WHAT DOES BOYCE-CODD NORMAL FORM DO?[†]

Philip A. Bernstein
Nathan Goodman

Aiken Computation Laboratory
Harvard University
Cambridge, MA 02138

Abstract

Normalization research has concentrated on defining normal forms for database schemas and developing efficient algorithms for attaining these normal forms. It has never been *proved* that normal forms are *good*, i.e. that normal forms are beneficial to database users. This paper considers one of the earliest normal forms (Boyce Codd normal form [Cod2]) whose benefits are intuitively understood. We formalize these benefits and attempt to prove that the normal form attains them. Instead we prove the opposite: *Boyce-Codd normal form fails to meet its goals except in trivial cases.* This counterintuitive result is a consequence of the "universal relation assumption" upon which normalization theory rests. Normalization theory will remain an isolated theoretical area, divorced from database practice, until this assumption is circumvented.

1. Introduction

Normalization is a schema design process. The input is a schema (i.e., database description) that describes an application; the output is another schema that describes the same application, but in a "better" way. Normalization theory provides mathematical tools for defining schemas, for characterizing the application that a schema represents, and for specifying the form a "good" schema must satisfy. Most normalization research has focused on defining good forms for schemas (*normal forms*) and developing efficient algorithms for attaining these normal forms. However, normalization theory does not provide tools for explaining *why* normal forms are good. Presumably, normal forms help database users in some way, but these benefits have never been characterized within the formal framework of normalization theory. This paper attempts such a characterization for one particular normal form, namely Boyce Codd normal form.

[†]This work was supported by the National Science Foundation under Grant Number MCS-77-05314 and by the Advanced Research Projects Agency of the Department of Defense, Contract Number N00039-78-G-0020.

The benefits of Boyce Codd normal form (BCNF) are well understood at an intuitive level. BCNF and its precursor, third normal form (3NF), were introduced by Codd to eliminate "anomalous" update* behavior of relations. BCNF accomplishes this by "placing independent relationships into independent relations." (A more precise characterization appears later.) BCNF is widely recommended in practice for this purpose [Date, Mar], and there is little doubt that BCNF is a good normal form in many practical situations.

This paper formalizes the intuitive benefits of BCNF and attempts to prove that these benefits are attained. The attempt fails. Instead we prove that *BCNF does not meet its goals, except in trivial cases.* This result is quite disturbing because it contradicts BCNF's intuitive appeal and empirically observed benefits. Either the theory is wrong, or database practitioners are making a serious error by using BCNF. Our opinion is that the theory is at fault--that normalization theory, as currently constituted, is inadequate to characterize the practical use and benefits of normal forms. The source of inadequacy can be traced to the "universal relation assumption" upon which normalization theory is based.

On the surface, this paper is an analysis of Boyce Codd normal form. Our deeper purpose is to draw attention to a missing link in normalization theory. For normalization theory to be well grounded in the database problems it intends to solve, it must be possible to use the theory to prove the benefits of normal forms. Unfortunately, it appears that normalization theory, as currently formulated, cannot do this. Until this missing link is filled in, normalization theory will be incapable of explaining what Boyce Codd normal form does.

*The word "update" is used generically to denote insertions, deletions, or replacements (i.e., modification) of tuples in a relation.

2. Terminology

2.1 Relations and Databases

In formalizing relational databases we distinguish the time-invariant description of a relation vs. its contents at a particular moment. The description of a relation is called a *relation schema* and consists of a set of *attributes*, and a set of *functional dependencies*. $\underline{R} = <ATTR,F>$ denotes a relation schema $\underline{R}$ with attributes ATTR, and functional dependencies F. Figure 2.1 illustrates a relation schema. The contents of a relation is called a *relation state*, and may be visualized as a table of data. Figure 2.2 illustrates a relation state.

Relation Schema:

$$\underline{R} = <\{E\#,\ JC,\ D\#,\ M\#,\ CT\},\ \{E\# \to JC,\ D\#,\ M\#,\ CT;\ D\# \to M\#,\ CT;\ M\# \to D\#,\ CT\}>$$

Description of Attributes:

E#	employee number
JC	job code
D#	department number
M#	employee number of manager
CT	contract type

Figure 2.1. An Example Relation Schema.

Relation Schema: as defined in Fig. 2.1

Relation: R

E#	JC	D#	M#	CT
1	a	x	11	g
2	c	x	11	g
3	a	y	12	n
4	b	x	11	g
5	b	y	12	n
6	c	y	12	n
7	a	z	13	n
8	c	z	13	n

Figure 2.2. An Example Relation.

Notationally, states of relation schema $\underline{R}$ are represented by R with possible superscript. Elements of R (called *tuples*) are denoted r with possible superscript. If $X \subseteq ATTR$, we use $r[X]$ to denote the projection of r on the attributes in X, and $R[X] = \{r[X] \mid r \in R\}$.

A *database schema* is a set of relation schemas and is denoted $\underline{D} = \{\underline{R}_i = <ATTR_i,F_i> \mid i = 1,\ldots,n\}$. A database state D for $\underline{D}$ is an assignment of relation states to the relation schemas of $\underline{D}$. $D(\underline{R}_i)$ denotes the relation state assigned by D to $\underline{R}_i$, for $i = 1,\ldots,n$. If D and D' are both database states for database schema $\underline{D}$, we define $D \leq D'$ iff $D(\underline{R}_i) \subseteq D'(\underline{R}_i)$ for $i = 1,\ldots,n$; clearly, $\leq$ is a partial order.

2.2 Functional Dependencies

A *functional dependency* (abbr. FD) is a statement of the form $f{:}X \to Y$, where f is the name of the FD, and X,Y are sets of attributes. f is defined on relation schema $\underline{R} = <ATTR,F>$ if $X,Y \subseteq ATTR$. We shall assume throughout this paper that every $f \in F$ is defined on $\underline{R}$. If f is defined on $\underline{R}$ we interpret it as a predicate on states of $\underline{R}$; f(R) is true if for all $r,r' \in R$, whenever $r[X] = r'[X]$, $r[Y] = r'[Y]$ as well.

A relation state R of schema $\underline{R} = <ATTR,F>$ is *consistent* if every $f \in F$ is true in R. For example, the state depicted in Fig. 2.2 is consistent. The set of all consistent states of $\underline{R}$ is denoted $DOM(\underline{R})$.

Given that a set of FDs, F, holds in state R, one can infer other FDs that must hold in R as well. The set of all such FDs can be constructed syntactically using a system of *inference rules* [DC], see Figure 2.3. The set of all FDs derivable from F using the inference rules of Fig. 2.3 is called the *closure* of F and is denoted F^+.

If $Y \subseteq X$ then $X \to Y$

If $Z \subseteq W$ and $X \to Y$ then $XW \to YZ$

If $X \to Y$ and $Y \to Z$ then $X \to Z$

Figure 2.3. Inference Rules for Functional Dependencies.

Within the confines of a single relation schema, F^+ is precisely the set of FDs *implied by* F: i.e., $f \in F^+$ iff f holds in every state of $\underline{R}$ in which F holds [Arm]. This fact is called the *completeness property* of FD-closure [BFH].

FDs also enjoy the *uniqueness property* within a single relation schema, which states that syntactically identical FDs are semantically equivalent as well [Bern].

2.3 The Universal Relation Assumption: Schema Equivalence

Intuitively, two schemas are equivalent if they represent the same external application. For relation schemas this notion may be formalized in terms of consistent states: relation schema $\underline{R} = <ATTR,F>$ and $\underline{R}' = <ATTR',F'>$ are *equivalent* if $DOM(\underline{R}) = DOM(\underline{R}')$. As a consequence of completeness and uniqueness, $DOM(\underline{R}) = DOM(\underline{R}')$ iff $ATTR = ATTR'$ and $F^+ = F'^+$. Hence, when normalizing $\underline{R}$ one can substitute F' for F whenever $F'^+ = F^+$, since the resulting schema <ATTR,F'> remains equivalent to $\underline{R}$.

All normalization algorithms depend on this fact. Bernstein [Bern], for example, exploits it to substitute a "reduced" set of FDs for the ones supplied by the user. Without this substitution, many schemas cannot be normalized due to redundancies in the way FDs are expressed in the schema [Bern].

However, completeness and uniqueness do not hold in multi-relation databases generally. Consider Figure 2.4. The schemas $\underline{D}^1$, $\underline{D}^2$, and $\underline{D}^3$ "contain" the same FDs--i.e., $F^{1+} = F^{2+} = F^{3+}$--yet these three schemas differ qualitatively in representational capability. $\underline{D}^1$ can only represent universities in which professors and teaching assistants must teach courses in their own department; $\underline{D}^2$ permits teaching assistants to be drawn from other departments, but not professors; and $\underline{D}^3$ permits complete flexibility in the assignment of teaching assistants and professors. These schemas do not represent the same external application, and hence are not equivalent, even though they contain the same attributes and FDs.

$$
\begin{aligned}
\underline{D}^1 = \{&\underline{\text{TEACH}}^1 = <\{\text{COURSE,PROF,TA,DEPT}\},\\
&\quad\{\text{COURSE}\to\text{PROF};\ \text{COURSE}\to\text{TA};\\
&\quad\ \text{COURSE}\to\text{DEPT};\ \text{PROF}\to\text{DEPT};\\
&\quad\ \text{TA}\to\text{DEPT}\}>\}\\
\underline{D}^2 = \{&\underline{\text{TEACH}}^2 = <\{\text{COURSE,PROF,TA}\},\\
&\quad\{\text{COURSE}\to\text{PROF};\ \text{COURSE}\to\text{TA}\}>,\\
&\underline{\text{FACULTY}}^2 = <\{\text{COURSE,PROF,DEPT}\},\\
&\quad\{\text{COURSE}\to\text{DEPT};\ \text{PROF}\to\text{DEPT}\}>,\\
&\underline{\text{STUDENTS}}^2 = <\{\text{TA,DEPT}\},\ \{\text{TA}\to\text{DEPT}\}>\}\\
\underline{D}^3 = \{&\underline{\text{TEACH}}^3 = <\{\text{COURSE,PROF,TA}\},\\
&\quad\{\text{COURSE}\to\text{PROF};\ \text{COURSE}\to\text{TA}\}>,\\
&\underline{\text{FACULTY}}^3 = <\{\text{PROF,DEPT}\},\quad \{\text{PROF}\to\text{DEPT}\}>,\\
&\underline{\text{STUDENTS}}^3 = <\{\text{TA,DEPT}\},\quad \{\text{TA}\to\text{DEPT}\}>,\\
&\underline{\text{CLASSES}}^3 = <\{\text{COURSE,DEPT}\},\ \{\text{COURSE}\to\text{DEPT}\}>\}
\end{aligned}
$$

Figure 2.4. Completeness and Uniqueness Do Not Hold in Multi-Relation Databases.

