Functional dependencies with `null` markers

Antonio Badia[1]* and Daniel Lemire[2]

[1]CECS Department, University of Louisville, Louisville KY 40292, USA
[2]LICEF, Université du Québec, 5800 Saint-Denis, Montreal, QC, H2S 3L5 Canada
Email: antonio.badia@louisville.edu

Functional dependencies are an integral part of database design. However, they are only defined when we exclude `null` markers. Yet we commonly use `null` markers in practice. To bridge this gap between theory and practice, researchers have proposed definitions of functional dependencies over relations with `null` markers. Though sound, these definitions lack some qualities that we find desirable. For example, some fail to satisfy Armstrong's axioms—while these axioms are part of the foundation of common database methodologies. We propose a set of properties that any extension of functional dependencies over relations with `null` markers should possess. We then propose two new extensions having these properties. These extensions attempt to allow `null` markers where they make sense to practitioners. They both support Armstrong's axioms and provide realizable `null` markers: at any time, some or all of the `null` markers can be replaced by actual values without causing an anomaly. Our proposals may improve database designs.

Keywords: Functional Dependencies; Database Design; Missing Information

1. INTRODUCTION

Functional dependencies (hereafter, FDs) are the basis of good relational database design. Database designers use FDs to define and enforce consistency; for example, if each user account has only one corresponding primary email address, we say that the user account determines the email address (user-account → email). Though database designers may not always work directly with FDs, almost all of them work with normal forms which exist to support FDs.

Meanwhile, relational databases use `null` markers to cope with incomplete information. Indeed, Codd introduced `null` markers in the relational model along with a 3-value logic: a comparison between a value and a `null` marker has an unknown truth value [1]. A `null` marker can indicate anything from an applicable but unknown value to a non-applicable attribute; it may even indicate that we have no information [2, 3, 4]. We refer the reader to Libkin [5] for an early survey on incomplete information within databases.

Unfortunately, the concept of functional dependence, even though central to relational database design, is not part of the SQL standard. Hence, FDs are not directly supported by relational databases. Admittedly, given the procedural extensions of the SQL standard, SQL is computationally complete and therefore can enforce FDs using checks, assertions or triggers. However, the problem is more fundamental: FDs are not defined in the presence of `null` markers. Thus, there is no clear semantics to enforce when we have FDs and `null` markers.

We believe that the fact that FDs are not defined in the presence of `null` markers undermines the role of FDs in database design [6]. While there are proposed definitions of FDs in relations with `null` markers, such proposals may not be practical.

Our work is organized as follows. In the next section, we argue for a minimal set of desirable properties that FDs in relations with `null` markers should have to be of practical use. After briefly introducing our formal notation in § 3, we review two existing extensions of FDs to `null` markers in § 4: weak and strong FDs. We show that neither extension is satisfactory in our context, as they do not have the desired properties. We introduce two novel extensions in § 5: literal and super-reflexive functional dependencies, and show that they that possess our properties. In § 6 we compare our extensions to weak and strong FDs, in order to give the reader a context for interpretation. Next in § 7 we address the issue of whether the newly proposed concepts can be efficiently supported. In § 8, we show how to extend logical database design to include `null`

TABLE 1: Example relation with attributes professor, chair and department

professor	chair	department
Joe	`null`	Mathematics
Joe	Jill	Computer Science
Bill	Arthur	Mathematics

TABLE 2: Example relation with attributes SSN, income and taxation

SSN	income	taxation
1112233	`null`	15%
1112233	`null`	25%

markers using one of the proposed extensions. Finally, we close in § 9 with some comments and a discussion of further research.

2. DESIRABLE PROPERTIES

The SQL standard allows `null` markers, but does not allude to FDs [7]. One could point out that the concept of key is explicitly present in the SQL standard; given that the concepts of key and FD are related, it would seem that we support FDs by enforcing the key constraint [8]. However, when we enforce FDs by key constraints, we assume that all the tables in the database have attained Boyce-Codd normal form (BCNF). Yet such a normal form excludes the commonly used `null` markers.

Codd was well aware of the perceived problems with `null` markers. Yet he was unconcerned by our inability to enforce FDs in the presence of `null` markers. He believed that only when the `null` markers where replaced by actual values would concepts such as keys, normalization and FDs be applicable:

> It should be clear that, because nulls (or, as they are now called, marks) are NOT database values, the rules of functional dependence—and of multi-valued dependence—do not apply to them. (E. F. Codd [1])

However, consider the relation in Table 1 subject to the FDs professor → chair and department → chair. Given that Jill and Arthur are distinct people, it is not possible to replace the `null` marker by an actual value. Going back to Codd's vision, we have that keys, normalization and FDs are never going to be applicable to all tuples of such relation. This might reasonably be considered anomalous.

To avoid such problems, we establish as a first goal that if a FD is enforced in a relation with `null` markers, it should always be possible to replace some or all of the `null` markers by actual values without violating the FD (**G1**). In such a case, we say that such FDs are realizable under null markers. Moreover, and since we consider sets of FDs for design (as opposed to single FDs in isolation), we posit that FDs should be defined so that if a set of FDs is enforced in a relation with `null` markers, it should be possible to replace some or all of the `null` markers without violating any of the FDs in the set (**strong G1**).

When defining FDs in the presence of `null` markers, we are also interested in Armstrong's three axioms, especially transitivity:

1. Reflexivity: If $Y \subseteq X$, then $X \to Y$.
2. Augmentation: we have that $X \to Y$ implies $XZ \to YZ$.
3. Transitivity: If $X \to Y$ and $Y \to Z$, then $X \to Z$.

