A SOPHISTICATE'S INTRODUCTION TO DATABASE NORMALIZATION THEORY†

Catriel Beeri Philip A. Bernstein Nathan Goodman

Computer Science Department Aiken Computation Lab. Computer Corp. of America
The Hebrew University Harvard University 575 Technology Square
Jerusalem, ISRAEL Cambridge, MA 02138 Cambridge, MA 02139

Abstract

Formal database semantics has concentrated on dependency constraints, such as functional and multivalued dependencies, and on normal forms for relations. Unfortunately, much of this work has been inaccessible to researchers outside this field, due to the unfamiliar formalism in which the work is couched. In addition, the lack of a single set of definitions has confused the relationships among certain results. This paper is intended to serve the two-fold purpose of introducing the main issues and theorems of formal database semantics to the uninitiated, and to clarify the terminology of the field.

1. INTRODUCTION

1.1 Database Semantics

A database is a collection of information about some enterprise in the world. The role of database semantics is to ensure that stored information accurately represents the enterprise. Database semantics studies the creation, maintenance, and interpretation of databases as models of external activities. A wide variety of database semantic tools exist, ranging from data type constraints, to integrity constraints, to semantic modelling structures used in Artificial Intelligence [26, 36, 39]. This paper is concerned with a specific type of database semantic tool, namely data dependencies—both functional and multi-valued dependencies. This paper surveys the major results in this area. Our aim is to provide a unified framework for understanding these results.

1.2 Database Models

Most work on data dependencies uses the relational data model, with which we assume reader familiarity at the level of [16]. Briefly, a relational database consists of a set of relations defined on certain attributes. R(X) is our notation for a relation named R defined on a set of attributes X.† The relation R(X) is a set of m-tuples, where m=|X|. A relation can be visualized as a table whose columns are labelled with attributes and whose rows depict tuples. Fig. 1 illustrates a relation in this way. The data manipulation operators used in this paper are projection and natural join. The projection of relation R(X) on attributes T is denoted R[T]. If V=X-T, R[T] = \{t\ where t[V] \in R(X)\}, and is defined iff T \subseteq X. (If we visualize R as a table, R[T] is those columns of R labelled with elements of T.) The natural join of relations R and S is denoted R \Join S. Given R(X,Y) and S(Y,Z), where X,Y,Z are disjoint sets, R \Join S = \{<x,y,z>|<x,y> \in R \text{ and } <y,z> \in S\}.

Functional and multivalued dependencies are predicates on relations. Intuitively, a functional dependency (abbr. FD) f:X \rightarrow Y holds in R(X,Y,Z) iff each value of X in R is associated with exactly one value of Y (see Fig. 1). The truth-value of f can of course vary over time, since the contents of R can vary over time. A multivalued dependency (abbr. MVD) g:X \leftrightarrow Y holds in R iff each X-value in R is associated with a set of Y-values in a way that does not depend on Z-values (see Fig. 1). FDs and MVDs are defined formally in the next section.

FIGURE 1. A relation with functional and multi-valued dependencies

Relation: RENTAL-UNITS
Attributes: LANDLORD, ADDRESS, APT#, RENT, OCCUPANT, PETS
Functional dependencies:
ADDRESS, APT# \rightarrow RENT—Each unit has one rent
OCCUPANT \rightarrow ADDRESS, APT#—Every occupant lives in one unit
Multivalued dependencies:
LANDLORD \leftrightarrow ADDRESS—Each landlord can own many buildings
OCCUPANT \leftrightarrow PETS—Each occupant may have several pets

† This work was supported in part by the National Science Foundation under Grant MCS-77-05314.
1.3 Description vs. Content

The interplay between database description and database content is a major theme in database semantics. A database description is called a schema, and contains descriptions for each relation in the database. The description of a single relation is called a relation scheme and consists of the relation name, its attributes and a set of data dependencies. \( R=<T,D> \) denotes a relation scheme \( R \) with attributes \( T \) and dependencies \( D \) (see Fig. 2). We sometimes use the notation \( R(T) \) when \( D \) is either unknown or irrelevant.

FIGURE 2. Formal notation for a relation scheme based on Figure 1.

\[
\text{RENTAL-UNITS} = \\
\langle \text{LANDLORD, ADDRESS, APT#, RENT, OCCUPANT, PETS}, \\
\{\text{ADDRESS, APT# + RENT}; \\
\text{OCCUPANT + ADDRESS, APT#}; \\
\text{LANDLORD + PETS}\rangle
\]

The contents of a relation is called the state or extension of the corresponding scheme, and is a set of tuples as stated above. \( R(T) \) denotes an extension of \( R=<T,D> \). If \( R(T) \) satisfies all dependencies in \( D \), it is called an instance of \( R \) (notationally, \( R \) denotes an instance of \( R \)). A relational database for a schema is a collection of instances, one for each relation scheme in the schema.

In summary, schema and scheme are syntactic objects; database and relation refer to database content. The distinction between schema-related and content-related concepts is often subtle yet important, and we keep it sharp in this paper.

1.4 The Universal Relation Assumption

Most work on data dependencies assumes that all relations in a database are projections of a single relation. Formally, suppose \( R_1(T_1), \ldots, R_n(T_n) \) is a database of intersect, and let \( T = \bigcup_{1 \leq i \leq n} T_i \). It is assumed that a universal relation \( U(T) \) exists, such that \( R_i = U[T_i] \) for \( 1 \leq i \leq n \).

This "universal relation assumption" is a controversial issue in the field. On the other hand, it has formal advantages: it permits us to specify relations solely in terms of their attributes; also it supports the FD and MVD uniqueness rule which states that syntactically identical dependencies are semantically equivalent. On the other hand, many practical applications do not naturally conform to the assumption; to force these applications into the universal relation mold places an added burden on the database administrator, and can obscure desired relationships in the database. The reader should note that all results in this paper make the universal relation assumption, and in some cases they do not extend to alternative frameworks.

1.5 Topics

Formal work in database semantics falls roughly into the areas of schema design and data manipulation.

