and* $\Gamma = \Delta \cup \Theta$. *Let* σ *be a JD with* r *components. Assume that* $|\,dom(A)\,| \geq 2$ *for each nonprime attribute* A, *and that* $|\,dom(A)\,| \geq r$ *for each prime attribute* A. *Then* $\Delta \models \sigma$ *if and only if* $\Gamma \models \sigma$.

PROOF. If $\Delta \models \sigma$, then $\Gamma \models \sigma$ since $\Delta \subseteq \Gamma$. So, we need only show that if $\Gamma \models \sigma$, then $\Delta \models \sigma$. Assume that $\Gamma \models \sigma$, but that it is false that $\Delta \models \sigma$. We shall derive a contradiction. Since it is false that $\Delta \models \sigma$, we know that there is a counterexample

relation R for which Δ holds and σ fails. Let us write σ as $*\{X_1, \ldots, X_r\}$. Since σ fails in R, there are tuples $s_1, \ldots, s_r$ of R (not necessarily distinct) and a tuple u not in R such that $s_i[X_i] = u[X_i]$ for each i $(1 \leq i \leq r)$. Let S be a relation containing only the tuples $s_1, \ldots, s_r$ of R. Then σ fails in S. However, Δ holds in S since it holds in R. We now define new tuples $t_1, \ldots, t_r$, attribute by attribute.

Case 1. A is a prime attribute. We can define $t_1[A], \ldots, t_r[A]$ so that

(a) $t_i[A] \in \text{dom}(A)$, $1 \leq i \leq r$;
(b) $t_i[A] = t_j[A]$ if and only if $s_i[A] = s_j[A]$, $1 \leq i \leq r$, $1 \leq j \leq r$.

The reason that (b) is possible is that $|\text{dom}(A)| \geq r$.

Case 2. A is a nonprime attribute. Let a and b be two distinct members of $\text{dom}(A)$. Recall that u is a tuple not in S such that $s_i[X_i] = u[X_i]$, $1 \leq i \leq r$. For each i $(1 \leq i \leq r)$, set

$$t_i[A] = a \text{ if } s_i[A] = u[A].$$

$$t_i[A] = b \text{ if } s_i[A] \neq u[A].$$

Let T be a relation containing precisely the tuples $t_1, \ldots, t_r$. Our goal is to show that Γ holds in T and that σ fails in T. This will give us a contradiction, since, by assumption, $\Gamma \vDash \sigma$. Clearly Θ, the set of DDs, holds in T, by construction. So, the proof is complete if we show that each KD in Δ holds in T and that σ fails in T.

Let KEY(K) be an arbitrary KD in Δ. Let K' be a minimal subset of K such that KEY(K') is in Δ. To show that KEY(K) holds in T, we need only show that KEY(K') holds in T. Since K' contains only prime attributes, it follows from our construction that for each i, j $(1 \leq i \leq r, 1 \leq j \leq r)$, we have $t_i[K'] = t_j[K']$ if and only if $s_i[K'] = s_j[K']$. But $s_i[K'] \neq s_j[K']$ when $i \neq j$, since KEY(K') holds in S. Thus $t_i[K'] \neq t_j[K']$ if $i \neq j$. So, KEY(K') holds in T, which was to be shown.

Finally, we must show that σ does not hold in T. We now define a tuple v, attribute by attribute. Let A be an arbitrary attribute. We know that there is some i such that $s_i[A] = u[A]$ (we simply find i such that $A \in X_i$. Then $s_i[X_i] = u[X_i]$, so $s_i[A] = u[A]$). Set $v[A] = t_i[A]$. We must show that $v[A]$ is well defined. That is, we must show that if $s_i[A]$ and $s_j[A]$ both equal $u[A]$, then $t_i[A] = t_j[A]$. But this is easily verified from our definitions, whether A is prime or nonprime.

We now show that for each attribute A, we have

$$t_i[A] = v[A] \text{ if and only if } s_i[A] = u[A]. \tag{A1}$$

If $s_i[A] = u[A]$, then $t_i[A] = v[A]$ by definition of v. Conversely, assume that $t_i[A] = v[A]$. We wish to show that $s_i[A] = u[A]$. Assume that $s_i[A] \neq u[A]$; we shall derive a contradiction. Find j such that $A \in X_j$. Then $s_j[A] = u[A]$. By definition of v, we know that $t_j[A] = v[A]$. So $t_i[A] = t_j[A]$, since both equal $v[A]$. There are now two cases, depending on whether or not A is prime. Assume first that A is prime. Since $t_i[A] = t_j[A]$, it follows from the definition of $t_i[A]$ and of $t_j[A]$ that $s_i[A] = s_j[A]$. This is a contradiction, since $s_i[A] \neq u[A]$ and $s_j[A] = u[A]$. Now assume that A is nonprime. Since $s_i[A] \neq u[A]$ and $s_j[A] = u[A]$, it follows from the definition of $t_i[A]$ and of $t_j[A]$ that $t_i[A]$ and $t_j[A]$ are unequal. This is a contradiction. Thus (A1) is proved in both cases.