E.g., we might say that your social security number (SSN) determines your income (SSN → Income), and that your income determines your tax bracket (Income → Taxation). If transitivity holds, then your SSN determines your tax bracket (SSN → Taxation).

Codd's interpretation, that FD do not apply when there are `null` markers fails to enforce transitivity in the following sense: given the FD SSN → Income and Income → Taxation, both tuples in Table 2 are allowable, even though one would expect not to see such data in the database. It implies that SSN → Taxation does not hold (whereas it should under transitivity) even though there is no `null` marker over attributes SSN and Taxation.

Codd would no doubt reply that there is no violation of transitivity since FDs do not apply in the presence `null` markers. But we wish to consider `null` markers as an integral part of the database.

Failing to enforce Armstrong's axioms has significant consequences. For one thing, without these axioms, normalization is no longer sufficient to enforce FDs. In practical terms, any redefinition of the FDs that fails to satisfy Armstrong's axioms cannot be enforced through normalization. Indeed, given a database D and a set of FDs $\mathcal{F}$ on D, before we can use $\mathcal{F}$ to determine normal forms for the relations in D, we need to make sure that $\mathcal{F}$ is in minimal or canonical form [9]. But minimizing $\mathcal{F}$ depends crucially on FDs respecting transitivity, as covered in standard database textbooks.

We might be willing to forgo normalization and enforce FDs through other means. In such a case, it might seem like Armstrong's axioms are no longer required. For example, we might think that it is possible to build a database design without assuming that FDs are transitive. However, such watered-down FDs might be impractical for other reasons. The first problem that we encounter is that standard database design methodologies, like the entity-relationship model, implicitly assume Armstrong's axioms and transitivity in particular [10].

We believe that database designers would have a hard time coping with the lack of transitivity (see, for instance, the example of Table 2, where intuitively one would expect to see, for the same SSN, the same Taxation), and hence we require that Armstrong's axioms hold, even though such a requirement could be seen as too strong. Of course, we could help designers with additional tools and methodologies [11] to compensate for the added constraint. Nevertheless, everything else being equal, we view as desirable that a new definition of FDs in the presence of null markers should respect Armstrong's axioms (**G2**), as well as enforce realizable null markers.

Of course, our goals so far can be accomplished by being very restrictive on the use of null markers. One might even take the stance that null markers should always be forbidden [12]. But we also want to allow common uses seen in production-quality applications. For example, we have observed that many database schemas allow null markers on attributes that do not determine other attributes. Thus, it is another objective (**G3**) of this research to allow null markers when this can be done without violating other goals. At a minimum, we should allow null markers without having to come up with contrived examples.

Finally, any enforcement of FDs is going to be considered, from the point of view of transaction or query processing, as overhead—just like enforcing primary and foreign key constraints. Thus, any definition should be computationally inexpensive (**G4**). In practice, this means that we exclude elegant but challenging models such as v-tables [13]. For example, we should be able to determine whether the FD $X \rightarrow Y$ holds by considering only the attributes in X and Y.

To summarize, we seek to extend FDs to include null markers in such a way that:

1. **G1**: FDs enforce realizable null markers. Further, this should hold for sets of FDs (**strong G1**).
2. **G2**: Armstrong's axioms are satisfied.
3. **G3**: FDs should not restrict the use of null markers unnecessarily.
4. **G4**: Enforcing FDs should be computationally practical.

3. BASIC CONCEPTS

Let a relation r be as in SQL: a finite multiset of tuples over a given schema sch(r), with the caveat that a tuple may contain null markers. Two tuples are considered duplicates if all non-null attributes are equal and any null marker in one tuple is matched by a null marker in the other tuple; otherwise the tuples are distinct.

We assume that there is an infinite set V of values, from where all the constants in any relation are drawn. These values have a relation '=' defined on them, which is reflexive, symmetric and transitive: given any $x, y, z \in V$, we have than $x = x$, $(x = y) \Rightarrow (y = x)$ and $(x = y, y = z) \Rightarrow x = z$. Hence, the relation "=" is an equivalence relation.

Given an attribute A in the schema of relation r and a tuple t, we use $t[A]$ as is customary, to denote the value of t for A. This is extended to sets of attributes $X \subseteq \text{sch}(r)$ as usual. We then say, for two tuples t, t', that $t[A] = t'[A]$ is true if both $t[A]$ and $t'[A]$ are equal non-null values; false if both $t[A]$ and $t'[A]$ are values, but they are different; and unknown otherwise (i.e., if either one of $t[A]$ or $t'[A]$, possibly both, are null markers). Again, this is extended to sets of attributes $X \subseteq \text{sch}(r)$ in the usual way: for two tuples t, t', $t[X] = t'[X]$ is true if $t[A] = t'[A]$ is true for every $A \in X$; false if $t[A] = t'[A]$ is false for some $A \in X$, and unknown otherwise. As a shorthand, we write $t = t'$ for $t[\text{sch}(r)] = t'[\text{sch}(r)]$. We write $\pi_X(r)$ for the projection of all tuples in r on X: starting from $\{t[X] | t \in r\}$, all duplicates are removed.

Let r be a fixed relation. If we disallow null markers in r, then a FD $X \rightarrow Y$ is satisfied if $t[Y] = t'[Y]$ when two tuples t, t' are such that $t[X] = t'[X]$. In this context (where null markers are forbidden), FDs satisfy Armstrong's axioms. We can also formalize the concept of key if there is no null marker in r. A set of attributes K is a superkey iff $K \rightarrow A$ holds for any attribute $A \in \text{sch}(r)$; and a key if it is a minimal superkey. Primary keys in SQL are keys with attributes where null markers are forbidden; SQL allows null markers in other keys.