This problem is quite serious. It means that the definition of schema equivalence used to construct a normalized schema $\underline{D}$ is not semantically valid. To circumvent this problem the *universal relation assumption* is invoked.

Let $\underline{D} = \{\underline{R}_i = <\text{ATTR}_i, F_i> | i = 1,\dots,n\}$ be a database schema and let D be a database state for $\underline{D}$. D *satisfies the universal relation assumption* (or is UR-*consistent*) if there exists a relation state U for schema $\underline{U} = <\text{ATTR},\emptyset>$ such that $U[\text{ATTR}_i] = D(\underline{R}_i)$ for $i = 1,\dots,n$. D is *consistent* if it is UR-consistent, and $D(\underline{R}_i) \in \text{DOM}(\underline{R}_i)$ for $i = 1,\dots,n$. $\text{DOM}(\underline{D})$ denotes the set of all consistent states of $\underline{D}$.

By requiring that database states satisfy the UR-assumption, we can attain the desired notion of schema equivalence. Any UR-consistent state D is precisely representable by $U = *_{i=1}^{n} D(\underline{R}_i)$, since $U[\text{ATTR}_i] = D(\underline{R}_i)$ for $i = 1,\dots,n$. Therefore $\text{DOM}(\underline{D})$ is precisely representable by the set $\text{REP}(\underline{D}) = \{U | \exists D \in \text{DOM}(\underline{D}),\ U = *_{i=1}^{n} D(\underline{R}_i)\}$. Two schemas $\underline{D}$ and $\underline{D}'$ represent the same external application, i.e., are *equivalent under the* UR-*assumption* if $\text{REP}(\underline{D}) = \text{REP}(\underline{D}')$. But $\text{REP}(\underline{D}) = \text{DOM}(\underline{U} = <\text{ATTR},F>)$ where $\text{ATTR} = (\cup_{i=1}^{n} \text{ATTR}_i)$ and $F = (\cup_{i=1}^{n} F_i)$, and $\text{REP}(\underline{D}') = \text{DOM}(\underline{U}' = <\text{ATTR}',F'>)$ where $\text{ATTR}' = (\cup_{i=1}^{n'} \text{ATTR}_i')$ and $F' = (\cup_{i=1}^{n'} F_i')$. Consequently $\text{REP}(\underline{D}) = \text{REP}(\underline{D}')$ iff $\text{DOM}(\underline{U}) = \text{DOM}(\underline{U}')$ iff $\text{ATTR} = \text{ATTR}'$ and $F^+ = F'^+$, which is precisely the notion of schema equivalence required by normalization theory.

Finally we observe that the FDs that hold in $D(\underline{R}_i)$ for every $D \in \text{DOM}(\underline{D})$ are $F^+[\text{ATTR}_i]$, not simply F_i^+. We say that $D(\underline{R}_i)$ is FD-*consistent* relative to $\underline{D}$ if every FD in $F^+[\text{ATTR}_i]$ is true in $D(\underline{R}_i)$.

2.4 Representation

A special case of schema equivalence arises frequently in this paper. Let $\underline{U} = <\text{ATTR},F>$ be a (universal) relation schema, and let $\underline{D} = \{\underline{R}_i = <\text{ATTR}_i,F_i> | i = 1,\dots,n\}$. Using the terminology of [BBG], we say $\underline{D}$ *Rep2-represents* $\underline{U}$ if $(\cup_{i=1}^{n} \text{ATTR}_i) = \text{ATTR}$ and $(\cup_{i=1}^{n} F_i)^+ = F^+$. Note that $\underline{D}$ Rep2-represents $\underline{U}$ iff $\text{REP}(\underline{D}) = \text{REP}(\underline{U})$. Rep2 is essentially the notion of schema equivalence used in [Bern].

Given that $\underline{D}$ Rep2-represents $\underline{U}$ we define mappings between their consistent states:

$J:\text{DOM}(\underline{D}) \to \text{DOM}(\underline{U})$, where $J: D \mapsto U = *_{i=1}^{n} D(\underline{R}_i)$; and

$P:\text{DOM}(\underline{U}) \to \text{DOM}(\underline{D})$, where $P: U \mapsto D$, st. $D(\underline{R}_i) = U[\text{ATTR}_i]$ for $i = 1,\dots,n$.

The following facts are well known:

(i) J and P are total functions.

(ii) $J(P(U)) \supseteq U$.

(iii) $P(J(D)) = D$.

(iv) $U \subseteq U'$ implies $P(U) \leq P(U')$.

(v) $D \leq D'$ iff $J(D) \subseteq J(D')$.

A stronger form of representation is used by some authors [Cod1], [Ris]. $\underline{D}$ *Rep4-represents* $\underline{U}$ if $\underline{D}$ Rep2-represents $\underline{U}$ and for all $U \in \text{DOM}(\underline{U})$, $J(P(U)) = U$. Rep4-representation implies that $\text{DOM}(\underline{D})$ and $\text{DOM}(\underline{U})$ are isomorphic, with J and P the respective isomorphisms. Hence every consistent state of $\underline{U}$ can be exactly reconstructed from the corresponding state of $\underline{D}$.

2.5 Boyce Codd Normal Form

Let $\underline{R} = <\text{ATTR},F>$, and let $X \subseteq \text{ATTR}$. X is called a *superkey* of $\underline{R}$ if $X \to \text{ATTR} \in F^+$. X is a *key* of $\underline{R}$ if X is a superkey but does not properly contain a superkey. An FD $X \to X'$ is called *trivial* if $X' \subseteq X$, because it holds in every state of a relation independent of F.

$\underline{R}$ is in *Boyce Codd Normal Form* (abbr. BCNF) if for all non-trivial FDs $X \to Y$ in F^+, X is a superkey. In extending this notion to database schemas, we must be conscious of the UR-assumption. We say that $\underline{R}_i = \langle ATTR_i, F_i \rangle$ is in BCNF if the schema $\langle ATTR_i, F^+[ATTR_i] \rangle$ is in BCNF, and $\underline{D}$ is in BCNF if each $\underline{R}_i$ is.

3. Goals of Boyce Codd Normal Form

Third normal form and later Boyce Codd normal form were introduced to eliminate anomalous update behavior. Let us quote from [Cod1] the motivation for these normal forms.

> Looking at a sample instantaneous tabulation of $\underline{R}$ (Figure 2.2) the undesirable properties of the $\underline{R}$ schema become immediately apparent. We observe, for example, that, if the manager of department y should change, more than one tuple has to be updated. The actual number of tuples to be updated can, and usually will, change with time. A similar remark applies if department x is switched from government work (contract type g) to non-government work (contract type n).
>
> Deletion of the tuple for an employee has two possible consequences: deletion of the corresponding department information if his tuple is the sole one remaining just prior to deletion, and non-deletion of the department information otherwise.

Inserting an employee tuple has the complementary problem: if the employee is the first member of a department, we must simultaneously insert new M# and CT information for that department; inserting the second (third, etc.) employee into that department has no such requirement.

Abstracting from the above discussion, we find $\underline{R}$ to be an undesirable schema because the precise effects of a given update operation cannot be predicted simply by examining the schema. One must examine the state of $\underline{R}$ to see the effects that a given update produces. When all of the effects of an update can be determined by examining the schema alone, we say that the update is *syntactically predictable* (more precise definitions will appear later).

In state R for example, replacing department y's manager by another value implies updating a (syntactically) unpredictable number of tuples. Inserting an employee into (a new) department v requires creating new department-related data, while inserting an employee into (an existing) department x does not have this effect.

In this section we formalize this motivation, and prove that BCNF *relation schemas* attain it. In Sections 4 and 5 we will consider multi-relation database schemas.

3.1 Update Operations

Since the goal of normalization is to improve update behavior, our first step is to define update operations.

Let $\underline{R} = \langle ATTR, F \rangle$ be a schema. The update operations we define on $\underline{R}$ are + (insertion), - (deletion), and & (replacement).

If $R \in DOM(\underline{R})$, then

$$+(R,r) = \begin{cases} R \cup \{r\}, & \text{if consistent} \\ R, & \text{otherwise} \end{cases}$$

$$-(R,r) = R - \{r\}$$

$$\&(R,r,r') = \begin{cases} (R-\{r\}) \cup \{r'\}, & \text{if consistent} \\ R, & \text{otherwise.} \end{cases}$$

3.2 Insertion and Deletion Anomalies

Intuitively, $\underline{R}$ exhibits an *insertion anomaly* if for some states R and tuples r, the operation +(R,r) affects one collection of FDs, while for other states and tuples the insertion affects a different collection of FDs. The goal of normalization (with respect to insertions) is to eliminate such anomalies.

