Possible and Certain SQL Keys

Henning Köhler
School of Engineering &
Advanced Technology
Massey University
Palmerston Nth, New Zealand
h.koehler@massey.ac.nz

Sebastian Link
Department of
Computer Science
The University of Auckland
Auckland, New Zealand
s.link@auckland.ac.nz

Xiaofang Zhou
The University of Queensland
Brisbane, Australia; and
Soochow University
Suzhou, China
zxf@itee.uq.edu.au

ABSTRACT

Driven by the dominance of the relational model, the requirements of modern applications, and the veracity of data, we revisit the fundamental notion of a key in relational databases with NULLs. In SQL database systems primary key columns are NOT NULL by default. NULL columns may occur in unique constraints which only guarantee uniqueness for tuples which do not feature null markers in any of the columns involved, and therefore serve a different function than primary keys. We investigate the notions of possible and certain keys, which are keys that hold in some or all possible worlds that can originate from an SQL table, respectively. Possible keys coincide with the unique constraint of SQL, and thus provide a semantics for their syntactic definition in the SQL standard. Certain keys extend primary keys to include NULL columns, and thus form a sufficient and necessary condition to identify tuples uniquely, while primary keys are only sufficient for that purpose. In addition to basic characterization, axiomatization, and simple discovery approaches for possible and certain keys, we investigate the existence and construction of Armstrong tables, and describe an indexing scheme for enforcing certain keys. Our experiments show that certain keys with NULLs do occur in real-world databases, and that related computational problems can be solved efficiently. Certain keys are therefore semantically well-founded and able to maintain data quality in the form of Codd's entity integrity rule while handling the requirements of modern applications, that is, higher volumes of incomplete data from different formats.

1. INTRODUCTION

Keys have always been a core enabler for data management. They are fundamental for understanding the structure and semantics of data. Given a collection of entities, a key is a set of attributes whose values uniquely identify an entity in the collection. For example, a key for a relational table is a set of columns such that no two different rows have matching values in each of the key columns. Keys are essential for many other data models, including semantic models, object models, XML and RDF. They are fundamental in many classical areas of data management, including data modeling, database design, indexing, transaction processing, query optimization. Knowledge about keys enables us to i) uniquely reference entities across data repositories, ii) minimize data redundancy at schema design time to process updates efficiently at run time, iii) provide better selectivity estimates in cost-based query optimization, iv) provide a query optimizer with new access paths that can lead to substantial speedups in query processing, v) allow the database administrator (DBA) to improve the efficiency of data access via physical design techniques such as data partitioning or the creation of indexes and materialized views, vi) enable access to the deep Web, and vii) provide new insights into application data. Modern applications raise the importance of keys even further. They can facilitate the data integration process and prune schema matches. Keys can further help with the detection of duplicates and anomalies, provide guidance in repairing and cleaning data, and provide consistent answers to queries over dirty data. The discovery of keys from data is one of the core activities in data profiling.

According to a Gartner forecast, the NoSQL market will be worth 3.5 billion US dollars annually by 2018, but by that time the market for relational database technology will be worth more than ten times that number, namely 40 billion US dollars annually [12]. This underlines that "relational databases are here to stay, and that there is no substitute for important data" [12]. For these reasons relational databases must meet basic requirements of modern applications, inclusive of big data. On the one hand, relational technology must handle increases in the volume, variety and velocity of data. On the other hand, veracity and quality of this data must still be enforced, to support data-driven decision making. As a consequence, it is imperative that relational databases can acquire as much data as possible without sacrificing reasonable levels of data quality. Nulls constitute a convenient relational tool to accommodate high volumes of incomplete data from different formats. Unfortunately, nulls oppose the current golden standard for data quality in terms of keys, that is, Codd's rule of entity integrity. Entity integrity is one of Codd's integrity rules which states that every table must have a primary key and that the columns which form the primary key must be unique and not null [11]. The goal of entity integrity is to ensure that every tuple in the table can be identified efficiently. In SQL, entity integrity is enforced by adding a primary key clause to a schema definition. The system enforces entity integrity by

Articles from this volume were invited to present their results at the 41st International Conference on Very Large Data Bases, August 31st - September 4th 2015, Kohala Coast, Hawaii.
Proceedings of the VLDB Endowment, Vol. 8, No. 11
Copyright 2015 VLDB Endowment 2150-8097/15/07.

not allowing operations, that is inserts and updates, that produce an invalid primary key. Operations that create a duplicate primary key or one containing nulls are rejected. Consequently, relational databases exhibit a trade-off between their main mechanism to accommodate the requirements of modern applications and their main mechanism to guarantee entity integrity.