We say that an attribute A is non-null in a relation r if for all $t \in r$ we have that $t[A]$ is non-null. Given a set of attributes X, we say that the null marker appearing in a tuple t at attribute A ($t[A] = \texttt{null}$) is in X if $A \in X$.

4. STRONG AND WEAK FUNCTIONAL DEPENDENCIES

Levene and Loizou [14] propose one of the few extensions of FDs over null markers. We formalize their definitions as follows. A valuation φ for a relation r assigns to each null marker in a tuple of r a value from V—each null marker may receive a different value—while leaving non-null values from V unchanged. Given a relation r, each $\varphi(r)$ is called a possible world for r. The semantics of FDs with null markers, as defined by Levene and Loizou, follows the idea of modal logic [15]: we have two distinct readings, depending on whether the FD holds in some or all possible worlds.

DEFINITION 4.1. *A FD F holds weakly in relation r iff F holds in a possible world for r—i.e., there exists a valuation φ such that F holds in $\varphi(r)$. F is called a weak FD (WFD).*

DEFINITION 4.2. *A FD F holds strongly in relation r iff F holds in all possible worlds for r—i.e., for every valuation φ for r, F holds in $\varphi(r)$. F is called a strong FD (SFD).*

Looking back at Table 1, consider the following FDs: chair $\rightarrow$ professor and professor $\rightarrow$ chair. Both hold weakly, while neither holds strongly (see Table 3).

TABLE 3: Various FDs applied to the relation of Table 1 and whether they hold strongly, weakly, super-reflexively or literally

	SFD	WFD	SRFD	LFD
chair $\rightarrow$ professor	no	yes	no	yes
professor $\rightarrow$ chair	no	yes	yes	no

A strong FD is always also a weak FD. Strong FDs satisfy Armstrong's axioms while weak FDs do not. When a FD holds strongly, we can replace any `null` marker by a value, and the FD still holds. In fact, strong FDs enforce realizable `null` markers.

The Levene and Loizou model has a substantial formal appeal, but it also has some drawbacks from a pragmatic point of view:

- Weak FDs might be too weak. Even if each FD in a set $\mathcal{F}$ of FDs holds weakly, there might be no single possible world $\varphi(r)$ where all of the FDs in the set $\mathcal{F}$ hold. See Table 1 for a counterexample: both FDs (professor $\rightarrow$ chair and chair $\rightarrow$ professor) hold weakly, but there is no possible world where they both hold. That is, we can substitute some value for the `null` marker to satisfy the first FD, and substitute another value to satisfy the second FD, but no single value makes both FDs true at the same time (thus, $\mathcal{F}$ does not have property **strong G1**). This is true even when each attribute appears only once on the right-hand-side of a FD: given the schema A, B, C and the FDs $A \rightarrow B$ and $B \rightarrow C$, the set of tuples $(a, \texttt{null}, b)$ and $(a, \texttt{null}, c)$ satisfy both FD weakly, but there is no world where they both hold. This last example also illustrates that weak FDs are not transitive: $A \rightarrow B$ and $B \rightarrow C$ can hold weakly while $A \rightarrow C$ may not. That is, weak FDs do not satisfy Armstrong's axioms (thus failing **G2**).

  We could fix weak FDs to ensure that they enforce, for example, realizable `null` markers by requiring that there exists a valuation corresponding to the set of all FDs. However, this may prove computationally challenging (hence failing **G4**).

- Strong FDs might be too strong (thus failing **G3**). Indeed, it seems unnecessary to always require that FDs should hold in all possible worlds. For example, consider the schema A, B, C and the FDs $A \rightarrow B$ and $B \rightarrow C$, and the set of tuples $(a, b, \texttt{null})$ and $(c, b, \texttt{null})$. Though it appears like a reasonable relation, the FD $B \rightarrow C$ fails to hold strongly.

In some sense, the weak and strong FDs are two extremes, whereas the right solution might be somewhere in between. Indeed, any form of FD that supports realizable `null` markers (**G1**) on a per FD basis, is going to be equivalent to or stronger than weak FDs. Meanwhile, strong FDs have the properties we seek, except that they are too restrictive (**G3**).

5. LITERAL AND SUPER-REFLEXIVE FUNCTIONAL DEPENDENCIES

We propose two alternative definitions of the concept of FD in the presence of `null` markers. The first one is a natural extension of the 3-value logic proposed by Codd and used by SQL.

Definition 5.1. *A FD $X \rightarrow Y$ holds super-reflexively if, for any two tuples t, t', when $t[X] = t'[X]$ is not false (i.e., it is either true or unknown), then $t[Y] = t'[Y]$ is also not false. We say $X \rightarrow Y$ is a super-reflexive FD (SRFD).*

In this first definition, the `null` marker is effectively equal to any other value (i.e., `null` $= a$ is treated as true for any value a), hence the term super-reflexive (SR).

As an illustration, consider Table 1. We have that professor $\rightarrow$ chair hold super-reflexively whereas chair $\rightarrow$ professor does not (see Table 3).

Our second definition is reminiscent of how languages such as JavaScript handle `null` markers. As a first approximation, they consider `null` to be effectively a regular value, with `null` = `null` is always true (i.e., "=" remains a reflexive relation[3]), but `null` $= a$ always false for a non-`null`.