In formalizing this notion a technical problem surfaces. If $R = \emptyset$, the insertion of any tuple must affect *every* FD, while if $R \neq \emptyset$, the insertion of any tuple already in R does not affect *any* FD. Consequently, every schema exhibits insertion anomalies in this trivial sense (unless $ATTR = \emptyset$). A similar problem arises with deletions: if $R = \{r\}$, the deletion of r must affect every FD, while for any R, the deletion of a tuple not in R cannot affect any FD.

A second problem arises with respect to *trivial* FDs. Since every attribute functionally determines itself, the "collection of FDs" affected by an update will in general vary by these trivial FDs at least. We therefore choose to discard trivial FDs in this context.

Formally, we say that +(R,r) *affects* $f: X \to Y$ if $R[X \cup Y] \neq (+(R,r))[X \cup Y]$, and we define

$Affect(+\underline{R}) = \{f \in F^+ \mid f$ is non-trivial, and for some $R \neq \emptyset$, and some r, +(R,r) affects f$\}$

$NoAffect(+\underline{R}) = \{f \in F^+ \mid f$ is non-trivial, and for some $R \neq \emptyset$, and some r, $+(R,r) \neq R$, yet +(R,r) does not affect f$\}$

$\underline{R}$ is free of *insertion anomalies* iff $Affect(+\underline{R}) \cap NoAffect(+\underline{R}) = \emptyset$. The definition of *deletion anomalies* is similar and appears in Figure 3.1.

$\text{Affect}(-\underline{R}) = \{f \in F^+ | f$ is non-trivial, and for some R and r, $R \neq \{r\}$, yet $-(R,r)$ affects $f\}$.

$\text{NoAffect}(-\underline{R}) = \{f \in F^+ | f$ is non-trivial, and for some R and r, $R \neq \{r\}$ and $-(R,r) \neq R$, yet $-(R,r)$ does not affect $f\}$.

$\underline{R}$ is free of *deletion anomalies* iff $\text{Affect}(-\underline{R}) \cap \text{NoAffect}(-\underline{R}) = \emptyset$.

Figure 3.1. Definition of Deletion Anomaly.

It is conjectured in [Cod2] that BCNF schemas are the only ones that can avoid insertion and deletion anomalies. Now that we have formalized these anomalies, we can prove this conjecture true.

Theorem 1. (i) $\underline{R}$ *is free of insertion anomalies iff* $\underline{R}$ *is in* BCNF.

(ii) $\underline{R}$ *is free of deletion anomalies iff* $\underline{R}$ *is in* BCNF.

Proof. (i) $\text{Affect}(+\underline{R}) = \{$non-trivial FDs in $F^+\}$ whether or not $\underline{R}$ is in BCNF. So it suffices to prove that $\text{NoAffect}(+\underline{R}) = \emptyset$ iff $\underline{R}$ is in BCNF.

if: Let $f:X \to Y$ be any non-trivial FD in F^+. We will construct an insertion that affects f. By definition of BCNF, X is a superkey. Hence, all tuples in any consistent state of $\underline{R}$ have different X-projections. Let R' be any consistent state containing two or more tuples; such a state must exist for any $\underline{R}$, e.g., the state $\{<1,\ldots,1>,<2,\ldots,2>\}$ is such a state. Let $r \in R'$, and let $R = R' - \{r\}$. $R \neq \emptyset$, $+(R,r) = R' \neq R$, yet $+(R,r)$ affects f. Consequently $\text{NoAffect}(+\underline{R}) = \emptyset$ as desired.

only if: We prove that if $\underline{R}$ is not in BCNF, then $\text{NoAffect}(+\underline{R}_i) \neq \emptyset$. If $\underline{R}$ is not in BCNF, then there exists a non-trivial FD $f:X \to Y$ in F^+ for which X is not a superkey. We will construct an insertion that does not affect f.

Let R' be any consistent state containing two (or more) tuples r and r' such that $r[X] = r'[X]$. Such a state must exist by the completeness of F^+: since $X \to \text{ATTR} \notin F^+$, there must exist a state R' in which F^+ holds, but $X \to \text{ATTR}$ does not hold [Theorem 3, BFH]; for $X \to \text{ATTR}$ not to hold in R', distinct tuples r and r' must be present in R' with $r[X] = r'[X]$.

Let $R = R' - \{r\}$. $R \neq \emptyset$, $+(R,r) = R' \neq R$, yet $+(R,r)$ does not affect f. Hnece, $\text{NoAffect}(+\underline{R}) \neq \emptyset$, as desired.

(ii) The result for deletions follows by similar proof. □

3.3 Replacement Anomalies

Replacement anomalies are concerned with non-predictability of a different nature. The replacement depicted in the Introduction to Section 3 sought to change the manager of department y, leaving all other values unchanged. This replacement was judged to be anomalous because an unpredictable *number* of tuples might have to be replaced to achieve this effect. If we scrutinize this motivation, however, difficulties emerge.

The operation used in the example is *not* a single-tuple replacement of the type defined in Section 3.1. Instead, it attempts to modify a set of tuples that satisfy a boolean qualification, "D# = y" in this case. Arbitrarily many tuples may satisfy an arbitrary qualification. So, nonpredictability of this sort can hardly be called "anomalous". Apparently, Codd did not have such general operations in mind.

The operation that seems to be intended is the following. Let $\underline{R} = <\text{ATTR},F>$, let $R \in \text{DOM}(\underline{R})$, and let $f: X \to Y$ be a non-trivial FD in F^+. For any value x, and for each r in $\{r \in R | r[X] = x\}$ the user may replace $r[Y]$ with a new value. The replacement has *predictable size* if $|\{r \in R | r[X] = x\}| \leq 1$. $\underline{R}$ is free of *replacement anomalies* if for all non-trivial $f: X \to Y$ in F^+, all $R \in \text{DOM}(\underline{R})$, and all x, the replacement has predictable size.

Theorem 2. $\underline{R}$ *is free of replacement anomalies iff* $\underline{R}$ *is in* BCNF.

Proof. All replacements have predictable size iff for all non-trivial $f: X \to Y$ in F^+, X is a superkey. □

It is not apparent to us that multi-tuple replacements of this type warrant special consideration in the schema design process. Replacement anomalies will not be studied further in this paper.

4. Update Operations Under UR-Assumption

In the following two sections we attempt to extend Theorem 1 to multi-relation database schemas. This attempt will *fail*, because of the UR-assumption. Instead, we shall prove that BCNF database schemas are not free of insertion and deletion anomalies except in trivial cases.

4.1 Problems in Preserving UR-Consistency

Recall the example of Section 3. $\underline{R}$ is not in BCNF and therefore exhibits update anomalies; the remedy recommended in [Cod1] is to decompose $\underline{R}$ into two relation schemas, $\underline{R}_1$ and $\underline{R}_2$, which are in BCNF (see Fig. 4.1). Notice that the database schema $\underline{D} = \{\underline{R}_1, \underline{R}_2\}$ Rep4-represents $\underline{R}$; therefore $\underline{D}$ and $\underline{R}$ have equivalent representational power, provided the database system only permits UR-consistent states of $\underline{D}$ to occur.

This leaves us the problem of designing update operations that preserve UR-consistency. For example, to insert <v, 14, n> into $\underline{R}_2$, a tuple r_1 with $r_1[D\#] = v$ must be inserted into R_1. Similarly, to delete <x, 11, g> from $\underline{R}_2$, all tuples r_1 in R_1 with $r_1[D\#] = x$ must be deleted too. In these two cases, the requirement of UR-consistency dictates a "natural" semantics for the update. However, in other cases, the choice of a reasonable semantics is less constrained.

For example, suppose we want to insert <9, d, v> into $\underline{R}_1$. One way to preserve UR-consistency is to insert <9, d, v> into R_1 and <v, null, null> into R_2, where "null" denotes a blank or uncommitted value that is distinguishable from "real" values, such as 11. To adopt this interpretation, we must develop a complete semantics for update operations when null values are present. We attempted such a development (see Appendix I) but discovered that for each semantics that we tried, certain bizarre behavior was forced by the UR-assumption. For now, we are defeated in taking this route.

The choices that remain are to use D# values already in R_2 or to invent new ones. Consider the first choice. To insert <9, d, v> into $\underline{R}_1$, we can preserve UR-consistency by also replacing <x, 11, g> by <v, 11, g> in R_2, and replacing each instance of 'x' in R_1 by 'v'. There are several problems with this semantics. First, the choice of <x, 11, g> as the victim to be replaced was arbitrary; <y, 12, n> or <z, 13, n> would have worked as well. Second, the interpretation fails when R_2 is empty, because there are no tuples in R_2 to replace. Third, although we have not inserted new D# *values*, we *have* produced new *relationships* both in R_1 and R_2 that were not part of the insertion request. Finally, this approach effectively asserts that the tuple being inserted is "more reliable" than data already in the database, since we have chosen to modify existing relationships instead of rejecting the insertion as an integrity violation. For these reasons we have rejected this "replacement" approach to insertion semantics.

The only remaining choice is to insert a "real" valued tuple into R_2, such as <v, 14, g>. This approach, like the preceding one, arbitrarily selects values to be placed into the database, and produces a relationship that was not part of the insertion request. However, it has desirable properties not shared by the replacement interpretation. First, it succeeds even when R_2 is empty. Second, the effect on R_1 of "insert <9, d, v> into $\underline{R}_1$" is to insert that tuple into that relation, and to make no other changes to R_1. Third, an insertion never has the effect of deleting a value or relationship; this is consonant with the intuitive understanding that an insertion creates data and does not destroy it. For these reasons, we have selected this interpretation of insertion in the UR-environment.