Since $s_i[X_i] = u[X_i]$ $(1 \leq i \leq r)$, it follows from (A1) that $t_i[X_i] = v[X_i]$ $(1 \leq i \leq r)$. Hence to show that σ does not hold in T, we need only show that v is not in T. If v were in T, then for some i, we would have $v = t_i$. But then, by (A1), it would follow that $u = s_i$. This is impossible since u is not in S. So σ does not hold in T. This was to be shown. □

Definition A2. A JD $*\{X_1, \ldots, X_r\}$ is *irreducible* if for no i, j with $i \neq j$ is it true that $X_i \subseteq X_j$.

LEMMA A3. *Every JD is equivalent to an irreducible JD.*

PROOF. As we now show, the JD $*\{X_1, \ldots, X_r\}$ is equivalent to an irreducible JD, which is obtained from the original JD by "removing" each component X_i that is a proper subset of another component X_j. (We do not need to consider the case where two components are equal since we are thinking of $\{X_1, \ldots, X_r\}$ as a *set*, which by definition has no duplicates.) The reason for the equivalence is as follows. The JD $*\{X_1, \ldots, X_r\}$ holds for a relation R precisely if $R = R[X_1] * \ldots * R[X_r]$. If $X_i \subseteq X_j$, then $R[X_i] * R[X_j] = R[X_j]$. Thus since the join is commutative and associative, $R = R[X_1] * \ldots * R[X_r]$ if and only if $R = R[X_1] * \ldots * R[X_{i-1}] * R[X_{i+1}] * \ldots * R[X_r]$. This latter equality holds if and only if the JD obtained from the original JD by removing the X_i component holds in R. This process is repeated until an irreducible JD is obtained. □

We note that Lemma A3 is an immediate consequence of results by Aho, Sagiv, and Ullman [2].

LEMMA A4. *The maximum number of components an irreducible JD can have is $C(n, [n/2])$, where n is the number of attributes.*

PROOF. This follows immediately from a theorem of Sperner [31], which states that the maximum number of pairwise incomparable (under $\subseteq$) subsets of a set of n elements is $C(n, [n/2])$. □

We can now prove Lemma 6.4. By Lemma A3, we need only consider irreducible JDs. By Lemma A4, no irreducible JD can have more than $C(n, [n/2])$ components. Thus if r is as in Lemma A1, then the maximal value of r we need to consider is $C(n, [n/2])$. Lemma 6.4 then follows.

ACKNOWLEDGMENTS

The author is grateful to Bill Armstrong, Phil Bernstein, Larry Carter, Ted Codd, Chris Date, Nat Goodman, Dick Karp, Jorma Rissanen, John Smith, John Sowa, and Jeff Ullman for helpful comments. He is especially grateful to Jean-Marie Cadiou for asking the probing questions which inspired this research.

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. AHO, A. V., SAGIV, Y., AND ULLMAN, J. D. Equivalences among relational expressions. *SIAM J. Comput. 8*, 2 (May 1979), 218–246.
3. ARMSTRONG, W. W. Dependency structures of database relationships. In *Proc. 1974 IFIP Congr.* North-Holland, Amsterdam, 1974, pp. 580–583.