We say that $t[A]$ and $t'[A]$ are identical if both contain `null` or both contain the same value; this is also extended to set of attributes X and to whole tuples as usual: $t[X]$ is identical to $t'[X]$ if and only if $t[A]$ and $t'[A]$ are identical for all $A \in X$.

Definition 5.2. *A FD $X \rightarrow Y$ holds literally if, for any two tuples t, t', when $t[X]$ and $t'[X]$ are identical then $t[Y]$ and $t'[Y]$ are also identical. We say $X \rightarrow Y$ is a literal FD (LFD).*

Consider again Table 1. In contrast with the super-reflexive case, we have that the FDs chair $\rightarrow$ professor holds literally whereas professor $\rightarrow$ chair does not. (See again Table 3.)

There are alternative definitions that we could have used. For example, we could have defined FDs $X \rightarrow Y$ to hold if when $t[X] = t'[X]$ is true (as per Codd's 3-value logic) then $t[Y] = t'[Y]$ must be true. Or, we could have defined an FD $X \rightarrow Y$ to hold if when $t[X] = t'[X]$ is not false then $t[Y] = t'[Y]$ must be true; or if when $t[X] = t'[X]$ is true then $t[Y] = t'[Y]$ must be not false. However, these alternative definitions are unsatisfying: they fail to satisfy Armstrong's axioms (**G2**), or do not allow `null` markers where they are commonly used (**G4**). By contrast, Definitions 5.1 and 5.2 have the properties that we required in § 2 starting with

[3]In fact, '=' remains an equivalence relation, something that does not hold for Codd's 3-value logic if you consider `null` markers to be part of the value domain.

Armstrong's axioms (which follows by inspection). For example, regarding transitivity, consider the two FDs $X \to Y$ and $Y \to Z$ that hold super-reflexively (resp. literally). Given two tuples t, t' such that $t[X] = t'[X]$ is not false (resp. $t[X]$ and $t'[X]$ are identical), then $t[Y] = t'[Y]$ is not false (resp. $t[Y]$ and $t'[Y]$ are identical) which implies that $t[Z] = t'[Z]$ is not false (resp. $t[Z]$ and $t'[Z]$ are identical). Thus we have that $X \to Z$, proving transitivity.

LEMMA 5.1. *Super-reflexive and literal FDs respect Armstrong's axioms (***G2***).*

Before we proceed to establish other properties, we need a technical result that makes other proofs easier. First, we can verify that all FDs can be decomposed into FDs where the right-hand-side contains a single attribute. For example, the FD professor $\to$ {chair, department} is equivalent to the following two FDs:

- professor $\to$ chair, and
- professor $\to$ department.

LEMMA 5.2. *We have that $X \to Y$ for $Y = \{A_1, \ldots A_n\}$ is equivalent to $(X - \{A_1\} \to \{A_1\}) \wedge \cdots \wedge (X - \{A_n\} \to \{A_n\})$ whether we consider weak, strong, literal or super-reflexive FDs.*

The proof of this lemma follows by inspection. Hence, it is enough to consider FDs $X \to Y$ where Y is a singleton disjoint from X.

First, we must show that SRFDs and LFDs satisfy condition **G1**: `null` markers are realizable. We begin with SRFDs. We show that given the SRFD $X \to Y$, all `null` markers in $X \cup Y$ are realizable with respect to $X \to Y$. To illustrate this result, consider Table 1 where professor $\to$ chair holds super-reflexively: we can substitute Jill for the `null` marker without violating the FD.

LEMMA 5.3. *Consider a SRFD $X \to Y$ in a relation such that X and Y are disjoint and Y is a singleton.*

1. *We can replace any `null` marker in X by* ***any*** *actual value without violating the SRFD $X \to Y$.*
2. *We can replace any `null` marker in r in Y by* ***at least one*** *actual value without violating the SRFD $X \to Y$, assuming that attributes in X are non-`null`.*

Proof. Assume that the SRFD is initially satisfied over r.

(1) Suppose we replace a `null` marker in attribute $B \in X$. Regarding the SRFD $X \to Y$, the following might happen for two tuples t, t':

- if $t[X] = t'[X]$ was false, then it is still false: this cannot affect the SRFD;
- if $t[X] = t'[X]$ was true, then it is still true: this cannot affect the SRFD;
- if $t[X] = t'[X]$ was unknown, then it might become true or false. Because of the SRFD $X \to Y$, and because $t[X] = t'[X]$ was not false, we have that $t[Y] = t'[Y]$ is not false. Therefore, if the tuples t, t' satisfied the SRFD before the update, they must satisfy it after the update as well.

Because all pairs of tuples satisfy the conditions of SRFD after the update, the SRFD still hold, proving the first part of the result.

(2) Assume that attributes in X are non-`null`. We want to show that we can replace any `null` marker in Y by an actual value. Pick a tuple t in r where $t[Y]$ contains a `null` marker. Consider the set τ of t' such that $t'[X] = t[X]$ is not false. (Because attributes in X are non-`null`, we have that "$t'[X] = t[X]$ is not false" is equivalent to "$t'[X] = t[X]$ is true".) We have that the projection of τ over Y contains at most one actual value. (Suppose it does not, then you can find tuples t'' and t''' in τ such that $t''[X] = t'''[X]$ is not false but $t''[Y] = t'''[Y]$ is false.) If there is one actual value, set $t[Y]$ to this value; if not, pick a value at random. This modification clearly does not violate the SRFD $X \to Y$ but it eliminates one `null` marker. □

To see why Lemma 5.3 implies that SRFDs satisfy **G1**, consider any SRFD $X \to Y$ over a given relation. We can substitute actual values for any `null` marker in an attribute of X by the first part of the lemma. As a second step, since attributes in X have become non-`null`, we can substitute actual values for any `null` marker in Y.