In Section 4.2 we formalize this semantics and generalize it to deletions and replacements. Basic properties of these operations are investigated in Appendix II.

4.2 Update Operations that Preserve UR-Consistency

Let $\underline{D} = \{\underline{R}_1, \ldots, \underline{R}_n\}$ be a database schema. The update operations defined on $\underline{D}$ are named $+\underline{R}_i$, $-\underline{R}_i$, and $\&\underline{R}_i$ for $i = 1, \ldots, n$. (We include the schema name, $\underline{R}_i$, as part of the operation name for notational clarity.) If $D \in \mathrm{DOM}(\underline{D})$, then

$+\underline{R}_i(D, r_i)$ denotes insertion of r_i into $D(\underline{R}_i)$,

$-\underline{R}_i(D, r_i)$ denotes deletion of r_i from $D(\underline{R}_i)$,

$\&\underline{R}_i(D, r_i, r_i')$ denotes replacement of r_i by r_i' in $D(\underline{R}_i)$.

As indicated in Section 4.1, each operation may have to update other relations to preserve UR-consistency; to formalize the semantics of these operations we must also specify these other effects. As we have illustrated there is no unique, "natural" semantics. Instead we choose to base the semantics of these operations on properties that we judge to be desirable for update operations in general. These properties are closely related to the correctness criteria for "view updates" postulated in [DB]. We state the properties for insertions; the definitions for deletions and replacements are analogous.

Property 1--Consistency

The result of an update operation must be a consistent state.

Property 2--Exact Performance

Given $+\underline{R}_i(D, r_i)$, let $R_i' = +(D(\underline{R}_i), r_i)$. State D' *exactly performs* $+\underline{R}_i(D, r_i)$ if $D'(\underline{R}_i) = R_i'$.

Property 3--Intent Performance

State D' *performs the intent* of $+\underline{R}_i(D, r_i)$ if D' can be obtained from D by applying insertion operations to individual relations. (This property is stated more precisely in Appendix II.)

If D' satisfies Properties 1-3 for an update operation, we say that D' *performs* that update. In general, many states may perform an update. Our final property provides a metric choosing certain of these states over others.

Property 4--Nonextraneous Performance

Suppose D' performs $+\underline{R}_i(D, r_i)$ and let Ω_+ be a sequence of insertions that maps D into D'. Since every operation in Ω_+ (except for the one that inserts r_i into R_i) causes a change to the database not specified by the user, all unnecessary operations should be eliminated. D' *nonextraneously performs* $+\underline{R}_i(D, r_i)$ if (a) D' performs the insertion, and (b) for all subsets* Ω_+' of Ω_+, Ω_+' either maps D into D', or it maps D

*Technically, we mean "projection" here. In Appendix II, however, we demonstrate that Ω_+ can be treated as a set with no loss of generality; with this understanding, the word "subset" is correct.

into a state that does not perform the insertion. (A more precise statement is given in Appendix II.)

The semantics of $+\underline{R}_i$, $-\underline{R}_i$, and $I\underline{R}_i$ can now be defined. Let $D \in \mathrm{DOM}(\underline{D})$. $+\underline{R}_i(D,r_i) =$ *any* state that nonextraneously performs $+\underline{R}_i(D,r_i)$. $-\underline{R}_i$ and $\&\underline{R}_i$ have analogous definitions. In general, $+\underline{R}_i(D,r_i)$ and $\&\underline{R}_i(D,r_i,r_i')$ have many possible meanings and we treat these operations as non-deterministic functions. By contrast, $-\underline{R}_i(D,r_i)$ is uniquely specified by this definition, a fact we prove in Appendix II.

Database Schema:

$$\underline{D} = \{\underline{R}_1 = \langle\{E\#,JC,D\#\},\{E\# \rightarrow JC,D\#\}\rangle,$$
$$\underline{R}_2 = \langle\{D\#,M\#,CT\},\{D\# \rightarrow M\#,CT; M\# \rightarrow D\#,CT\}\rangle\}$$

Database:

$R_1 =$

E#	JC	D#
1	a	x
2	c	x
3	a	y
4	b	x
5	b	y
6	c	y
7	a	z
8	c	z

, $R_2 =$

D#	M#	CT
x	11	g
y	12	n
z	13	n

$\underline{R}_1$ and $\underline{R}_2$ are in BCNF and $\underline{D}$ Rep4-represents $\underline{R}$ of Fig. 2.1.

Figure 4.1. A Normalized Database Schema.

5. BCNF Does Very Little

Having developed an update semantics that preserves UR-consistency, we are ready to study normalization in this context. There are two issues: (1) characterizing database schemas that avoid insertion and deletion anomalies; and (2) the schema design question--characterizing relation schemas $\underline{U}$ for which there exists a database schema $\underline{D}$ that represents $\underline{U}$, yet avoids insertion and deletion anomalies.

5.1 Database Schemas that Avoid Anomalies

Throughout this section let $\underline{D} = \{\underline{R}_1,\ldots,\underline{R}_n\}$, and $\underline{U} = \langle ATTR,F\rangle$, such that $\underline{D}$ Rep2-represents $\underline{U}$. If $\underline{D}$ is to avoid update anomalies it is necessary for $\underline{D}$ to be in BCNF: otherwise the "internal effects" of updates would exhibit anomalies as proved in Section 3. However, we shall prove that BCNF is not sufficient to prevent anomalies caused by the "external effects" of these operations.

The definitions of insertion and deletion anomalies require technical changes in the UR-environment. First, the definitions of Affect and NoAffect given in Section 3 quantify over all *consistent* states of a schema, but it is technically better to quantify over *reachable* states only; in light of Theorem II.1 (see Apendix II), though, this distinction has no impact. Second, since we may assume that $\underline{D}$ is in BCNF, we may interpret Affect and NoAffect as sets of *relation schemas* instead of FDs: e.g., for any $\underline{R}_i$, $\underline{R}_j \in \underline{D}$, $\underline{R}_j \in \mathrm{Affect}(+\underline{R}_i)$ iff every non-trivial FD in $F^+[ATTR_j] \in \mathrm{Affect}(+\underline{R}_i)$, because all external effects of $+\underline{R}_i$ are themselves insertions. Figure 5.1 presents the definitions of Affect, NoAffect, and anomalies as used in this section.

$\mathrm{Affect}(+\underline{R}_i) = \{\underline{R}_j \in \underline{D}|$ for some D other than the empty state, for some tuple r_i, and for some meaning, $+\underline{R}_i(D,r_i)$ affects $\underline{R}_j\}$

$\mathrm{NoAffect}(+\underline{R}_i) = \{\underline{R}_j \in \underline{D}|$ for some D other than the empty state, for some tuple r_i, and for some meaning, $+\underline{R}_i(D,r_i) \neq D$, yet $+\underline{R}_i(D,r_i)$ does not affect $\underline{R}_j\}$

$\mathrm{Affect}(-\underline{R}_i) = \{\underline{R}_j \in \underline{D}|$ for some D and r_i, $D(\underline{R}_i) \neq \{r_i\}$, yet $-\underline{R}_i(D,r_i)$ affects $\underline{R}_j\}$

$\mathrm{NoAffect}(-\underline{R}_i) = \{\underline{R}_j \in \underline{D}|$ for some D and r_i, $D(\underline{R}_i) \neq \{r_i\}$, and $-\underline{R}_i(D,r_i) \neq D$, yet $-\underline{R}_i(D,r_i)$ does not affect $\underline{R}_j\}$.

$\underline{D}$ is free of *insertion anomalies* iff for all $\underline{R}_i \in \underline{D}$, $\mathrm{Affect}(+\underline{R}_i) \cap \mathrm{NoAffect}(+\underline{R}_i) = \emptyset$

$\underline{D}$ is free of *deletion anomalies* iff for all $\underline{R}_i \in \underline{D}$, $\mathrm{Affect}(-\underline{R}_i) \cap \mathrm{NoAffect}(-\underline{R}_i) = \emptyset$

Figure 5.1. Definitions of Anomalies for UR Environment.

A graphic representation of $\underline{D}$ helps characterize the effects of insertions. Let $G(\underline{D})$ be an undirected graph whose *vertex set* equals $\underline{D}$, and whose *edge set* contains $(\underline{R}_i, \underline{R}_j)$ iff $ATTR_i \cap ATTR_j \neq \emptyset$.

Lemma 1. (i) $\mathrm{Affect}(+\underline{R}_i) = \{\underline{R}_j \in \underline{D}|\underline{R}_i$ *and* $\underline{R}_j$ *are connected by a path in* $G(\underline{D})\}$.

(ii) $\mathrm{NoAffect}(+\underline{R}_i) = \{\underline{R}_j \in \underline{D}|ATTR_j \rightarrow ATTR_i \in F^+\}$.

Proof. See Appendix II.4. □

Lemma 2. (i) $\mathrm{Affect}(-\underline{R}_i) = \mathrm{Affect}(+\underline{R}_i)$

(ii) $\mathrm{NoAffect}(-\underline{R}_i) = \mathrm{NoAffect}(+\underline{R}_i)$.

Proof. See Appendix II.4. □

Combining these results we obtain a precise characterization of database schemas that are free of insertion and deletion anomalies under the UR-assumption.

Theorem 3. $\underline{D}$ *is free of insertion anomalies (or equivalently deletion anomalies) iff* $\underline{D}$ *is in* BCNF, *and for all* $\underline{R}_i$, $\underline{R}_j$, *if* $ATTR_i \cap ATTR_j \neq \emptyset$ *then* $ATTR_i \rightarrow ATTR_j \in F^+$ *and* $ATTR_j \rightarrow ATTR_i \in F^+$.