4. ARMSTRONG, W. W., AND DELOBEL, C. Decompositions and functional dependencies in relations. *ACM Trans. Database Syst. 5*, 4 (Dec. 1980), 404–430.
5. ASTRAHAN, M. M., ET AL. System R: A relational approach to database management. *ACM Trans. Database Syst. 1*, 2 (June 1976), 97–137.
6. BERNSTEIN, P. A. Synthesizing third normal form relations from functional dependencies. *ACM Trans. Database Syst. 1*, 4 (Dec. 1976), 277–298.
7. CADIOU, J.-M. On semantic issues in the relational model of data. In *Proc. Int. Symp. Mathematical Foundations of Computer Science*, Gdansk, Poland, *Lecture Notes in Computer Sciences*, Springer-Verlag, Heidelberg, Sept. 1975.
8. CHURCH, A. A note on the Entscheidungsproblem. *J. Symbolic Logic 1*, 40–41. Correction, 101–102.
9. CODD, E. F. A relational model of data for large shared data banks. *Commun. ACM 13*, 6 (June 1970), 377–387.
10. CODD, E. F. Further normalization of the database relational model. In *Data Base Systems*, Courant Computer Science Symposia 6. Prentice-Hall, Englewood Cliffs, N.J., 1972, pp. 65–98.
11. CODD, E. F. Recent investigations in relational database systems. In *Proc. 1974 IFIP Congr.* North-Holland, Amsterdam, 1974, pp. 1017–1021.
12. CODD, E. F. Extending the database relational model to capture more meaning. *ACM Trans. Database Syst. 4*, 4 (Dec. 1979), 397–434.
13. DATE, C. J. *An Introduction to Database Systems*, 2nd ed. Addison-Wesley, Reading, Mass., 1977.
14. DAYAL, U., AND BERNSTEIN, P. A. The fragmentation problem: Lossless decomposition of relations into files. Tech. Rep. CCA-78-13, Computer Corporation of America, Cambridge, Mass., Nov. 15, 1978.
15. DELOBEL, C. Contribution theorique a la conception des systemes d'information. Ph.D. Dissertation, Univ. Grenoble, Grenoble, France, 1973.
16. FAGIN, R. Multivalued dependencies and a new normal form for relational databases. *ACM Trans. Database Syst. 2*, 3 (Sept. 1977), 262–278.
17. FAGIN, R. Normal forms and relational database operators. In *Proc. ACM SIGMOD*, 1979, pp. 153–160.
18. FAGIN, R. Horn clauses and database dependencies. In *Proc. ACM SIGACT Symp. Theory of Computation*, 1980, pp. 123–134.
19. IBM CORPORATION. *Generalized Information System/Virtual Storage, User's Guide.* SH20-9036.
20. IBM CORPORATION. *Information Management System/Virtual Storage Utilities Reference Manual.* SH20-9029.
21. IBM CORPORATION. *Query by Example, User's Guide.* SH20-2078.
22. KANELLAKIS, P. C. On the computational complexity of cardinality constraints in relational databases. *Inf. Process. Lett. 11*, 2 (Oct. 1980), 98–101.
23. MAIER, D., MENDELZON, A., AND SAGIV, Y. Testing implications of data dependencies. *ACM Trans. Database Syst. 4*, 4 (Dec. 1979), 455–469.
24. NICOLAS, J. M. Mutual dependencies and some results on undecomposable relations. In *Proc. 4th Conf. Very Large Data Bases*, West Berlin, 1978, pp. 360–367.
25. RISSANEN, J. Independent components of relations. *ACM Trans. Database Syst. 2*, 4 (Dec. 1977), 317–325.
26. RISSANEN, J. Theory of relations for databases—A tutorial survey. In *Proc. 7th Symp. Mathematical Foundations of Computer Science*, 1978, *Lecture Notes in Computer Science*, vol. 64, Springer-Verlag, pp. 537–551.
27. SAGIV, Y., DELOBEL, C., PARKER, D. S., AND FAGIN, R. An equivalence between relational database dependencies and a fragment of propositional logic. *J. ACM 28*, 2 (April 1981), 435–453.
28. SMITH, J. M. A normal form for abstract syntax. In *Proc. 4th Conf. Very Large Data Bases*, West Berlin, 1978, pp. 156–162.
29. SMITH, J. M., AND SMITH, D. C. P. Database abstractions: Aggregation. *Commun. ACM 20*, 6 (June 1977), 405–413.
30. SOFTWARE AG. *ADABAS Introductory Manual.* Software AG North America Inc., 11800 Sunrise Valley Drive, Reston, Va. 22091.

31. SPERNER, E. Ein satz über Untermengen einer endlichen Menge. *Math. Zeitschrift 27* (1928), 544–548.
32. STONEBRAKER, M., WONG, E., KREPS, P., AND HELD, G. The design and implementation of INGRES. *ACM Trans. Database Syst. 1*, 3 (Sept. 1976), 189–222.
33. TYMSHARE INC. *MAGNUM Reference Manual*, Nov. 1975.
34. ZANIOLO, C. Design of relational views over network schemas. In *Proc. ACM SIGMOD*, 1979, pp. 179–190.

Received July 1979; revised June 1980; accepted October 1980