As for LFDs, we are going to prove something stronger: that they support **strong G1**.

LEMMA 5.4. *LFDs strongly enforces realizable `null` markers (***strong G1***).*

Proof. To prove **G1**, it suffices to replace all the `null` markers with a single $v \in V$ not already in the relation. By inspection, this extends to sets of FDs, so we get also **strong G1** with this method. □

We have that both LFDs and SRFDs enforce realizable `null` markers. That is, given a relation with a set of FDs, we can always replace `null` markers with some actual values without violating the FDs. In fact, if we add an extra constraint on SRFDs, they both strongly enforce realizable `null` markers (in the sense of **strong G1**). In this context, we adopt the practical convention that some attributes are allowed to contain `null` markers while others may not: this is motivated by the SQL standard. Given a set of FDs $\mathcal{F}$, we say that an attribute B determines another attribute A under $\mathcal{F}$ if there is a FD $B \in X \to Y \ni A$ in the transitive closure of $\mathcal{F}$. Naturally, this property is transitive: if A determines B and B determines C then A determines C. (By convention, we omit loops in $\mathcal{F}$: $X \to X$.)

Condition 1RHS. *Consider a set of FDs $\mathcal{F}$ over a relation. Consider any attribute A allowed to contain*

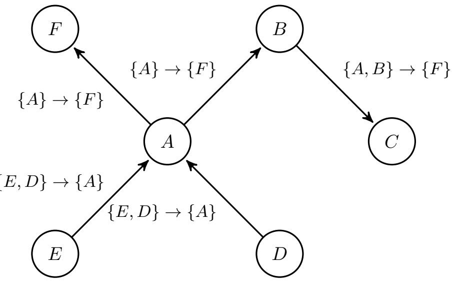

FIGURE 1: Illustration used by the proof of Lemma 5.5 for the set of FDs $\{\{E, D\} \to \{A\}, \{A\} \to \{F\}, \{A, B\} \to \{F\}\}$.

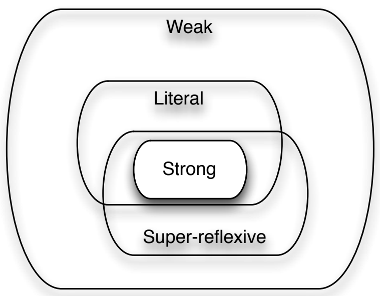

FIGURE 2: Venn diagram illustrating Lemma 6.1.

`null` markers. Then A must appear on the right-hand-side of at most one FD in the set of FDs $\mathcal{F}$ of the relation. Moreover, given two distinct attributes allowed to contain `null` marker, A and B, if A determines B, B cannot determine A.

We stress that Condition 1RHS only applies to attributes allowed to contain `null` markers: no constraint is required on other attributes.

Fig. 1 gives an example of a set of FDs satisfying the condition 1RHS even if all attributes are allowed to contain `null` markers: $\{E, D\} \to \{A\}, \{A\} \to \{F\}, \{A, B\} \to \{F\}$. However, if we replaced the single FD $\{E, D\} \to \{A\}$ by two FDs such as $\{E\} \to \{A\}$ and $\{D\} \to \{A\}$, we would need to add the requirement that A is non-`null` to satisfy 1RHS. Similarly, if we added the FD $\{F\} \to \{A\}$ in addition to the existing FD $\{A\} \to \{F\}$, we would need to require that both A and F are non-`null` since F would determine A while A determines F.

LEMMA 5.5. *SRFDs strongly enforces realizable `null` markers whenever the 1RHS condition is satisfied (**strong G1**).*

Proof. Assume without loss of generality that all SRFDs in the set are of the form $X \to Y$ where Y is a singleton and X, Y are disjoint.

Construct a graph where each attribute allowed to contain `null` markers is a node, and there is an edge between two attributes A, B if and only if A determines B. We illustrate such a graph in Fig. 1 where, for simplicity, we omitted some of the edges that are implied by transitivity.

Temporarily assume that the graph is not empty. Because of condition 1RHS, the graph must be cycle-free and, therefore, some of the nodes must have a zero in-degree (e.g., E and D in Fig. 1). Call this set of nodes/attributes $\mathcal{A}_0$.

As per Lemma 5.3, we can substitute actual values for any `null` marker they contain. Indeed, consider such an attribute $A \in \mathcal{A}_0$. This attribute appears on the right-hand-side of at most one FD in the set (call it F_A), however A may appear on the left-hand-side of several FDs. These FDs are not a concern: replacing a `null` marker in attribute A may never violate a super-reflexive FD as per the first part of Lemma 5.3. Meanwhile, because F_A is such that no attribute on its left-hand-side contains a `null` marker, then the second part of Lemma 5.3 tells us that the `null` markers of A are realizable.

After substituting actual values for any `null` marker in the attributes of $\mathcal{A}_0$, remove these nodes from the graph. There must again be nodes with zero in-degree (e.g., node A in Fig. 1) or the graph is empty. Repeat the process until the graph is empty.

Attributes that either do not appear as part of any FD, or that are not allowed to contain `null` markers, are not a concern. □

6. COMPARING FUNCTIONAL DEPENDENCIES

We are now in a position to characterize LFDs and SRFDs properly. We can relate LFDs and SRFDs to each other, and with Levene and Loizou's definitions. All are conservative extensions of the classical concept: in a table without any `null`, they coincide with classical FDs. However, in general, LFDs and SRFDs are incomparable as illustrated by Table 3.