We limit our attention to the first area, though some of the work we cover has application in the second area also. The problem of schema design is: Given an initial schema, find an equivalent one that is better in some respect. As we will see, different definitions of "equivalence" and "better" lead to startlingly different results.

The paper is organized as follows. Section 2 formally defines data dependencies and reviews their basic properties. Section 3 states the schema design problem more precisely. Then Sections 4, 5 and 6 examine several definitions of schema "equivalence" and several criteria for one schema to be "better" than another. Section 7 ties these ideas together by looking at specific schema design methods. We conclude with an historical look at our field and predications for its future.
It is possible to tell whether \( g \) is implied by \( \Gamma \) using systems of inference rules \([3,6]\). Inference rules permit us to derive new dependencies implied by a given set. A system of inference rules is complete if (a) every \( g \) derivable from \( \Gamma \) is in fact implied by \( \Gamma \), and (b) every \( g \) implied by \( \Gamma \) is derivable using the rules. Fig. 3 shows three complete systems of inference rules for FDs and MVDs. The FD-rules are complete when FDs only are considered. The MVD-rules are complete for MVDs. When FDs and MVDs are considered, all three systems are needed for completeness. Fig. 3 also presents other rules that are useful, though not needed.

2.4 Coverings

A covering of \( \Gamma \) is any set \( \hat{\Gamma} \) such that \( \hat{\Gamma} - \hat{\Gamma}^+ \). \( \hat{\Gamma} \) is nonredundant if no proper subset of it is a covering. One can obtain a nonredundant covering of \( \Gamma \) as follows. A dependency \( g \in \hat{\Gamma} \) is redundant iff \( g \in (\Gamma - \{g\})^+ \). For each \( g \in \Gamma \) the above test is performed using the membership algorithm, and \( g \) is removed from \( \Gamma \) if it is found to be redundant.

2.5 Inherently Difficult Dependency Problems

We list here two inherently difficult dependency problems. Other such problems are presented in \([5,28]\).

Key Finding: Given a set of FDs \( F \) over attributes \( U \), a relation scheme \( R(X) \) where \( X \subseteq U \), and a subset of \( X \)'s keys, determine whether \( R \) has any other keys. This problem is NP-complete \([5]\) (i.e., probably requires exponential time \([2]\)).

Key Listing: Given \( F \) and \( R \) as above, list all keys of \( R \). This problem has exponential worst-case time since there are relation schemes with an exponential number of keys \([40]\).

3. THE SCHEMA DESIGN PROBLEM

We now return to the problem of schema design. Our treatment considers one particular schema design scenario. We assume that a schema \( S_A \) containing a single relation scheme is given. The problem is to design a schema \( S_D \) that is equivalent to \( S_A \), but is better in some specified way. Let \( S_A = (U,<T,r>), \) and \( S_D = (U,<T_i,r_i> \mid i=1,\ldots,n) \). In our scenario \( S_D \) contains "projections" of \( U \), i.e., each \( T_i \subseteq T \) and \( T_i \) is "inherited" from \( T \). For FDs, "inheritance" means \( r_i \) is a covering of the FDs in \( \Gamma^+ \) that are defined for \( T_i \). For MVDs, the situation is more complicated and will not be elaborated here. An instance \( U \) of \( U \) is represented in \( S_D \)'s database by \( \{U(T_i) \mid i=1,\ldots,n\} \).

Our study of schema design can now be considered to be a study of the mapping between \( S_A \) and \( S_D \) and between the set of instances of \( S_A \) and the sets of instances of \( S_D \).

4. THE PRINCIPLE OF REPRESENTATION

A clear requirement for schema \( S_D \) to replace \( S_A \) is that \( S_D \) and \( S_A \) be equivalent; that is, \( S_D \) must represent the same information as \( S_A \). Different researchers formulate this concept in different ways—ways that lead to startlingly different conclusions. In the following, let \( S_F = (U, \{g \in \Gamma \}) \) and \( S_G = (U, \{g \in \Gamma_i \}) \mid i=1,\ldots,n \).
Definition Rep1. $S_D$ represents the same information as $S_4$ if they contain the same attributes; that is, if $\cap_{i=1}^{n} R_i = \cap_{i=1}^{n} S_i$.

This definition is inadequate because it ignores relationships among attributes. By this definition, the schemas in Figs. 4 and 5 are equivalent to the one in Fig. 2, even though they contain no data dependencies.

**FIGURE 4.** A schema equivalent to one in Fig. 2 under Def. Rep1.

\[
S_D = \{ R_1, R_2, R_3 \}
\]

\[
R_1 = \langle \text{LANDLORD}, \text{RENT}, \text{PETS} \rangle, ()
\]

\[
R_2 = \langle \text{ADDRESS}, \text{APT#} \rangle, ()
\]

\[
R_3 = \langle \text{OCCUPANT}, () \rangle
\]

**FIGURE 5.** Another schema equivalent to one in Fig. 2 under Rep1.

\[
S_D = \{ R_1, R_2, R_3, R_4, R_5, R_6 \}
\]

\[
R_1 = \langle \text{LANDLORD}, () \rangle
\]

\[
R_2 = \langle \text{RENT}, () \rangle
\]

\[
R_3 = \langle \text{ADDRESS}, () \rangle
\]

\[
R_4 = \langle \text{APT#}, () \rangle
\]

\[
R_5 = \langle \text{OCCUPANT}, () \rangle
\]

\[
R_6 = \langle \text{PETS}, () \rangle
\]

Definition Rep2. $S_D$ represents the same information as $S_4$ if they have the same attributes and the same data dependencies.

When only FDs are involved, this definition can be made precise. The rules of $S_4$ are $\Gamma'$. The FDs of $S_D$ are $(\cup_{i=1}^{n} \Gamma_i)^+$. $S_D$ represents $S_4$ if $\Gamma' = (\cup_{i=1}^{n} \Gamma_i)^+$, i.e., if $(\cup_{i=1}^{n} \Gamma_i)$ is a covering of $\Gamma$.