The conditions placed on $\underline{D}$ by this result are quite restrictive; for example, the normalized schema suggested in Figure 4.1 does not satisfy them, nor for that matter do most of the normalized schemas commonly illustrated in the literature. This bodes poorly for the schema design aspect of normalization.

5.2 Attainability of Normalized Schemas

The remaining issue is one of schema design. Let $\underline{U} = \langle ATTR, F\rangle$ be any relation schema; we prove that $\underline{U}$ can be normalized into a schema $\underline{D}$ that is free of insertion and deletion anomalies iff $\underline{U}$ is "almost" in BCNF itself.

A database schema $\underline{D}_p = \{\underline{D}_{pi} = \langle ATTR_{pi}, F_{pi}\rangle \mid i=1,\ldots,n\}$ is said to *partition* $\underline{U}$ if $\underline{D}_p$ Rep2-represents $\underline{U}$ and $ATTR_{pi} \cap ATTR_{pj} = \emptyset$ for $i \neq j$.

Theorem 4. *There exists a database schema* $\underline{D}$ *that* Rep2-represents $\underline{U}$ *yet is free of insertion and deletion anomalies iff there exists a* BCNF *database schema* $\underline{D}_p$ *that partitions* $\underline{U}$.

Proof. if: If $\underline{D}_p$ exists, it Rep2-represents $\underline{U}$ and satisfies the conditions of Theorem 3.

only if: Let $P_1,\ldots,P_n$ be the connected components of $G(\underline{D})$, and let $\{\underline{R}_{k1},\ldots,\underline{R}_{km_k}\}$ be the schemas in component P_k. The schemas in each component are "equivalent" in the sense that $\forall \underline{R}_{ki}, \underline{R}_{kj}$ in P_k, $ATTR_{ki} \rightarrow ATTR_{kj} \in F^+$ and $ATTR_{kj} \rightarrow ATTR_{ki} \in F^+$. Also each schema is in BCNF. Therefore, if we merge all schemas in P_k into one schema $\underline{R}_{pk} = \langle \cup_{i=1}^{m_k} ATTR_{ki}, \cup_{i=1}^{m_k} F_{ki}\rangle$, $\underline{R}_{pk}$ is in BCNF also. The database schema $\underline{D}_p = \{\underline{R}_{pk} \mid k=1,\ldots,m\}$ is thus a BCNF schema which partitions $\underline{U}$. □

Figure 5.2 illustrates two typical relation schemas that are BCNF-partitionable, and database schemas that Rep2-represent them while satisfying the conditions of Theorem 3. In both cases the database schemas do not Rep4-represent the original schema.

If Rep4-representation is required, $\underline{U}$ can be "normalized" if and only if it already is in BCNF!

Corollary to Theorem 4. *There exists a database schema* $\underline{D}$ *that* Rep4-represents $\underline{U}$ *yet is free of insertion and deletion anomalies iff* $\underline{U}$ *is in* BCNF.

(a) $\underline{U} = \langle\{E\#, D\#, M\#, PROJ\}, \{E\# \rightarrow D\#; M\# \rightarrow PROJ\}\rangle$

$\underline{D} = \{\underline{R}_1 = \langle\{E\#, D\#\}, \{E\# \rightarrow D\#\}\rangle,$

$\underline{R}_2 = \langle\{M\#, PROJ\}, \{M\# \rightarrow PROJ\}\rangle\}$

(b) $\underline{U} = \langle\{SUPPLIER, PART, CITY\}, \{SUPPLIER \rightarrow CITY\}\rangle$

$\underline{D} = \{\underline{R}_1 = \langle\{SUPPLIER, CITY\}, \{SUPPLIER \rightarrow CITY\}\rangle$

$\underline{R}_2 = \langle\{PART\}, \{\ \}\rangle\}$

Figure 5.2. Schemas That Can Be Normalized.

Proof. If $\underline{U}$ is in BCNF there is nothing to prove. To prove the converse, observe that if $\underline{D}$ Rep4-represents $\underline{U}$, the J operator must denote a lossless join [ABU], hence $G(\underline{D})$ must be a connected graph. If we merge all relation schemas as in the proof of Theorem 4, the result is a BCNF relation schema $\underline{R} = \langle ATTR, F'\rangle$ where $F'^+ = F^+$. Hence $\underline{U} = \langle ATTR, F\rangle$ is in BCNF as well. □

6. Conclusion

BCNF was invented to prevent anomalous side effects of relational updates. The question we have asked is, "does BCNF attain this goal?" We have given two conflicting answers: (1) In the context of a single relation schema, BCNF is successful. But (2) in a multirelation database schema, BCNF fails.

In spite of (2), we believe BCNF to be an important schema design goal.

The failure of BCNF in the multirelation context is caused by the universal relation assumption. The UR-assumption is adopted in normalization theory to formalize notions of FD completeness and uniqueness, and schema equivalence. Recent work on normal forms, e.g., fourth and fifth normal forms [Fag1,2], leans even more heavily upon the UR-assumption; these normal forms cannot even be defined without the UR-assumption. Recent work on schema equivalence [BMSU] depends strongly on this assumption as well.

Yet update operations are clumsy to define when the UR-assumption is present, and behave quite badly. In addition to the problems noted in this paper, the UR-assumption introduces complexity problems as well, since testing whether an insertion preserves the UR-assumption is NP-complete [HLY]. Apparently, the UR-assumption is incompatible with databases that are updated.

At present, normalization theory is an isolated theoretical area divorced from database practice. This separation will persist until normalization theory is made adequate to prove the benefits of normal forms.

APPENDIX I

Pitfalls in Null-Value Update Semantics

Let us return to the example database of Section 4.1. The schema is

$$\underline{D} = \{\underline{R}_1 = \langle\{E\#,JC,D\#\},\{E\# \rightarrow JC,D\#\}\rangle,$$
$$\underline{R}_2 = \langle\{D\#,M\#,CT\},\{D\# \rightarrow M\#,CT;M\# \rightarrow D\#,CT\}\rangle\}$$

and the database state is

$D:R_1 =$

E#	JC	D#
1	a	x
2	c	x
3	a	y
4	b	x
5	b	y
6	c	y
7	a	z
8	c	z

, $R_2 =$

D#	M#	CT
x	11	g
y	12	n
z	13	n

.

If the user says to insert <9,d,v> into $\underline{R}_1$, the most plausible way to preserve UR-consistency is to simultaneously insert <v,null,null> into R_2. All other interpretations, we saw in 4.1, share the dubious property of inventing data values and/or relationships among data values in the database.

In this appendix we explore the consequences of using null values to preserve UR-consistency.

I.1 The Second-Insertion Problem

The database state that results from the preceding insertion is

$D^1:R_1^1 =$

E#	JC	D#
1	a	x
8	c	z
9	d	v

, $R_2^1 =$

D#	M#	CT
x	11	g
y	12	n
z	13	n
v	null	null

.

Suppose the user now says to insert <v,14,g> into R_2. We cannot simply add <v,14,g> to R_2^1, because the resulting relation state would be inconsistent: the FD D#→M#, CT would be violated. To overcome this inconsistency, the natural interpretation is to replace <v,null,null> by <v,14,g>. The rationale for this interpretation is that <v,14,g> signifies "more complete" information than <v,null,null>, hence this new tuple makes the old old one obsolete.

Applying this rationale to another case, consider the insertion of the same tuple in the state

$D^2:R_1^2 =$

E3	JC	D#
1	a	x
9	d	null

, $R_2^2 =$

D#	M#	CT
x	11	g
null	null	g

.

since <v,14,g> is "more complete" than <null, null, g> the correct action apparently is to replace <null,null,g> by <v,14,g>. But then UR-consistency is violated, since <9,d,null> in R_1^2 no longer "matches up" with any tuple in R_2^2, and further correction is needed. The correction would seem to be the replacement of <9,d,null> by <9,d,v>. The resulting state is

$D^3:R_1^3 =$

E#	JC	D#
1	a	x
9	d	v

, $R_2^3 =$

D#	M#	CT
x	11	g
v	14	g

.

D^3 asserts that employee 9 works in department 'v', but this relationship is in no sense implied by the user level insertion operation. The database system has *invented* a relationship between employee 9 and department 'v', which is precisely the problem we thought null values would prevent!

This problem is not caused by the use of null values per se; it is caused by the interpretation of insertion operations as replacements when null values are present. The alternative, of course, is to treat insertions as insertions whether or not nulls are present. However, this interpretation begs the issue, because it assigns the same semantics to null values as real ones.

I.2 What Do Deletions Mean?

Let us return to state D and consider possible meanings of the delete operation. Suppose the user says to delete <z,13,n> from $\underline{R}_2$. Two plausible meanings for this are

1. replace <z,13,n> by <z,null,null>;
2. delete <z,13,n> from R_2 and delete employee in department 'z' from R_1.

Meaning (1) is the inverse of the insertion semantics suggested in I.1. The rationale for this interpretation is that deletions are too drastic for the system to undertake automatically--the database system should never delete more data than the user specifies. Meaning (1) supports this reasoning while meaning (2) deletes far more data than the user specified.

But consider what happens if we add relation schemas

$$\underline{R}_3 = \langle\{CT,VP\},\{CT \rightarrow VP\}\rangle$$

and

$$\underline{R}_4 = \langle\{PROJ,M\#\},\{PROJ \rightarrow M\#\}\rangle$$

to the database with states

$R_3 =$

CT	VP
g	MONDALE
n	FORD

, and $R_4 =$

PROJ	M#
p1	11
p2	12
p3	13

.