We show that strong ⇒ super-reflexive ⇒ weak and strong ⇒ literal ⇒ weak (see Fig. 2). That is, both LFDs and SRFDs are stronger than weak FDs, whereas strong FDs are stronger than both LFDs and SRFDs.

LEMMA 6.1. *The following holds:*

1. *If the FD $X \to Y$ holds literally, then it holds weakly.*
2. *If the FD $X \to Y$ holds super-reflexively, then it holds weakly.*
3. *If the FD $X \to Y$ holds strongly then it must hold literally and super-reflexively.*

Proof. Assume without loss of generality that X and Y are disjoint and that Y is a singleton.

(1) If the FD $X \to Y$ holds literally then we can replace any `null` marker by any one $v \in V$ not already present in the relation, and the FD still holds by

inspection. This shows that literal FDs are stronger than weak FDs.

(2) By Lemma 5.3, we can construct a valuation such that the super-reflexive FD is valid in the conventional sense. Because of the existence of the valuation, we have that $X \to Y$ holds weakly.

(3) We first prove that strong implies super-reflexive. Suppose that $X \to Y$ holds strongly. Consider two tuples t, t'. If $t[X] = t'[X]$ is not false (in Codd's 3-value logic), then in some possible world, $t[X] = t'[X]$ must hold. In such a possible world $t[Y] = t'[Y]$ must hold which implies that $t[Y] = t'[Y]$ must be not false (in Codd's 3-value logic). This prove that a strong FD implies a super-reflexive FD.

We prove that strong implies literal. Suppose that the FD $X \to Y$ holds strongly and that $t[X]$ and $t'[X]$ are identical. (We assume that X and Y are disjoint and Y is a singleton as previously stated.) There are some worlds where $t[X] = t'[X]$ is true. If $t[Y]$ and $t'[Y]$ are not identical, then in at least one of these worlds, they would differ. That is, if one has a `null` marker in $t[Y]$ and a value a in $t'[Y]$ (or vice versa) we can replace the `null` marker by a value that differs from a. Hence, we must have that $t[Y]$ and $t'[Y]$ are identical given that $t[X]$ and $t'[X]$ are identical. Therefore strong FDs imply literal FDs. This completes the proof. □

We can verify that LFDs and SRFDs are strictly weaker than SFD. Recall that given the schema A, B, C and the FDs $A \to B$ and $B \to C$, the set of tuples $(a, b, \texttt{null})$ and $(c, b, \texttt{null})$ violates the strong FD $B \to C$. However, it satisfies the FD $A \to B$ and $B \to C$ literally and super-reflexively. The fact that LFDs and SRFDs allow this use of `null` markers support our claim that they do not unnecessarily forbid `null` markers where they might make sense (**G3**).

7. COMPUTATIONAL EFFICIENCY

The remaining question is the efficient implementation of our proposed concepts (**G4**). Here we show that both LFDs and SRFDs can be enforced with a cost similar to enforcing regular FDs.

Since SQL does not enforce FDs directly, we compare with a closely associated property, the cost of enforcing the key constraint in a relation. This assumes that the database is in an appropriate normal form such that enforcing keys is equivalent to enforcing FDs. Given relation r, if $X \subseteq r$ is declared as the primary key for r, any insertion $t \in r$ is checked to make sure that X remains a key. In practice, relational database systems forbid the occurrence of two distinct tuples t, t' agreeing on X ($t[X] = t'[X]$ must be false). This can be achieved by creating an index (traditionally, a tree-based index) on X: on inserting t, we search for any $t' \in r$ with $t[X] = t'[X]$ (note that `null` markers are forbidden in a primary key). Call the set of tuples resulting from the search S. When inserting, if $S = \emptyset$, the insertion can proceed; else, for each $t' \in S$ we check whether t' is identical to t. If this is so, the insertion can proceed; else it is forbidden. Updates are handled in a similar manner. The complexity of this procedure is considered $\mathcal{O}(\log \mid r \mid)$ when using a tree-based index, since it is expected that $\mid S \mid \leq 1$. (Of course, we can get an expected constant time complexity by using a hash-based index.)

We can also enforce FDs directly, whether they are literal or super-reflexive. Given relation r and arbitrary FD $X \to Y$ on it, we create an index on X—for concreteness, we assume a B^*-tree index since these are available on almost any database system.

To enforce FDs with an index, it suffices to check whether the insertion of a new tuple t is allowed. Indeed, an update can be viewed as a deletion followed by an insertion, and deletions cannot violate a FD.

Given an index, enforcing LFDs is not difficult. We build a single index on X by considering a concatenation of all attributes—lexicographic order is followed, with `null` markers in individual attributes kept together at the beginning or end of their positions. That is, in an index for attributes ABC, tuple $(a_1, b_1, \texttt{null})$ would go before or after all tuples (a_1, b_1, c), for c any value of C; tuple $(a_1, \texttt{null}, b_1)$ would go before or after any tuples (a_1, c, b_1) for c any value of C; and so on. This can be achieved by considering `null` markers as either strictly larger than any other value, or strictly smaller. Suppose that we insert a new tuple t. We can find in time $\mathcal{O}(\mid S \mid + \log \mid r \mid)$ the set S of all tuples t' such that $t[X]$ and $t'[X]$ are identical:

$$S = \bigcap_{A \in X} \{t' \in r | t'[A] = t[A] \ \vee \ t'[A] \text{ and } t[A] \text{ are } \texttt{null}\}$$

We can then check whether $t[Y]$ and $t'[Y]$ are identical for all t' in linear time $\mathcal{O}(\mid S \mid)$. Note that we consider the cardinality of the set X ($\mid X \mid$) to be a small constant.