However, there is a problem with the definition as stated. The inference rules in Sec. 2 are only defined with respect to dependencies in a single relation. Since $S_D$ involves multiple relations, it is not obvious that those inference rules can validly be applied to it. Suppose, $S_4 = \{ (x,y,z), \Gamma = (x+y+z) \}$, and $S_D = \{ (x,y,z), \Gamma' = (x+y+z), \Gamma_2 = \{ (x,z), \Gamma_2 = (y+z) \} \}$. Notice that $x+y$ is in $\Gamma'$ and that $S_D$ does not depend on a relation scheme in which $x+y$ is defined!

This problem is rectified by the "universal relation assumption" (Sec. 1) and the "inverse projectivity property of FDs" (Sec. 2). Let $R_1$ and $R_2$ be instances of $F_1$ and $F_2$; define $R_1^* = \{ R_1, UT_1 \}$, $\Gamma' = (\Gamma_1 + 1 \cup \Gamma_2)^+$, and define $R_2' = R_1^* R_2$. From the inverse projectivity property it can be shown that $R_2'$ is an instance of $R_1^*$ if $R_1$ and $R_2$ are instances of $R_1$ and $R_2$. Thus the FD $X + Z$ (which is in $\Gamma' + 1$) is valid in $R_2'$. Moreover, by the universal relation assumption $R_1$ and $R_2$ are projections of $U$, as is $R_2' [XZ]$. Consequently, the user can obtain the "extension" of $X + Z$ from $S_D$, even though $X + Z$ is not explicitly represented. In fact, all FD inference rules can be "simulated" by relational operators applied to relations containing the FDs. It follows that all FDs in $\Gamma'$ can be retrieved from $S_D$ if $S_D$'s schemes contain a covering of $\Gamma$.

**FIGURE 6.** An example of a lossy join.

Let $S_D = \{ \text{LANDLORD-UNITS} \}$ defined in Fig. 2, with instance of Fig. 1.

Let $S_4 = \{ \text{LANDLORD-APT#, APT#-RENT, PERSON-PETS} \}$

\[
\text{LAND-APT#} = \{ \langle \text{LANDLORD-APT#}, () \rangle \}
\]

\[
\text{APT#-RENT} = \{ \langle \text{APT#, ()} \rangle \}
\]

\[
\text{PERSON-PETS} = \{ \langle \text{OCCUPANT, PETS, ADDRESS}, () \rangle, \langle \text{OCCUPANT, ADDRESS+OCCUPANT++PETS} \rangle \}
\]

Instances corresponding to Fig. 1 are

<table>
<thead>
<tr>
<th>LAND-APT# (LANDLORD, APT#)</th>
<th>APT#-RENT (APT#, RENT)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wizard, #1</td>
<td>#3, $50</td>
</tr>
<tr>
<td>Wizard, #1</td>
<td>#1, $50</td>
</tr>
<tr>
<td>Wizard, #2</td>
<td>#2, $75</td>
</tr>
<tr>
<td>Codd, #1</td>
<td>#1, $500</td>
</tr>
<tr>
<td>Codd, #2</td>
<td>#2, $600</td>
</tr>
</tbody>
</table>

**LAND-APT# (LANDLORD, APT#) APT#-RENT (APT#, RENT)**

Attributes: LANDLORD, APT#, RENT

Tuples: Wizard, #1, $50
Wizard, #1, $50
Wizard, #1, $500
Wizard, #2, $75
Wizard, #2, $600
FIGURE 6 continued

Attributes: LANDLORD, APT#, RENT

Tuples:  
Codd, #1, $ 50  
Codd, #1, $500  
Codd, #2, $ 75  
Codd, #2, $600

Note that each LANDLORD is associated with RENTS charged by the other.

Formally we say that $S_D$ has the lossless join property if for each instance $U$ of $S_U$, 
\[ U = \bigstar U[T_1]. \]

When only pairs of relations are considered, we have the following results.

FACT 1: If $U = \langle T, F \rangle$ (that is, only FDs are given) then for sets $T_1, T_2$ such that $T_1 \cup T_2 = T$, 
\[ (U[T_1], U[T_2]) \text{ has the lossless join property iff either } T_1 \cap T_2 + T_1 \text{ or } T_1 \cap T_2 + T_2 \text{ is in } \Gamma'[31]. \]

FACT 2: For $U = \langle T, T \rangle$ and for $T_1, T_2$ as above, 
\[ (U[T_1], U[T_2]) \text{ has the lossless join property iff } T_1 \cap T_2 + T_1 \text{ (and, by rule MVDO, } T_1 \cap T_2 + T_2 \text{) is in } \Gamma'[23]. \]

These facts are stated as properties of univocally quantified sets of instances; i.e., the conditions of Facts 1 & 2 hold iff all instances of the given schemas have lossless joins. It is possible, though, for the conditions not to hold, yet for specific instances to have lossless joins, nonetheless. Facts 1 & 2 can be adapted for specific instances as follows.

FACT 1*: Given $U = \langle T, T \rangle$, and $T_1, T_2$ as above, 
An instance $U[U[T_1], U[T_2]]$ holds in $U$ iff (but not only if) $T_1 \cap T_2 + T_1$ or $T_1 \cap T_2 + T_2$ holds in $U$.

FACT 2*: Given $U = \langle T, T \rangle$, and $T_1, T_2$ as above, 
An instance $U[U[T_1]*U[T_2]]$ holds in $U$ iff $T_1 \cap T_2 + T_1$ (and by MVDS $T_1 \cap T_2 + T_2$) holds in $U$.

When more than pairs of relations are considered, the situation is more complex. An algorithm for deciding the lossless join property in general is presented in [1]. The algorithm requires polynomial time for FDs but may require exponential time for MVDs. Another interesting result is that for all $n > 2$ there are sets of $n$ relation schemes that have the lossless join property, for which no proper subset has this property.

Definition Rep4. $S_D$ represents the same information as $S_\phi$ iff there exists a one-to-one mapping between the databases of $S_\phi$ and databases of $S_D$.