We illustrate this trade-off by a powerful example. Consider the snapshot I of the RFAM (RNA families) data set in Table 1, available at `http://rfam.sanger.ac.uk/`. The data set violates entity integrity as every potential primary key over the schema is violated by I. In particular, column *journal* carries the null marker $\perp$. Nevertheless, every tuple in I can be uniquely identified by a combination of column pairs, such as (title, journal) or (author, journal). This is not possible with SQL `unique` constraints which cannot always uniquely identify tuples in which null markers occur in the columns involved. We conclude that the syntactic requirements of primary keys are sufficient to meet the goal of entity integrity. However, the requirements are not necessary since they prohibit the entry of some data without good reason. The inability to insert important data into the database may force organizations to abandon key validation altogether, exposing their future database to less data quality, inefficiencies in data processing, waste of resources, and poor data-driven decision making. Finding a notion of keys that is sufficient and necessary to meet the goal of entity integrity will bring forward a new database technology that accommodates the requirements of modern applications better than primary keys do.

title	*author*	*journal*
The uRNA database	Zwieb C	Nucleic Acids 1997
The uRNA database	Zwieb C	Nucleic Acids 1996
Genome wide detect.	Ragh. R	$\perp$

Table 1: Snippet I of the RFAM data set

These observations have motivated us to investigate keys over SQL tables from a well-founded semantic point of view. For this purpose, we permit interpretations of $\perp$ as missing or unknown data. This interpretation leads to a possible world semantics, in which a possible world results from a table by replacing independently each occurrence of $\perp$ by a value from the corresponding domain (missing data is represented by a distinguished domain 'value'). A possible world can therefore be handled as a relation of total tuples in which duplicate tuples may occur. As usual, a possible world w satisfies a key X if and only if there are no two tuples in w that have distinct tuple identities and matching values on all the attributes in X. For example, the possible world w_1 of I in Table 2 satisfies keys such as $\{$*journal*$\}$, $\{$*author,journal*$\}$ and $\{$*title,journal*$\}$, and the possible world w_2 of I in Table 3 satisfies the keys $\{$*author,journal*$\}$ and $\{$*title,journal*$\}$ but violates the key $\{$*journal*$\}$. When required we stipulate any minimality requirement on keys explicitly.

title	*author*	*journal*
The uRNA database	Zwieb C	Nucleic Acids 1997
The uRNA database	Zwieb C	Nucleic Acids 1996
Genome wide detect.	Ragh. R	PLOS One 2000

Table 2: Possible World w_1 of snippet I in Table 1

title	*author*	*journal*
The uRNA database	Zwieb C	Nucleic Acids 1997
The uRNA database	Zwieb C	Nucleic Acids 1996
Genome wide detect.	Ragh. R	Nucleic Acids 1997

Table 3: Possible World w_2 of snippet I in Table 1

This approach naturally suggests two semantics of keys over SQL tables. A *possible key* $p\langle X\rangle$ is satisfied by an SQL table I if and only if there is some possible world of I in which the key X is satisfied. A *certain key* $c\langle X\rangle$ is satisfied by an SQL table I if and only if the key X is satisfied in every possible world of I. In particular, the semantics of certain keys does not prevent the entry of incomplete tuples that can still be identified uniquely, independently of which data null marker occurrences represent. For example, the snapshot I in Table 1 satisfies the possible key $p\langle journal\rangle$ as witnessed by world w_1 in Table 2, but violates the certain key $c\langle journal\rangle$ as witnessed by world w_2 in Table 3. Moreover, I satisfies the certain keys $c\langle title, journal\rangle$ and $c\langle author, journal\rangle$. The example illustrates that primary keys cannot enforce entity integrity on data sets that originate from modern applications, while certain keys can. In fact, primary keys are only sufficient to uniquely identify tuples in SQL tables, while certain keys are also necessary. Our observations provide strong motivation to investigate possible and certain keys in detail. In particular, our research provides strong evidence that certain keys can meet the golden standard of entity integrity and meet the requirements of modern applications. The contributions of our research can be summarized as follows:

1. We propose a possible world semantics for keys over SQL tables. Possible keys hold in some possible world, while certain keys hold in all possible worlds. Hence, certain keys identify tuples without having to declare all attributes `NOT NULL`. While primary keys provide a sufficient condition to identify tuples, the condition is not necessary. Certain keys provide a sufficient and necessary condition, thereby not prohibiting the entry of data that can be uniquely identified.