We can also enforce $X \to Y$ as an SRFD efficiently, though the total cost is probably larger. Index each one of the attribute $A \in X$ using, as before, the convention that y considering `null` markers as either strictly larger than any other value, or strictly smaller. Given t, first we check whether $t[Y]$ contains a `null` marker (assume Y is a singleton disjoint from X), in which case no work is needed. Otherwise, we find all t' such that "$t[X] = t'[X]$ is not false" by computing

$$S = \bigcap_{A \in X | t[A] \text{ not } \texttt{null}} \{t' \in r | t'[A] = t[A] \vee t'[A] \text{ is } \texttt{null}\}$$

with the convention that the result is r if $t[X]$ only contains `null` markers. Computing the intersection S between several sets $S_1, S_2, \ldots, S_{|X|}$ requires only complexity $\mathcal{O}(\sum_{i=1}^{|X|} |S_i| + \mid S \mid)$ or better [16]. Each set S_i can be generated in time $\mathcal{O}(\mid S_i \mid + \log \mid r \mid)$ for an overall complexity of $\mathcal{O}(\sum_{i=1}^{|X|} |S_i| + |S| + \log |r|)$. We then consider $t'[Y]$ for all $t' \in S$: if there is an actual value, check that $t[Y]$ is equal to $t'[Y]$. That is, we return

$$\bigwedge_{t' \in S} t'[Y] = t[Y] \ \vee \ t'[Y] \text{ is } \texttt{null}.$$

This can be computed in time $\mathcal{O}(|S|)$.

To sum up, enforcing a LFD or SRFD under an update or insertion can be done in time

- $\mathcal{O}(|S| + \log |r|)$ or
- $\mathcal{O}(\sum_{i=1}^{|X|} |S_i| + |S| + \log |r|)$.

Our sketched implementations only require tree lookups and set intersections, both of which are well supported by all database systems. This is sufficient to conclude that they are computationally practical (**G4**).

8. EXTENDING LOGICAL DATABASE DESIGN TO INCLUDE NULL MARKERS

As we discussed in § 5, it is possible to enforce FDs with `null` markers using either SRFDs or LFDs—without any particular effort on the part of the database designer. However, we commonly enforce FDs using logical database design. That is, we decompose relations into normal forms and identify keys.

We would like to remain as close as possible to the spirit of SQL. Thus, we ask whether we can use logical design with SRFDs and LFDs. This would allow an extension of logical database design to include `null` markers. Unfortunately, while both LFDs and SRFDs fulfill all our desiderata (**G1** to **G4**), SRFDs are not compatible with conventional logical design. But we have better luck with LFDs.

Recall that in a relation r_1, a set of attributes $X \subseteq \text{sch}(r_1)$ is a key if the (standard) FD $X \to \text{sch}(r_1)$ holds and if X is minimal. Moreover, in relation r_2, a set of attributes $Y \subseteq \text{sch}(r_2)$ is a foreign key for r_1 if $\pi_Y(r_2) \subseteq \pi_X(r_1)$ for all extensions of r_1 and r_2.

A join $r = r_1 \bowtie_{X=Y} r_2$ is a new relation made by combining all tuples $t_1 \in r_1$ and all tuples $t_2 \in r_2$ such that $t_1[X] = t_2[Y]$ into a new tuple t equal to t_1 on $\text{sch}(r_1)$ and equal to t_2 on $\text{sch}(r_2) - Y$. A join $r = r_1 \bowtie_{X=Y} r_2$ is lossless when $\pi_{\text{sch}(r_1)}(r) = r_1$ and $\pi_{\text{sch}(r_2)}(r) = r_2$.

We extend the concepts of keys and joins in the context of LFDs as follows.

- We say that $X \subseteq \text{sch}(r)$ is a literal superkey for r if $X \to Y$ holds literally for any $Y \subseteq \text{sch}(r)$ and a literal key iff it is a minimal literal superkey. A literal foreign key is an integrity constraint between two relations: a set of attributes X in relation r_1 must match a set of attributes X in a relation r_2 such that for every tuple t in r_1, there must be a tuple t' in r_2 such that $t[X]$ and $t'[X]$ are identical and such that X contains a literal key in r_2.
- As in SQL, each literal foreign key constraint supports a corresponding join. The literal join of r_1 and r_2 on X, a foreign key in r_1, noted $\widehat{\bowtie}$ is defined as follows: given any two tuples t_1, t_2 from r_1, r_2 such that $t_1[X]$ and $t_2[X]$ are identical, we generate the tuple t such that $t[A] = t_1[A]$ for all $A \in \text{sch}(r_1)$ and $t[A] = t_2[A]$ for all $A \in \text{sch}(r_2) - \text{sch}(r1)$.

With these definitions, we have that lossless joins are supported, even with `null` markers. Formally, given relation r with $\text{sch}(r) = Z \cup W$ and such that $Z \cap W \to W$ holds literally, $\widehat{\bowtie}$ is a lossless join: $r = \pi_Z(r) \widehat{\bowtie} \pi_W(r)$.

Proposition 8.1. *(Lossless join) If* $\text{sch}(r) = Z \cup W$ *and* $Z \cap W \to W$ *holds literally then* $r = \pi_Z(r) \widehat{\bowtie} \pi_W(r)$.