If only FDs are given, Rep4 is identical to the notion of independent components [31], and the following is proved:

Let $U = \langle T, F \rangle$, $R_1 = \langle T_1, F_1 \rangle$, and $R_2 = \langle T_2, F_2 \rangle$. 
\[ (R_1, R_2) \text{ are independent components of } U \]

(a) $F_1 + = F_2$, and (b) $F^*$ contains $T_1 \cap T_2 + T_1$ or $T_1 \cap T_2 + T_2$ [31].

Comparison of Definitions. Fig. 5 illustrated a schema equivalent to the schema of Fig. 2 by Rep1 but not by Rep2, Rep3, or Rep4. Fig. 7 differentiates between Rep2 and Rep3. Fig. 7(a) is similar to an example in [16, p. 165]. $S_D$ is equivalent to $S_\phi$, under Rep3 but not Rep2; it would be considered a good design by [16,23], but not by [17]. In Fig. 7(b), $S_\phi$ is equivalent to $S_D$ by Rep2 but not by Rep3; it would be approved by [17], but not [23,37]. These differences of opinion are examined further in later sections.

FACT 1: For $U = \langle T, T \rangle$ and for $T_1, T_2$ as above, 
\[ \{E_1(T_1), s_2(T_2)\} \text{ has the lossless join property iff either } T_1 \cap T_2 + T_1 \text{ or } T_1 \cap T_2 + T_2 \text{ is in } \Gamma'[33]. \]

FACT 2: For $U = \langle T, T \rangle$, and for $T_1, T_2$ as above, 
\[ \{E_1(T_1), s_2(T_2)\} \text{ has the lossless join property iff } T_1 \cap T_2 + T_1 \text{ is in } \Gamma'[23]. \]

5. THE PRINCIPLE OF SEPARATION

The next question is to understand how $S_D$ can be "better than" $S_\phi$. One way is for "independent relationships" to be represented by $S_D$ in independent relation schemes. To illustrate this point, let $S_D$ be the RENTAL-UNITS scheme of Figs. 1 & 2, and suppose we want to add a new LANDLORD to the database. This can only be done if values for other attributes are given, too. The new LANDLORD must be associated with an ADDRESS; the ADDRESS must be associated with an APT#; the APT# pair requires a RENT and an OCCUPANT; and the OCCUPANT needs PETS. So to add a new LANDLORD, $S_D$ forces us to add information that is at most distantly related to him. By the same token, when
the last PET of the last OCCUPANT of the last APT# of a given ADDRESS runs away, the association between LANDLORD and ADDRESS is also destroyed. Another problem with S is data redundancy. Each LANDLORD is represented in four tuples although each only owns one building. To change the building owned by Codd, say, requires that all four tuples with ADDRESS = "3 NF St" be updated. If some of these tuples were forgotten, the database would be inconsistent, meaning that some dependencies would no longer hold. In this case, LANDLORD --> ADDRESS would no longer hold.

These difficulties are caused by a lack of separation in S. To overcome these difficulties, a series of database normal forms have been proposed, four of which are of interest. Before defining them, we present several preliminary concepts.

Let S = \{(T_i, r_i) \in \mathbb{R} \mid i = 1, \ldots, n\} be a schema and let \( \Gamma = \{t_i \mid i = 1, \ldots, n\} \) be a set of attributes. (1) \textit{Superkey} -- Let \( X \subseteq T_i \) be a superkey of \( T_i \) if \( X \in \Gamma \). (2) \textit{Prime attribute} -- Let \( A \in T_i \); \( A \) is prime in \( T_i \) if \( A \) is in any key of \( T_i \). (4) \textit{Transitive dependence} -- Let \( A \subseteq T_i \) and \( X \subseteq T_i \); \( A \) is transitively dependent on \( X \) in \( T_i \) if there exists \( Y \subseteq T_i \) such that \( X \to Y \subseteq T_i \), \( Y \to T_i \), and \( A \subseteq Y \). (5) \textit{Trivial FD} -- \( X \to Y \) is trivial, meaning it holds in all relations, if \( Y \subseteq X \).

We now define four normal forms of interest.

1. Third Normal Form (abbr. 3NF):
   \( \mathcal{S} \) is in 3NF if none of its nonprime attributes is transitively dependent on any of its keys.

2. Boyce-Codd Normal Form (BCNF):
   \( \mathcal{S} \) is in BCNF if for all nontrivial FDs in \( \mathcal{S} \), \( X \to Y \subseteq \Gamma \) and \( X \neq Y \) imply \( X \to Y \subseteq T_i \) for some \( T_i \).

3. Weak Fourth Normal Form (W4NF):
   \( \mathcal{S} \) is in W4NF if for all nontrivial FDs in \( \mathcal{S} \), \( X \to Y \subseteq \Gamma \) and \( X \neq Y \) imply \( X \to Y \subseteq T_i \) for some \( T_i \).

4. Fourth Normal Form (4NF):
   \( \mathcal{S} \) is in 4NF if for all nontrivial FDs in \( \mathcal{S} \), \( X \to Y \subseteq \Gamma \) and \( X \neq Y \) imply \( X \to Y \subseteq T_i \) for some \( T_i \).

Notice that 3NF is a weak version of BCNF, and W4NF is a weak version of 4NF. Also W4NF implies 3NF and 4NF implies BCNF. BCNF and 4NF always succeed in separating independent relationships into separate schemes. This is illustrated in Fig. 8. Notice that \( \mathcal{S} \) in Fig. 8 Rep2-represents RENTAL-UNITS. There are cases, though, where the stronger normal forms cannot be achieved and we must settle for the weaker forms. Fig. 9 shows an example of this sort. The following formalize this observation.

Let \( S_D = \{(p_i) \mid p_i \in \mathbb{P} \} \).

FACT 3: There always exists a 3NF schema that Rep2-represents \( S_D \) [7].

FACT 4: There need not exist a BCNF schema that Rep2-represents \( S_D \). Moreover the question, "Is scheme \( \mathcal{S} \) in BCNF?" is NP-hard [5].