If we adopt meaning (1) now, the deletion of <z,13,n> has no effect whatever! Given this semantics, the following procedure is required for a user to actually remove <z,13,n> from the database:

1. delete <p3,13> from $\underline{R}_4$, and delete every employee from $\underline{R}_1$ with D# = 'z'. The resulting state is

$D^1: R_1^1 =$

E#	JC	D#
1	a	x
6	c	y
null	null	z

, $R_2^1 =$

D#	M#	CT
x	11	g
y	12	n
z	13	n

,

$R_3^1 =$

CT	VP
g	MONDALE
n	FORD

, $R_4^1 =$

PROJ	M#
p1	11
p2	12
null	13

.

2. delete <z,13,n> from $\underline{R}_2$; to implement this deletion the database system must delete <null,null,z> from R_1^1, <z,13,n> from R_2^1, and <null,13> from R_4^1. While in a strict sense this implementation deletes more data than the user specified, we judge these extra deletions to be acceptable since all unspecified data values are nulls.

As an exercise the reader may devise a similar procedure for the following case:

$\underline{D} = \{\underline{R}_1 = <\{A,B\},\{A\rightarrow B\}>,\ \underline{R}_2 = <\{B,C\},\{B\rightarrow C\}>,$
$\underline{R}_3 = <\{C,A\},\{C\rightarrow A\}>$

D: $R_1 =$

A	B
a_1	b_1
a_2	b_2

, $R_2 =$

B	C
b_1	c_1
b_2	c_2

, $R_3 =$

C	A
c_1	a_1
c_2	a_2

operation: delete $<a_1,b_1>$ from $\underline{R}_1$.

I.3 Another Interpretation of Delete

Let us return to the original schema $\underline{D}$ and its original state

D: $R_1 =$

E#	JC	D#
1	a	x
7	a	z
8	c	z

, $R_2 =$

D#	M#	CT
x	11	g
y	12	n
z	13	n

.

Another plausible meaning of "delete <z,13,n> from $\underline{R}_2$" is

Replace <z,13,n> by <null,null,null> in R_2, and replace the department of every department 'z' employee by 'null' in R_1.

The resulting state is

$D^1: R_1^1 =$

E#	JC	D#
1	a	x
7	a	null
8	c	null

, $R_2^1 =$

D#	M#	CT
x	11	g
y	12	n
null	null	null

.

This interpretation eliminates the problems noted in I.2, but its interaction with the insertion semantics of I.1 is strange. Suppose the user says to insert <v,14,g> into $\underline{R}_2$. Since <v,14,g> is "more complete" than <null,null,null>, the effect (according to I.1), is to replace <null,null,null> by <v,14,g>. This replacement violates UR-consistency, however, and the corrective action is to replace every 'null' department in R_1^1 by 'v'. This yields

$D^2: R_1^2 =$

E#	JC	D#
1	a	x
7	a	v
8	c	v

, $R_2^2 =$

D#	M#	CT
x	11	g
y	12	n
v	14	g

Observe that the deletion of <z,13,n> and the insertion of <v,14,g> have been *coupled* into a replacement, whether or not this was intended!

I.4 Minimizing the Use of Null Values--A Problem

Additional difficulties surround the question of how many nulls should be inserted when correcting UR violations. Let us agument schema $\underline{D}$ with relation schema $\underline{R}_3 = <\{CT,VP\},\{CT\rightarrow VP\}>$. And consider the state

$D^1: R_1^1 =$

E#	JC	D#
1	a	x
9	d	v

, $R_2^1 =$

D#	M#	CT
x	11	g
y	12	n
z	13	n
v	null	null

,

$R_3^1 =$

CT	VP
g	MONDALE
n	FORD
null	null

.

If the user says to insert <10,d,u> into $\underline{R}_1$ either of the following interpretations is possible:

1. insert <10,d,u> into R_1^1 and <u,null',null> into R_2^1; or

2. insert <10,d,u> into R_1^1, <u,null',null'> into R_2^1, and <null',null'> into R_3^1.

The advantage of (1) is that the number of relations affected is minimized. Since null values have no intrinsic value this seems a worthwhile goal. The state corresponding to (1) is

$D^2: R_1^2 =$

E#	JC	D#
1	a	x
9	d	v
10	d	u

, $R_2^2 =$

D#	M#	CT
x	11	g
v	null	null
u	null'	null

,

$R_3^2 =$

CT	VP
g	MONDALE
n	FORD
null	null

.

Suppose the user subsequently determines that department 'v' has vice-president 'AGNEW'. To place this fact in the database, the usual procedure is

I. Find the R_3 tuple that describes department v's vice-president. I.e.,

(i) restrict R_2 by D# = 'v'

(ii) joint the result of (i) with R_3

(iii) project onto $ATTR_3 = \{CT, VP\}$.

II. Replace the VP-projection of all tuples found in (I) by 'AGNEW'.

Applying this procedure to D^2, we obtain

$D^3: R_1^3 =$

E#	JC	D#
1	a	x
9	d	v
10	d	u

, R_2^3

D#	M#	CT
x	11	g
v	null	null
u	null'	null

,

$R_3^3 =$

CT	VP
g	MONDALE
n	FORD
null	AGNEW

.

This state asserts, *erroneously*, that AGNEW is vice-president of department 'u'.

To avoid this error, the retrieval step (part I) of the procedure must be cognizant of null join values and must *insert new null values* as it goes. It would certainly be simpler for the insert operator to insert these extra nulls in the first place.

I.5 Maximizing the Use of Null Values--A Problem

On the other hand there are cases where maximizing the use of null values is also wrong. Let us add the following relation schemas to D:

$R_4 = \langle\{A\#, D\#\}, \{A\# \rightarrow D\#\}\rangle$,

and

$R_5 = \langle\{E\#, A\#\}, \{\}\rangle$,

where A# denotes an account (in the financial sense), R_4 tells which accounts are used by each department, and R_5 identifies which employees may post expenses against which accounts. The FD A# → D# is present in R_4 because the "chart of accounts" normally mirrors the organizational hierarchy; R_5 has no FDs because each employee may generally charge to multiple accounts and vice versa.

Consider the state

$D^1: R_1^1 =$

E#	JC	D#
1	a	x
8	c	z

$, \ldots, R_4^1 =$

A#	D#
a1	x
a2	x
a3	y
a4	z
a5	z

$, R_5^1 =$

E#	A#
1	a1
2	a1
2	a2
3	a3
8	a5

and suppose two new employees are hired into department 'z' with job-codes 'd'. If we adopt the policy suggested in I.4 of inserting *unique* null values whenever a null value is needed, the resulting state is

$D^2: R_1^2 =$

E#	JC	D#
1	a	x
8	c	z
9	d	z
10	d	z

$, \ldots, R_4^2 =$

A#	D#
a1	x
a2	x
a3	y
a4	z
null	z
null'	z

$, R_5^2 =$

E#	A#
1	a1
8	a5
9	null
10	null'

The values of R_4^2 and R_5^2 are bizarre--it is highly unlikely that the hiring of every new employee would require the establishment of a new account. However, the alternative of letting all new hires share one null account number suffers the problems described in I.4 and is wrong, tto.

I.6 Conclusion

Null values are an escape clause from the UR-assumption; they attempt to circumvent UR-consistency in cases where it seems most unrealistic. Since the UR-assumption is central to currently developed database theories, it is not surprising that attempts to circumvent it are so troublesome.

APPENDIX II

Properties of Updates Under UR-Assumption

II.1 Update Maps

Properties 3 and 4 of Section 4.2 are defined in terms of sequences of updates operating on individual relations. This section formalizes this concept, refining it into the notion of "update map".

An *insertion command* is an ordered pair $\omega = \langle +R_j, r_j\rangle$ and is interpreted as a function on FD-consistent database states.

$\omega(D) = D'$ such that

$$D'(\underline{R}_i) = \begin{cases} D(\underline{R}_i) & , \text{ for } i \neq j \\ +(D(\underline{R}_j), r_j), & \text{for } i = j. \end{cases}$$

A sequence of insertion commands, $\Omega_+ = <\omega_1, \ldots, \omega_k>$ is interpreted as the composition of $\omega_1, \ldots, \omega_k$. Ω_+ is *minimal* in state D if for all projections Ω_+', $\Omega_+'(D) \neq \Omega_+(D)$. It is a fact that if Ω_+ is minimal in state D, every permutation of it is minimal too, and all map D into the same state. Conversely, if two sequences are both minimal and both map D into D', they are permutations of each other. Moreover, a minimal sequence of insertion commands cannot include duplicate elements. Therefore we can unambiguously consider such sequences to be *sets*. An *insertion map* is such a set. *Deletion maps* are defined analogously.

Replacement maps require different treatment because the order of replacements can affect the result even in a minimal sequence [BG]. Let Ω_- and Ω_+ be sets of deletion and insertion commands respectively, and let $\Omega_\& = <\Omega_-, \Omega_+>$. We interpret $\Omega_\&$ as the composition of Ω_- followed by Ω_+; $\Omega_\&$ is defined in state D iff Ω_- is a deletion map in state D and Ω_+ an insertion map in state $\Omega_-(D)$. We define a partial order over pairs of this form: $<\Omega_-', \Omega_+'> \leq <\Omega_-, \Omega_+>$ if both $\Omega_-' \subseteq \Omega_-$ and $\Omega_+' \subseteq \Omega_+$. $\Omega_\&$ is a *replacement map* in state D if (a) $\Omega_\&$ is defined in state D, and (b) for all $\Omega_\&' < \Omega_\&$ either $\Omega_\&'(D)$ is not defined or $\Omega_\&'(D) \neq \Omega_\&(D)$.