2. We establish simple syntactic characterizations to validate the satisfaction of possible and certain keys. Permitting possible worlds to be multisets rather than sets means that possible keys provide a semantics for SQL's `unique` constraint. That is, `unique`(X) is satisfied by an SQL table if and only if it satisfies $p\langle X\rangle$.

3. We characterize the implication problem for the combined class of possible and certain keys and `NOT NULL` constraints axiomatically and by an algorithm that works in linear time in the input. This shows that possible and certain keys can be reasoned about efficiently. As an important application, we can efficiently compute *the* minimal cover of our constraints in order to reduce the amount of integrity maintenance to a minimal level necessary. Our algorithm requires only little time to eliminate all future redundant enforcements of keys on potentially big data sets. Consequently, the bigger the data sets become the more time we save by not having to check redundant keys.

4. We address the data-driven discovery of possible and certain keys. Exploiting hypergraph transversals, we establish a compact algorithm to compute a cover for the possible and certain keys that hold on a given table. The discovery algorithm allows us to find possible and certain keys that hold on publicly available data sets. Several of the certain keys permit null marker occurrences, which makes them different from primary keys. Hence, certain keys do occur in practice, providing strong motivation for further research and exploiting their features in database technology. The discovery of keys is useful in many applications, as outlined before. This line of our research is also a contribution to the new important area of data profiling [35].

5. We investigate structural and computational aspects of Armstrong tables. Given some set of possible keys, certain keys and NOT NULL constraints, an Armstrong table for this set is an SQL table that satisfies the constraints in this set but violates all possible keys, certain keys and NOT NULL constraints that are not implied by the set. For example, snapshot I of Table 1 is an Armstrong table for $p\,\langle journal\rangle$, $c\,\langle title, journal\rangle$, $c\,\langle author, journal\rangle$ and the NOT NULL constraints on *title* and *author*. Despite much more involved challenges as encountered in the idealized special case of pure relations, we characterize when Armstrong tables exist and how to compute them in these cases. While being, in theory, worst-case double exponential in the input, our algorithm is very efficient in practice. For example, for a table with 20 attributes, a brute-force approach would require $2^{2^{20}}$ operations, i.e., never finish. While the theoretical worst-case bound for our algorithm is not much better, it actually establishes an Armstrong table within milliseconds. We also provide circumstantial evidence that the worst-case bound is difficult to improve upon in theory. For this purpose we show the NP-hardness of the key/antikey satisfiability problem, whose input size is worst-case exponential in the input to the Armstrong table existence problem, and closely linked to it. Armstrong tables provide a tool for database designers to communicate effectively with domain experts in order to acquire a more complete set of semantically meaningful integrity constraints. It is well-known that this results in better database designs, better data quality, more efficient data processing, exchange and integration, resource savings and better decision-making [33].

6. We propose an indexing scheme for certain keys. Our scheme improves the enforcement of certain keys on inserts by several orders of magnitude. It works only marginally slower than the enforcement of primary keys, provided that the certain keys have only a small number of columns in which null markers can occur. Exploiting our data-driven discovery algorithm from before, we have found only certain keys in which at most two columns can feature null markers.

7. Besides the discovery of possible and certain keys in real-life data sets we conducted several other experiments. These confirm our intuition that the computational problems above can be solved efficiently in practice. For example, we applied our construction of Armstrong tables to the possible and certain keys we had previously discovered, resulting in tables containing only seven rows on average, which makes them particularly useful for semantic samples in data profiling or as a communication tool when requirements are acquired from domain experts. Furthermore, their computation only took a few milliseconds in each of the 130 cases. For only 85 out of 1 million randomly generated schemata and sets of keys, Armstrong tables did not exist, otherwise Armstrong tables were computed in a few milliseconds. Although the computation of Armstrong tables may take exponential time in the worst case, such cases need to be constructed carefully and occur at best sparingly in practice. Experiments with our scheme showed that i) certain keys in practice are likely to not have more than two columns in which null markers occur, and ii) such certain keys can be enforced almost as efficiently as primary keys.

Our findings provide strong evidence that certain keys achieve the golden standard of Codd's principle of entity integrity under the requirements of modern applications.