FACT 5: There need not exist a 4NF schema that Rep2-represents \( S_D \). (Follows from Fact 4 when \( G = \emptyset \).) It is not known whether a W4NF scheme

*1NF simply requires that relations be "flat", non-hierarchical. 2NF is a weak form of 3NF and is subsumed by it [14,16].

FIGURE 8. 4NF schema and instance corresponding to RENTAL-UNITS (Figs. 1 & 2)

\[
S_D = \{\text{OWNs, CHARGES, LIVES, LOVES}\}
\]

\[
\begin{array}{lcr}
\text{OWNs} &=& \langle\text{LANDLORD, ADDRESS}, \langle\text{LANDLORD} \to \text{ADDRESS}\rangle\rangle \\
\text{CHARGES} &=& \langle\text{ADDRESS, APT#, RENT}, \langle\text{ADDRESS, APT#} \to \text{RENT}\rangle\rangle \\
\text{LIVES} &=& \langle\langle\text{OCCUPANT, ADDRESS, APT#}\rangle \\
\text{LOVES} &=& \langle\langle\text{OCCUPANT, PETS}\rangle \\
\text{OWNs} &=& \langle\text{LANDLORD, ADDRESS}\rangle \text{ LIVES} \langle\text{OCCUPANT, ADDRESS, APT#}\rangle \\
\text{LANDLORD} &=& \text{Codd, 3 NF St, #3} \\
\text{ADDRESS} &=& \text{3 NF St, #3} \\
\text{APT#} &=& \text{#3} \\
\text{RENT} &=& \text{#3} \\
\text{OCCUPANT} &=& \text{Oz, #3} \\
\text{PETS} &=& \text{Wizard, 02, #3} \\
\text{LIVES} &=& \text{Codd, 3 NF St, #3} \\
\end{array}
\]

FIGURE 9. W4NF schema and instance.

\[
S_D = \{\text{OWNs, CHARGES, LIVES, LOVES}\}
\]

\[
\begin{array}{lcr}
\text{CHARGES} &=& \langle\text{ADDRESS, APT#, RENT}, \langle\text{ADDRESS, APT#} \to \text{RENT}\rangle\rangle \\
\text{LIVES} &=& \langle\langle\text{OCCUPANT, ADDRESS, APT#}\rangle \\
\text{LOVES} &=& \langle\langle\text{OCCUPANT, PETS}\rangle \\
\text{OWNs} &=& \langle\text{LANDLORD, ADDRESS}\rangle \text{ LIVES} \langle\text{OCCUPANT, ADDRESS, APT#}\rangle \\
\text{LANDLORD} &=& \text{Codd, 3 NF St, #3} \\
\text{ADDRESS} &=& \text{3 NF St, #3} \\
\text{APT#} &=& \text{#3} \\
\text{OCCUPANT} &=& \text{Oz, #3} \\
\text{PETS} &=& \text{Wizard, 02, #3} \\
\text{LIVES} &=& \text{Codd, 3 NF St, #3} \\
\end{array}
\]
Another observation to make from Fig. 9 is that 4NF doesn't achieve total separation in the way 4NF does. OWNS' has redundant information and suffers the same kind of update anomalies as RENTAL-UNITS does. The same is true of 3NF vs. BCNF. And since 4NF and BCNF cannot always be achieved under Rep2 (Facts 4 & 5), we must conclude that Rep2 and total separation are incompatible concepts. This result is both surprising and fundamental; it holds for non-computerized databases as well as computerized ones, and has applicability in all data models.

This result has been interpreted differently by some workers [16,24] who argue that BCNF and 4NF schemas should be obtained even if Rep2 is not achieved. We saw such a case in Fig. 7(a), which we replicate in Fig. 10. In that example, SD violates Rep2 because it does not include ADDRESS,APT#+OCCUPANT. Without this FD legal instances of S0 can correspond to illegal instances of S4, and may represent illegal conditions in the real world (see Fig. 10(b)). It is suggested in [16] that these illegal instances be prevented by adding ADDRESS,APT#+OCCUPANT to SD as an "interrelational constraint." However, because Rep2 is incompatible with BCNF in this case, this suggestion is futile. If we add the suggested interrelational constraint, the two relations can no longer be updated independently, which simply defeats the original goal of separation.

In other words, while total separation is a goal of schema design, there simply are cases where it cannot be achieved.

FIGURE 10. An instance of SD that is not an instance of S4.

| S4 | Dweller = <ADDRESS,APT#,#OCCUPANT>
|    | ADDRESS,APT#+OCCUPANT; OCCUPANT+ADDRESS;ADDRESS|

| SD | ADD-OCC = <ADDRESS,OCCUPANT>; APT-OCC = <APT,OCCUPANT> |

S0 does not Rep2-represent S4. SD Rep3-represents S4.

(a)

Relation: ADD-OCC
Attributes: ADDRESS, OCCUPANT
Tuples: Oz, Timman
        Oz, Witch
        Oz, Lion
        Oz, Scarecrow

Relation: APT-OCC
Attributes: APT#, OCCUPANT
Tuples: #3, Timman
        #1, Witch
        #2, Lion
        #2, Scarecrow

Each relation has legal contents—all dependencies hold. But, ADDRESS,APT#+OCCUPANT does not hold in ADD-OCC+APT-OCC.

6. THE PRINCIPLE OF MINIMAL REDUNDANCY

Another goal in designing SD is minimal redundancy; SD must contain the information needed to represent S4 but it should not contain the information redundantly. The meaning of minimal redundancy depends on the definition of representation. Only by knowing what it means to represent information can we judge whether a certain representation is redundant.

Virtually all work on schema design adopts some notion of minimal redundancy, although often this point is addressed intuitively. Consequently our treatment of redundancy must be sketchier than the previous sections. We present here different definitions of redundancy analogous to the definitions of representation in Sec. 4. In the following, let SD = (Ri = (Ti, ri) | i = 1, ..., n).