Properties 3 and 4 can now be restated.

<u>Property 3</u>--Intent Performance

State D' *performs the intent* of $+\underline{R}_i(D, r_i)$ if there exists an insertion map from D to D'. The definitions for $-\underline{R}_i$ and $\&\underline{R}_i$ are analogous.

<u>Property 4</u>--Nonextraneous Performance

D' *nonextraneoulsy performs* $+\underline{R}_i(D, r_i)$ if (a) D' performs $+\underline{R}_i(D, r_i)$, and (b) for all $\Omega_+' \subsetneq \Omega_+$, where Ω_+ is the insertion map from D to D', $\Omega_+'(D)$ does not perform $+\underline{R}_i(D, r_i)$. The definition for $-\underline{R}_i$ is analogous; for $\&\underline{R}_i$, change '$\subsetneq$' in part (b) to '$\lneq$'.

The following facts about update maps are used later. Let D and D' be FD-consistent states.

(i) there exists an insertion map from D to D' iff $D \leq D'$;

(ii) there exists a deletion map from D to D' iff $D \geq D'$; and

(iii) there exists a replacement map from D to D' always.

II.2 <u>Reachability</u>

A database state is *reachable* if it can be attained by applying a sequence of update operations to an initially empty database state. Reachability is complementary to consistency. By assumption, the consistent states of a schema are the states that represent meaningful configurations of information. We require that update operations preserve consistency to ensure that if state D occurs, D represents a meaningful situation. Conversely, it is desirable that if D represents a meaningful situation (i.e., is consistent), then D *can* occur. This property is called *reachability*. In a single relation using the update operations defined in 3.1, reachability is obviously achieved; in this section we prove that reachability is also achieved in a database using the operations defined in 4.2.

Let $D, D' \in \text{DOM}(\underline{D})$. D' is *reachable from* D if there exists a sequence of update operations (as defined in 4.2) that maps D into D'.

LEMMA II.1. *Let* $D \in \text{DOM}(\underline{D})$, $U \in P^{-1}(D)$, *and* $U' = +(U, u)$. *Then* $D' = P(U')$ *is reachable from* D.

<u>Proof</u>. Let Ω_+ be the insertion map from D to D'; $\Omega_+ = \{<+\underline{R}_j, r_j> | r_j = u[\text{ATTR}_j] \wedge r_j \notin R_j$, for $j = 1, \ldots, n\}$. Observe that Ω_+ contains at most one element per relation schema. We shall prove that whenever Ω_+ has this form and $\Omega_+(D)$ is consistent, $\Omega_+(D)$ is reachable from D. The proof is by induction on $|\Omega_+|$.

<u>Basis</u>: If $|\Omega_+| = \emptyset$, the result holds trivially.

<u>Induction</u>: Assume $\Omega_+(D)$ is reachable whenever Ω_+ contains at most one command per relation schema, $|\Omega_+| < N$, and $\Omega_+(D)$ is consistent; prove that $\Omega_+(D)$ remains reachable when $|\Omega_+| = N$.

Let $\omega_j = <+\underline{R}_j, r_j>$ be any element of Ω_+. We claim that $\Omega_+(D)$ performs $+\underline{R}_j(D, r_j)$: (1) $\Omega_+(D)$ is consistent; (2) $\Omega_+(D)$ exactly performs the insertion, because ω_j is the only element of Ω_+ that operates on $\underline{R}_j$; and (3) $\Omega_+(D)$ performs the intent by definition. Hence, there exists $\Omega_j \subseteq \Omega_+$ such that $D_j = \Omega_j(D)$ nonextraneously performs $+\underline{R}_j(D, r_j)$; moreover $\Omega_j \neq \emptyset$ since $\omega_j \in \Omega_j$. Let $\Omega_+' = \Omega_+ - \Omega_j$; $\Omega_+'(D_j) = \Omega_+(D) = D'$ and $|\Omega_+'| < N$, hence Ω_+' satisfies the induction hypothesis. Therefore $D' = \Omega_+'(D_j)$ is reachable from D_j by induction hypothesis, while D_j is reachable from D by construction. This proves that D' is reachable from D as desired. □

THEOREM II.1. *Let* $\underline{D}$ *be any database schema. Every* $D \in \text{DOM}(\underline{D})$ *is reachable from the empty state.*

<u>Proof</u>. Let D' be any consistent state of $\underline{D}$ and let $U' \in P^{-1}(D')$. We shall prove that D' is

reachable from the empty state, by induction on $|U'|$.

<u>Basis</u>: If $|U'| = \emptyset$, there is nothing to prove.

<u>Induction</u>: Assume D' is reachable from the empty state when $|U'| < N$; prove that D' is reachable when $|U'| = N$.

Let $U = U' - \{u\}$ for any $u \in U'$, and let $D = P(U)$. D is reachable from the empty state by induction hypothesis, while $D' = P(U')$ is reachable from D by Lemma II.1. Thus D' is reachable from the empty state as desired. □

II.3 <u>Totality and Uniqueness</u>

An update semantics is *total* if it specifies a meaning for every operation applied to every possible configuration of arguments. A related issue is uniqueness: a semantics is *unique* if it specifies a single meaning for each operation whenever that operation is defined.

The example of Section 4.1 suggests that insertions are not uniquely specified in the UR-environment, and one would suspect that replacements are not unique either. In [BG] we demonstrate that this is the case. What is more surprising is that these operations are *not total*.

Consider the database of Figure II.1, and the insertion $+\underline{R}_1(D,\langle a3,b1,c2\rangle)$. We shall prove that no state D' exists that performs this insertion, and thus the insertion has no meaning. Suppose such a D' does exist and let $U' \in P^{-1}(D')$. Since $R_1 \cup \{\langle a3,b1,c2\rangle\}$ is FD-consistent $D'(\underline{R}_1)$ must have this value, and U' must include the following three tuples:

u1:	<a1 b1 c1 d1>	
u2:	<a2 b2 c2 d2>	
u3:	<a3 b1 c2 dx>,	where dx is a variable.

Since $u1[B] = u3[B]$, the FD $B \rightarrow D$ implies $dx =$ 'd1'; but since $u2[C] = u3[C]$, $C \rightarrow D$ implies $dx =$ 'd2'. Contradiction! Thus the given insertion is undefined.

$$\begin{aligned}\underline{D} = \{&\underline{R}_1 = \langle\{A,B,C\},\{A \rightarrow BC\}\rangle,\\ &\underline{R}_2 = \langle\{B,D\},\{B \rightarrow D\}\rangle,\\ &\underline{R}_3 = \langle\{C,D\}.\{C \rightarrow D\}\rangle\}\end{aligned}$$

D:R_1 =

A	B	C
a1	b1	c1
a2	b2	c2

, R_2 =

B	D
b1	d1
b2	d2

, R_3 =

C	D
c1	d1
c2	d2

Figure II.1. Insertion is Not Total

Replacements have greater felxibility in their external effects. Nonetheless, replacements are not defined in all cases.

Consider the database of Figure II.2, and the replacement. $\&\underline{R}_1(D,\langle a1,b1,c2\rangle,\langle a2,b1,c2\rangle)$. This operation is defined if and only if a consistent D' exists in which $D'(\underline{R}_1) = (R_1 - \{\langle a1,b1,c2\rangle\}) \cup \{\langle a2,b1,c2\rangle\}$. We shall prove that no such state exists. Suppose D' does exist, and let $U' \in P^{-1}(D')$. U' must include the following three tuples:

u1:	<a1 b1 c1 dx1 ex1>,	where dx1, ex1 are variables
u2:	<a1 b2 c2 dx2 ex2>,	where dx2, ex2 are variables
u3:	<a2 b1 c2 dx3 ex3>,	where dx3, ex3 are variables.

Since $u1[A] = u2[A]$, $A \rightarrow D$ implies $dx1 = dx2$; since $u2[C] = u3[C]$, $C \rightarrow D$ implies $dx2 = dx3$; and since $u1[B] = u3[B]$, $B \rightarrow E$ implies $ex1 = ex3$. But this means that $u1[DE] = u3[DE]$, and so $DE \rightarrow A$ implies 'a1' = 'a2'. Contradiction!

Deletion operations, on the other hand, are totally and uniquely specified.

THEOREM II.2. $-\underline{R}_i(D,r_i)$ *is defined and has a unique value for every* $D \in DOM(\underline{D})$ *and tuple* r_i.

<u>Proof</u>. Let $U = J(D)$, $U' = U - \{u \in U | u[ATTR_i] = r_i\}$, and $D' = P(U')$. We claim that D' performs $-\underline{R}_i(D,r_i)$: (1) it is consistent; (2) it exactly performs the deletion since $D'(\underline{R}_i) = U'[ATTR_i] = U[ATTR_i] - \{r_i\} = D(\underline{R}_i) - \{r_i\} = -(D(\underline{R}_i),r_i)$; and (3) it performs the intent of the deletion since $U' \subseteq U$ implies $P(U') \leq P(U) = D$. This establishes that $-\underline{R}_i(D,r_i)$ is defined.