Proof. For the purpose of the literal join, the `null` marker can be treated like any other value. That is, the set of values V extended with the `null` marker is effectively reflexive, symmetric and transitive. To conclude the proof, we have to show that $r = \pi_Z(r) \widehat{\bowtie} \pi_W(r)$.

1. Suppose that $t \in r$. There will be a tuple $t^{(Z)}$ in $\pi_Z(r)$ such that $t[Z]$ and $t^{(Z)}$ are identical. Similarly, there will be a tuple $t^{(W)}$ in $\pi_W(r)$ such that $t[W]$ and $t^{(W)}$ are identical. We have that $t^{(W)}[Z \cap W]$ is identical to $t^{(Z)}[Z \cap W]$. Thus we have that $t \in \pi_Z(r) \widehat{\bowtie} \pi_W(r)$.
2. Suppose that $t \in \pi_Z(r) \widehat{\bowtie} \pi_W(r)$. There must be $t' \in r$ such that $t'[Z] = t[Z]$. We have that $t[Z \cap W]$ and $t'[Z \cap W]$ must be identical because $Z \cap W \subset Z$. Because $Z \cap W \to W$, we have that $t[W]$ and $t'[W]$ are identical. Thus we have that $t[Z \cup W]$ and $t'[Z \cup W]$ which shows that $t \in r$.

This concludes the proof. □

In fact, we can show that logical design is sound over LFDs: as long as we can normalize a relation in the conventional sense, then we can normalize it in the sense of LFDs.

9. CONCLUSION AND FURTHER RESEARCH

We have reviewed the concept of FD in the presence of `null` markers, and have specified a set of properties that we believe any definition of the concept should satisfy. We have proposed two new definitions of what it means for an FD to hold in this situation for which the properties do hold: our definitions satisfy Armstrong's axioms for relations with `null` markers, allow `null` markers to be used in practice (not only with contrived examples), and at the same time allows those `null` markers to be updated to real values consistently. These FDs can also be enforced efficiently in computational terms despite `null` markers. Both definitions have slightly different properties (LFDs enforce lossless join, in addition to database consistency), but they both satisfy our axioms.

Clearly, our requirements are tied to our goal of obtaining a definition that can be used in practical situations. An open question is whether the properties put forth here as desirable are the only ones—or at least the important ones, in some sense of important. We believe that our properties satisfy intuitions that make the concept of FD usable in practical situations. However, additional properties should be explored to get a better understanding of the desired or expected behavior of FDs in the presence of `null` markers; perhaps a set of alternative properties, each giving raise to a concept of FD, can be developed for different scenarios. Agreeing on some set or sets of basic requirements would provide researchers with an explicit milestone by which to judge different formalizations.

A narrower question is whether the definitions proposed here are the only ones that can satisfy all the requirements put forth, or whether alternatives exist. While the authors have considered many alternative definitions, and found them missing some requirement, it is not known whether other definitions could exist that would still satisfy all properties, and if so, what would the relationships between different definitions be.

10. FUNDING

This work was supported by the Natural Sciences and Engineering Research Council of Canada [26143].

11. ACKNOWLEDGEMENTS

We thank Carles Farré and Andre Vellino for their helpful comments.

REFERENCES

[1] Codd, E. F. (1986) Missing information (applicable and inapplicable) in relational databases. SIGMOD Rec., **15**, 53–53.

[2] Zaniolo, C. (1984) Database relations with null values. J. Comput. Syst. Sci., **28**, 142–166.

[3] Atzeni, P. and Morfuni, N. M. (1984) Functional dependencies in relations with null values. Inf. Process. Lett., **18**, 233–238.

[4] Hartmann, S. and Link, S. (2012) The implication problem of data dependencies over SQL table definitions: Axiomatic, algorithmic and logical characterizations. ACM Trans. Database Syst., **37**, 13:1–13:40.

[5] Libkin, L. (1994) Aspects of partial information in databases. PhD thesis University of Pennsylvania.

[6] Badia, A. and Lemire, D. (2011) A call to arms: revisiting database design. SIGMOD Rec., **40**, 61–69.

[7] ISO 9075-1:2008 (2008) Information technology: database languages – SQL– Part 1 Framework, 3rd edition. ISO, Geneva.

[8] Hartmann, S., Leck, U., and Link, S. (2011) On Codd families of keys over incomplete relations. Comput. J., **54**, 1166–1180.

[9] Bernstein, P. (1976) Synthesizing third normal form relations from functional dependencies. ACM Trans. Database Syst., **1**, 277–298.

[10] Chen, P. P.-S. (1976) The entity-relationship model—toward a unified view of data. ACM Trans. Database Syst., **1**, 9–36.

[11] Le, V., Link, S., and Ferrarotti, F. (2013) Effective recognition and visualization of semantic requirements by perfect sql samples. Conceptual Modeling, Lecture Notes in Computer Science, **8217**, pp. 227–240. Springer, Berlin Heidelberg.

[12] Date, C. J. (2009) SQL and Relational Theory: How to Write Accurate SQL Code. O'Reilly Media, Inc., Sebastopol, California.

[13] Imieliński, T. and Lipski, W., Jr. (1984) Incomplete information in relational databases. J. ACM, **31**, 761–791.

[14] Levene, M. and Loizou, G. (1998) Axiomatisation of functional dependencies in incomplete relations. Theor. Comput. Sci., **206**, 283–300.

[15] Chellas, B. F. (1980) Modal logic: an introduction. Cambridge Univ Press, Cambridge.

[16] Ding, B. and König, A. C. (2011) Fast set intersection in memory. Proc. VLDB Endow., **4**, 255–266.