Definition Rel1. Ri ∈ SD is redundant if T = (Ri +)+ = (Rj -)+ - (Ri -)+ for all databases of SD. This approach, like Definition Rel1, is unsatisfactory since it does not account for relationships among attributes. Also, minimal redundancy under Rel1 is always attained in S4 since each attribute appears only once.

Definition Rel2. Ri ∈ SD is redundant if Ri+ is redundant in SD if

Ri+ is redundant in SD if

Ri+ is redundant if Rj+ - Ri+ = Rj- - Ri- for all databases of SD.

Definition Rel3. Ri ∈ SD is redundant if there is a one-to-one correspondence between the set of instances of S0 and the set of instances of R1. This definition, like Rel1, combines data and dependency aspects of schema design.

We note, in conclusion, that other approaches to redundancy are possible, e.g., using as a
7. SCHEMA DESIGN METHODS

Traditionally, schema design has been called "database normalization" in the literature in this area and two approaches are prominent: synthesis [5,7], and decomposition [14,20,21,25,41]. This section describes both approaches, explaining how they interpret and achieve the schema design principles discussed earlier.

The key difference between synthesis and decomposition lies in the definition of representation that each adopts. In synthesis $R_2$ 'represents' input $S_4$, whereas with decomposition $S_D$ 'represents' $S_4$. This difference leads to a series of other discrepancies between the methods:

1. Since $R_2$ is not compatible with total separation (Sec.5), synthesis can only achieve 3NF and not higher normal forms; decomposition, on the other hand, is not limited in this way. (2) $R_2$ leads to the $R_2$ definition of redundancy, while $R_3$ leads to $R_3$; therefore synthesis strives for minimality of dependencies while decomposition strives for minimality of data content. (3) Because definitions $R_2$ and $R_3$ do not easily extend to MVDs, it is not known how (or if) synthesis can handle MVDs; decomposition, on the other hand, is straightforwardly extendable to MVDs. (4) Finally, as explained in Sec. 5, $R_3$ does not guarantee that all instances of $S_4$ correspond to legal instances of $S_4$; thus schemas produced by decomposition admit instances that would not be permitted by synthesized schemas. These differences are summarized in Figure 11.

FIGURE 11. Differences between principal Normalization Methods

<table>
<thead>
<tr>
<th>Method described by</th>
<th>Synthesis</th>
<th>Decomposition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bernstein [7]</td>
<td>$R_2$</td>
<td>$R_3$</td>
</tr>
<tr>
<td>Definitional Changes</td>
<td>$R_2$ has same</td>
<td>$R_2$ has same</td>
</tr>
<tr>
<td>Reps</td>
<td>$S_4$ databases</td>
<td>$S_4$ databases</td>
</tr>
<tr>
<td>Dependencies</td>
<td>FDS</td>
<td>FDS + MVDs</td>
</tr>
<tr>
<td>Normal Form</td>
<td>3NF</td>
<td>4NF,BCNF</td>
</tr>
<tr>
<td>Definition of Redundancy</td>
<td>$R_2$&quot;redundancy of data content&quot;</td>
<td>$R_3$&quot;redundancy of data content&quot;</td>
</tr>
<tr>
<td>Redundancy of Dependence (attained)</td>
<td>$R_2$ + $R_3$</td>
<td>$R_3$ + $R_3$</td>
</tr>
<tr>
<td>Instances admitted by $S_D$</td>
<td>$S_4$ as</td>
<td>$S_4$ as</td>
</tr>
</tbody>
</table>

*An decomposition approach is suggested by [Rissanen, 77] in which $S_D$ $R_4$-represents $S_4$. This approach is algorithmically similar to the other decomposition approaches so will not be discussed separately.

7.1 The Synthesis Approach

We discuss the synthesis approach in terms of the specific method of [7]. A central concept of this method is embodied $FDS$, which are FDs implied by keys. Formally, given $R_i = \langle T_i, F_i \rangle$, $X \rightarrow A$ is embodied in $R_i$ if $X \times A \in F_i$, and $X$ is a superkey of $R_i$. Fig. 12 presents a simplified synthesis algorithm (called SYN1) that uses embodied FDs to construct an $S_D$ $R_2$-representing $S_4$. SYN1 is a first step towards a correct synthesis algorithm. SYN1 is not yet correct because $S_D$ is not necessarily in 3NF, so transitive dependencies can be exhibited within individual FDs, due to extraneous attributes in their left hand sides. An attribute is extraneous in an FD if it could be eliminated from the FD without affecting the closure ($F'$).

Let us precede SYN1 with a step that eliminates extraneous attributes from the left sides of FDs in $F$, and call the resulting algorithm SYN1'. SYN1' produces schemas that $R_2$-represent the input and are guaranteed to be in 3NF. Algorithm SYN1' thus meets the representation and separation goals of schema synthesis.

The next step is to achieve minimality. Let $F$ be the nonredundant covering of $F$ obtained by SYN1' after excising extraneous attributes, and suppose $F$ includes $V \times W$ and $X \times Y$, $X \neq V$. Clearly these FDs will be embodied in different relation schemes. But suppose $V$ and $X$ are equivalent, i.e., $V \times X$ are in $F$. Then $V \times W$ and $X \times Y$ can be embodied in one relation scheme with both $V$ and $X$ as keys. Doing so reduces the number of synthesized schemes and makes explicit the equivalence of $X$ and $V$.

So, we add a stage to SYN1' to find and merge all relation schemes with equivalent keys. Unfortunately, this modification takes a step backward: it no longer produces 3NF schemes! When we merge relation schemes we also add FDs to $F$ from

FIGURE 12. Simplified Synthesis Algorithm

Algorithm SYN1

Input: $S_4 = \{d = \langle T, F \rangle \}$
Output: $S_D = \{R_i = \langle T_i, F_i \rangle \}$ where $R_i$ = all attributes appearing in $T_i$

1. (Find Covering). Find a nonredundant covering $F$ of $F$.
2. (Partition). Partition $F$ into "groups", $F_i$, $i = 1, \ldots, n$, such that all FDs in each $F_i$ have the same left hand side, and no two groups have the same left hand side.
3. (Construct Relations). For each $F_i$ construct a relation scheme $R_i = \langle T_i, F_i \rangle$ where $T_i$ = all attributes appearing in $F_i$.