To prove uniqueness, observe that by construction, U' is the *unique subset* of U for which $U'[ATTR_i] = U[ATTR_i] - \{r_i\}$. Let D" be any state that performs $-\underline{R}_i(D,r_i)$, and let $U'' = J(D'')$. To satisfy the intent of the deletion $D'' \leq D$, hence $U'' \subseteq U$, while to exactly perform the deletion, $D''(\underline{R}_i) = -(R_i,r_i)$, hence $U''[ATTR_i] = U[ATTR_i] - \{r_i\}$. But U' is the unique subset of U whose $ATTR_i$ projection has that value. Therefore $J(D'') = U'' = U' = J(D')$, and since J is one-one, $D'' = D'$. Thus D' is the unique state that performs $-\underline{R}_i(D,r_i)$. □

$$\begin{aligned}\underline{D} = \{&\underline{R}_1 = \langle\{A,B,C\},\{BC \rightarrow A\}\rangle,\\ &\underline{R}_2 = \langle\{A,D,E\},\{A \rightarrow D, DE \rightarrow A\}\rangle,\\ &\underline{R}_3 = \langle\{B,E\},\{B \rightarrow E\}\rangle,\\ &\underline{R}_4 = \langle\{C,D\},\{C \rightarrow D\}\rangle\}\end{aligned}$$

D:R_1 =

A	B	C
a1	b1	c1
a1	b2	c2
a1	b1	c2

, R_2 =

A	D	E
a1	d1	e1
a1	d1	e2

, R_3 =

B	E
b1	e1
b2	e2

R_4 =

C	D
c1	d1
c2	d1

Figure II.2. Replacement Is Not Total.

II.4 Proofs of Lemmas 1 and 2

LEMMA 1. (i) $\mathrm{Affect}(+\underline{R}_i) = \{\underline{R}_j \in \underline{D} | \underline{R}_i$ *and* $\underline{R}_j$ *are connected by a path in* $G(\underline{D})$.

(ii) $\mathrm{NoAffect}(+\underline{R}_i) = \{\underline{R}_j \in \underline{D} | \mathrm{ATTR}_j \to \mathrm{ATTR}_i \notin F^+\}$.

Proof. (i) We first prove that if $\underline{R}_i$ and $\underline{R}_j$ are not connected, $\underline{R}_j \notin \mathrm{Affect}(+\underline{R}_i)$. Let D be any non-empty consistent state, and consider an arbitrary (defined) insertion, $+\underline{R}_i(D,r)$. Let D' be any meaning of that insertion, let Ω_+ be the insertion map from D to D', and let $\Omega_i = \{<+\underline{R}_k, r_k> \in \Omega | \underline{R}_i$ and $\underline{R}_k$ are connected$\}$. Observe that $\Omega_i(D)$ performs the given insertion, and $\Omega_i \subseteq \Omega_+$, consequently $\Omega_i = \Omega_+$. $\underline{R}_j \notin \mathrm{Affect}(+\underline{R}_i)$ since Ω_i contains no command of the form $<+\underline{R}_j, r_j>$ by construction.

We now prove that if $\underline{R}_i$ and $\underline{R}_j$ are connected, $\underline{R}_j \in \mathrm{Affect}(+\underline{R}_i)$. Let D be any non-empty consistent state, and let u be a "thoroughly distinct" tuple relative to D, meaning that for all $A \in \mathrm{ATTR}$, $u[A] \notin J(D)[A]$. Let $\Omega_+ = \{<+\underline{R}_j, u[\mathrm{ATTR}_j]> | \underline{R}_i, \underline{R}_j$ are connected$\}$. $\Omega_+(D)$ performs the insertion $+\underline{R}_i(D, u[\mathrm{ATTR}_i])$ and affects every $\underline{R}_j$ connected to $\underline{R}_i$, while for all $\Omega' \subset \Omega$. $\Omega'(D)$ does not perform the insertion.

(ii) If $\mathrm{ATTR}_j \to \mathrm{ATTR}_i \in F^+$, every $+\underline{R}_i(D,r_i)$ that has any effect must certainly affect $\underline{R}_j$; otherwise $J(+\underline{R}_i(D,r_i))$ would be inconsistent. This establishes that $\mathrm{NoAffect}(+\underline{R}_i) \subseteq \{\underline{R}_j \in \underline{D} | \mathrm{ATTR}_j \to \mathrm{ATTR}_i \notin F^+\}$.

To prove inclusion in the opposite direction, suppose $\mathrm{ATTR}_j \to \mathrm{ATTR}_i \notin F^+$. We will construct an insertion that does not affect $\underline{R}_j$. By the completeness property of FDs, there exists a consistent state U' in which $\mathrm{ATTR}_j \to \mathrm{ATTR}_i$ does not hold. For this FD not to hold, U' must contain distinct tuples u and u' such that $u[\mathrm{ATTR}_j] = u'[\mathrm{ATTR}_j]$, yet $u[\mathrm{ATTR}_i] \neq u'[\mathrm{ATTR}_i]$. Let $U = U' - \{u\}$, let $D = P(U)$, and consider the insertion $+\underline{R}_i(D, u[\mathrm{ATTR}_i])$. This insertion has an effect and $P(U')$ performs it. Thus there exists a state that nonextraneously performs the insertion such that $D < D' \leq P(U')$. Since $U[\mathrm{ATTR}_j] = U'[\mathrm{ATTR}_j]$ by construction, it follows that $D(\underline{R}_j) = D'(\underline{R}_j)$, and so the insertion does not affect $\underline{R}_j$ as claimed. □

LEMMA 2. (i) $\mathrm{Affect}(-\underline{R}_i) = \mathrm{Affect}(+\underline{R}_i)$

(ii) $\mathrm{NoAffect}(-\underline{R}_i) = \mathrm{NoAffect}(+\underline{R}_i)$.

Proof. Observe that $-\underline{R}_i(+\underline{R}_i(D,r_i),r_i) = D$ whenever $+\underline{R}_i(D,r_i) \neq D$. This establishes $\mathrm{Affect}(-\underline{R}_i) \supseteq \mathrm{Affect}(+\underline{R}_i)$ and $\mathrm{NoAffect}(-\underline{R}_i) \supseteq \mathrm{Affect}(-\underline{R}_i)$.

To prove inclusion in the opposite direction, let $-\underline{R}_i(D,r_i)$ be any deletion that has an effect, and let $D' = -\underline{R}_i(D,r_i)$. Observe that for all $u \in J(D) - J(D')$, $P(J(D') \cup \{u\})$ is a meaning of $+\underline{R}_i(D,r_i)$. Hence any $\underline{R}_j$ affected by the deletion is also affected by one of these meanings of the insertion, establishing $\mathrm{Affect}(-\underline{R}_i) \subseteq \mathrm{Affect}(+\underline{R}_i)$. Conversely, any $\underline{R}_j$ not affected by the deletion is also not affected by these meanings for the insertion, establishing $\mathrm{NoAffect}(-\underline{R}_i) \subseteq \mathrm{NoAffect}(+\underline{R}_i)$. □

References

[ABU] A.V. Aho, C. Beeri, and J.D. Ullman, "The Theory of Joins in Relational Databases," *ACM Trans. on Database Syst.*, 4,3 (Sept., 1979), 297-314.

[AM] M.A. Arbib, and E.G. Manes, *Arrows, Structures, and Functors--The Categorical Imperative*, Academic Press, New York, 1975.

[Arm] W.W. Armstrong, "Dependency Structures of Database Relationships," *Proc. IFIP 74* (1974), 580-583.

[BBG] C. Beeri, P.A. Bernstein, and N. Goodman, "A Sophisticate's Introduction to Database Normalization Theory," *Proc. 4th Int'l Conf. on Very Large Databases* (1978), 113-124.

[BFH] C. Beeri, R. Fagin, and J.H. Howard, "A Complete Axiomatization for Functional and Multivalued Dependencies," *Proc. ACM SIGMOD Conf.* (Aug. 1977), 47-61.

[Bern] P.A. Bernstein, "Synthesizing Third Normal Form Relations from Functional Dependencies," *ACM Trans. on Database Sys.*, 4 (Dec. 1976), 277-298.

[BG] P.A. Bernstein, and N. Goodman, "What Does Boyce-Codd Normal Form Do?" Tech. Rept. 07-79, Center for Research in Computing Technology, Harvard University (May 1979).

[BMSU] C. Beeri, A. Mendelzon, Y. Sagiv, and J.D. Ullman, "Equivalence of Relational Database Schemes," *Proc. ACM Symp. on Theory of Computing* (May 1979).

[Cod1] E.F. Codd, "Further Normalization of the Data Base Relational Models," in *Data Base Systems* (R. Rustin, ed.), Prentice-Hall, Englewood Cliffs, N.J. (1972), 33-64.

[Cod2] E.F. Codd, "Recent Investigations in Relational Data Base Systems," *Proc. IFIP 74* (1974), 1017-1021.

[Date] C.J. Date, *Introduction to Database Systems*, Addison-Wesley, Reading, MA (1977).

[DB] U. Dayal, and P.A. Bernstein, "The Updatability of Relational Views," *Proc. 4th Int'l Conf. on Very Large Databases* (1978).

[DC] C. Delobel, and R.C. Casey, "Decomposition of a Data Base and the Theory of Boolean Switching Functions," *IBM J. of Res. and Dev.*, 17, 5 (Sept. 1972), 370-386.

[Fag1] R. Fagin, "Multivalued Dependencies and a New Normal Form for Relational Databases," *ACM Trans. on Database Systems*, 2, 3 (Sept. 1977), 262-278.

[Fag2] R. Fagin, "Normal Forms and Relational Database Operators," *Proc. ACM SIGMOD Conf.* (May 1979).

[HLY] P. Honeyman, R.E. Ladner, and M. Yannakakis, "Testing the Universal Instance Assumption," *Inf. Proc. Letters*, 10, 1 (Feb. 12, 1980), 14-19.

[Mar] J. Martin, *Computer Database Organization*, Prentice Hall, Englewood Cliffs, NJ (1975).

[Ris] J. Rissanen, "Independent Components of Relations," *ACM Trans. on Database Sys.*, 2, 4 (Dec. 1977), 317-325.