Important Fact: The left hand side of every FD in $F_i$ is a superkey of $R_i$; each FD in $F_i$ is embedded in $R_i$. 


(\text{Rep}_2) \) which may thereby cause \( F \) to become redundant. A final stage is needed to eliminate this redundancy. This modification brings us to algorithm \text{SYN2} (Fig. 13), which is our final schema synthesis algorithm. The following facts, are proved in \cite{71}, establishing that \text{SYN2} achieves the three schema design principles. Given \( S_\phi = \{U = \langle T, F \rangle \} \) and \( S_D = \) the result of applying \text{SYN2} to \( S_\phi \).

\textbf{FACT 7:} \( S_D \) \text{Rep}_2-represents \( S_\phi \).

\textbf{FACT 8:} \( S_D \) is in 3NF.

\textbf{FACT 9:} \( S_D \) is minimally redundant under definition \text{Red}_2. In fact \( S_D \) is minimal in an even stronger sense. Let \( S_U = \{U_i = \langle T_i, F_i \rangle \} \) \text{Rep}_2-represents \( S_\phi \) and all FDs in \( S_U \) are \text{embodied} FDs. \( S_D \) contains no more relation schemes than any other scheme in \( S_U \). In other words, \( S_D \) is the smallest schema that can \text{Rep}_2-represent \( S_\phi \) using just keys.

\textbf{FIGURE 13. A Correct Synthesis Algorithm} \cite{5}.

\textbf{Algorithm SYN2}

\textbf{Input:} \( S_\phi = \{U = \langle T, F \rangle \} \)

\textbf{Output:} \( S_D = \{R_i = \langle T_i, F_i \rangle \} \)

1. (Eliminate Extraneous Attributes) Eliminates extraneous attributes from the left side of each FD in \( F \), producing the set \( F' \).
2. (Find Covering) Find a nonredundant covering \( F^\ast \) of \( F' \).
3. (Partition) Partition \( F^\ast \) into groups \( F_i \), \( i = 1, \ldots, n \), as in step 3 of \text{SYN1}.
4. (Merge Equivalent Keys) Set \( J:=\emptyset \). For each pair of groups \( F_i, F_j \), with left hand side \( X_i \), \( X_j \) do the following: If \( X_i \cup X_j \subseteq \text{F+} \) and \( X_i \cup X_j \subseteq \text{F}' \), merge \( F_i \) and \( F_j \), add \( X_i \cup X_j \), and remove them from \( F^\ast \).
5. (Eliminate Transitive Dependencies) Find a minimal \( \hat{F} \subseteq \hat{F}' \) such that \( (\hat{F} + J)^* = (\hat{F}' + J)^* \). Delete each element of \( \hat{F}' \) from the group in which it appears. For each \( X_i \cup X_j \) in \( J \), add it to the corresponding group.
6. (Construct Relations) For each \( F_i \) construct a relation scheme \( R_i = \langle T_i, F_i \rangle \) where \( T_i = \) all attributes appearing in \( F_i \).

7.2 The Decomposition Approach

Fig. 14 shows a typical decomposition algorithm (which we call algorithm \text{DEC}) adapted from \cite{29}. \text{DEC} achieves the representation and separation goals of decomposition but does not achieve minimal redundancy. These conclusions are stated formally as follows. Given \( S_\phi = \{U = \langle T, F \rangle \} \) and \( S_D = \) the result of applying \text{DEC} to \( S_\phi \).

\textbf{FACT 10:} \( S_D \) \text{Rep}_3-represents \( S_\phi \).

\textbf{Reason:} Whenever a scheme \( R \) is decomposed into \( R_1 \) and \( R_2 \) in step 1 of \text{DEC}, \( R_1 \mid R_2 \) has the lossless join property (by Fact 2).

\textbf{FIGURE 14. Basic Decomposition Algorithm}

\textbf{Algorithm DEC}

\textbf{Input:} \( S_\phi = \{U = \langle T, F \rangle \} \)

\textbf{Output:} \( S_D = \{R_i = \langle T_i, F_i \rangle \} \)

1. (Initialize) Set \( k:=0 \).
2. (Test for Separation) If all schemes in \( S_k \) are in 4NF, then output \( S_D := S_k \).
3. (Decompose) Set \( S_{k+1} := \emptyset \). Let \( R_i = \langle T_i, F_i \rangle \) be any non-4NF scheme in \( S_k \) and set \( S_k := S_k \mid R_i \).
   Decompose \( R_i \) into \( R_{i,1} \) and \( R_{i,2} \) as follows:
   (1) Let \( X \rightarrow Y \) be any non-trivial MVD in \( F \) defined on \( R_i \).
   (2) Let \( R_{i,1} := X \cup Y,\{g \in \Gamma_i \mid g \text{is defined on } R_{i,1}\} \).
   (3) Define \( R_{i,2} := T_i \setminus Y,\{g \in \Gamma_i \mid g \text{is defined on } R_{i,2}\} \).
   (4) Set \( S_{k+1} := S_k \cup \{R_{i,1}, R_{i,2}\} \).
   (5) Repeat for all \( R_i \in S_k \).
4. (Eliminate Some Redundancy) For each \( R_{i,1}, R_{i,2} \in S_k \),
   If \( T_i \subseteq T_j \) set \( S_{k+1} := S_k \mid \{R_j\} \).
5. (Iterate) Set \( k:=k+1 \). Go to step 2.

\textbf{FACT 11:} \( S_D \) is in 4NF.

\textbf{Reason:} \text{DEC} will not stop until \( S_D \) is in 4NF.

\textbf{FACT 12:} \( S_D \) is not necessarily minimally redundant under \text{Rep}_3 (or most other reasonable definitions).

\textbf{Reason:} Fig. 15 shows two ways \text{DEC} could decompose the same schema, one of which is minimal and one of which is not. Few minimality facts have been established regarding decomposition, and it is not even known whether minimal schemas can be produced by non-deterministic decomposition. Also, it is not known whether decomposition can consider coverings of dependencies rather than entire closures; in the specific case of Fig. 15(a) minimality would be guaranteed if \( S_D \) were decomposed using a nonredundant, nonextraneous covering of \( F \).

Algorithmic aspects of decomposition have not been considered fully either, and current algorithms have high computational complexity. For example, \text{DEC} is probably very slow, because the question "Is schema \( S \) in 4NF?" is NP-hard and \text{DEC} asks this question repeatedly. A related problem is caused by using closures rather than coverings. Closures can be exponentially large and their use can lead to exponential worst case running time.

Another problem is that decompositions are not unique. At each stage the algorithm may have several decomposition choices with different choices leading to very different outputs (e.g., Fig. 15). Some choices produce "natural looking" schemas while other choices may lead to bizarre results (see Fig. 16). Also, the dependencies in the output schema can depend idiosyncratically on the input and the algorithm (see Fig. 17).
Notice in Fig. 17 that $S_1^3$ represents $S_2^3$ of $S_2^3$.

Representations $S_1^3$ and $S_2^3$ both of $S_2^3$ represent a third schema $S_4^3$. Nonetheless, $S_1^3$ and $S_2^3$ have substantially different sets of legal instances. Heuristics for choosing "good" decompositions are suggested in [41], but no rules are known to work in all cases.

**FIGURE 15.** Algorithm D does not achieve minimal redundancy (italicized attributes become relations schemes).

$$S_0 = \{I = \{A, B, C, D\}, \Gamma = \{B \rightarrow C: D \rightarrow B: BC \rightarrow A\}\}$$

**FIGURE 16.** Natural and unnatural 4NF schemas produced by decomposition.

$$S = \{I = \{\text{LANDLORD, ADDRESS, APT\#, RENT, OCCUPANT, PETS}\}$$

$$= \{\text{LANDLORD, ADDRESS, APT\#, RENT, OCCUPANT, PETS}\}$$

$$\rightarrow \text{LANDLORD, ADDRESS, APT\#, RENT, OCCUPANT, PETS}$$

**FIGURE 17.** Idiosyncratic behavior of decomposition

$$S_1^3 = \{I_1^3 = \{\text{LANDLORD, ADDRESS, APT\#, RENT, OCCUPANT, PETS}\}$$

$$= \{\text{LANDLORD, ADDRESS, APT\#, RENT, OCCUPANT, PETS}\}$$

**8. HISTORY, CONCLUSIONS, AND FUTURE WORK**

The history of database normalization theory begins with Codd's early work [14]. Codd introduced the notion of FD, but did not formalize it. The first formalizations of FDs were by Delobel [17], Rissanen and Delobel [33], and Delobel and Casey [20]; these authors concentrated on formal properties of dependencies and their relationships to the decomposition approach. They were followed by Armstrong [3] who introduced the notion of completeness of inference rules and proved the completeness of a set of rules for FDs. This work laid the groundwork for the formal theory that has developed since. The earliest synthesis algorithm was an informal one described by Wang and Wedekind [38]. Bernstein [17] followed with a synthesis algorithm that used Armstrong's theory to prove properties of synthesized schemas. Bernstein's algorithm was the first to use a formal definition of representation. This algorithm was subsequently enhanced by Bernstein and Beeri [5,8] who improved its running time.

The first generalization of FDs was the concept of first order hierarchical decomposition by Delobel [18] and Delobel and Leonard [21]. The related concept of MVG was introduced by Pagin [23] and Zaniolo [41], and 4NF was introduced by Pagin
[23]. Completeness of inference rules for MVDs is treated by Beeri, Fagin and Howard [16] and Mendelzon [29], and algorithmic questions about MVDs by Beeri [4]. Recently, attention has been directed to the representation principle by the work of Aho, Beeri, and Ullman [1] and Rissanen [31]. These references are a mere sketch of the history of normalization theory; a more complete bibliography follows.

A variety of important results appear in these papers, but the lack of uniform definitions has obscured the calculations among many works. We hope the paper will clear up some of the confusion by comparing the major definitions and outlining a general framework in which all can be embedded.

Our main theme is that schema design is directed by the three principles of representation, separation, and minimal redundancy. A goal of research in schema design is to develop a design methodology that satisfies these three principles. Specific formulations of the principles depend upon the type of constraints involved, so a thorough understanding of the formal properties of $f$ and $mvd$ is a prerequisite for achieving this goal.

Many questions still remain unanswered. We list four important areas where more work is needed:

1. Other dependency structures--An MVD can hold in a projection of a relation, although it does not hold in the entire relation [19,25]. These embedded $mvd$s (abbr. EMVD) may appear when decomposing a relation scheme into smaller schemes. While some inference rules for EMVDs have appeared, a complete set is not currently known [19].

MVDs characterize lossless joins between two relations. Dependency structures that characterize lossless joins among $n$ relations have recently been suggested, and should be integrated into the theory [30,32]. In addition, the concept of representation (particularly Rep2, Rep4) has only been developed for FDs. Representation questions about MVDs and other dependency structures are open.

2. Semantic operations on dependencies--Dependency structures can be used to guide correct retrievals given only minimal logical access path information [11,34]. However, the influence of dependency structures on data operations and the constraints that hold in a relation constructed by operations are only known for special cases.

3. Universal relation assumption--This assumption simplifies many theoretical problems but apparently does not hold in practice. It should either be abandoned or adapted for practical situations in some way.

4. Design tools--Mechanical procedures must be developed to assist the database designer. A schema synthesis algorithm that takes FDo and MVDs as input could be one such design aid. Mechanical mappings from high level data descriptions (e.g., [36]) into dependency structures are also needed. The true test of the theory is demonstrating its effectiveness in solving day to day database design problems. On this metric the theory will live or die.

REFERENCES