Organization. Section 2 discusses related work further motivating our research. Possible and certain keys are introduced in Section 3 where we also establish their syntactic characterization, axiomatic and algorithmic solutions to their implication problem, the computation of their minimal cover, and their discovery from given tables. Structural and computational aspects of Armstrong tables are investigated in Section 4. An efficient indexing scheme for the enforcement of certain keys is established in Section 5, and results of our experiments are presented in Section 6. We conclude and comment on future work in Section 7. Proofs and further material are available in a technical report [27].

2. RELATED WORK

Integrity constraints enforce the semantics of application domains in database systems. They form a cornerstone of database technology [1]. Entity integrity is one of the three inherent integrity rules proposed by Codd [11]. Keys and foreign keys are the only ones amongst around 100 classes of constraints [1] that enjoy built-in support by SQL database systems. In particular, entity integrity is enforced by primary keys [34]. Core problems investigate reasoning [21], Armstrong databases [18], and discovery [22, 24, 32, 33, 40]. Applications include anomaly detection [43], consistency management [4], consistent query answers [3, 28], data cleaning [17], exchange [16], fusion [36], integration [9], profiling [35], quality [38], repairs [6], and security [7], schema design [14], query optimization [23], transaction processing [2], and view maintenance [37]. Surrogate keys ('autoincrement' fields) do not help with enforcing domain semantics or supporting applications while semantic keys do.

One of the most important extensions of the relational model [11] is incomplete information, due to the high demand for the correct handling of such information in real-world applications. We focus on SQL where many kinds of null markers have been proposed. The two most prolific proposals are "value unknown at present" [11] and "no information" [31, 42]. Even though their semantics differ, both notions possess the (for our purposes) fundamental property that different null marker occurrences represent either different or identical values in possible worlds. This holds even if the two notions were to be mixed, allowing two different interpretations of null marker occurrences. As a consequence, our definition of key satisfaction in possible worlds (where for "no information" nulls we permit possible worlds with missing values) remains sensible under either notion, and leads to the same syntactical characterization. Since all our results are based on this syntactical characterization, established in Section 3.2, they continue to hold under "no information" interpretation. For simplicity we will only refer to the "value unknown at present" interpretation for the remainder of the paper, but note that missing data can simply be handled as above.

We adopt a principled approach that defines the semantics of constraints in terms of possible worlds. Levene and Loizou introduced strong and weak functional dependencies (FDs)

[29]. We take a similar approach and strong/weak FDs and possible/certain keys are closely related. However, neither are certain keys special cases of strong FDs, nor are possible keys special cases of weak FDs. The reason is that we permit duplicate tuples in possible worlds, which is necessary to obtain entity integrity. Consider I_1 in Table 4: if we prohibited possible worlds with duplicates, title would become a certain key. Under such multiset semantics, keys are no longer special cases of FDs, since the former prohibit duplicate tuples and the latter do not. Duplicate tuples occur naturally in modern applications, such as data integration, entity linking and de-duplication, and must be accommodated by any reasonable notion of a key. For example, the snippet I_1 in Table 4 satisfies the strong FD $\mathit{title} \to \mathit{author}$, i.e., $\{\mathit{title}\}$ is a strong key for I_1, but $c\,\langle \mathit{title}\rangle$ is not a certain key for I_1. In particular, the world that results from replacing $\perp$ in I_1 with 'uRNA' contains two duplicate tuples that violate the key $\{\mathit{title}\}$. Similarly, the snippet I_2 in Table 4 satisfies the weak FD $\mathit{author} \to \mathit{journal}$, i.e., $\{\mathit{author}\}$ is a weak key for I_2, but $p\,\langle \mathit{author}\rangle$ is not a possible key for I_2. In particular, the world that results from replacing $\perp$ in I_2 with 'Acids' contains two duplicate tuples that violate the key $\{\mathit{author}\}$. The snippet I_2 further illustrates the benefit of permitting multisets of tuples as possible worlds. This guarantees that SQL's `unique` constraint coincides with our possible keys, but not with weak keys: I_2 violates `unique`(author) and $p\,\langle \mathit{author}\rangle$, but satisfies the weak key $\{\mathit{author}\}$.

I_1

title	*author*
uRNA	Zwieb
$\perp$	Zwieb

I_2

author	*journal*
Zwieb C	$\perp$
Zwieb C	Acids

Table 4: SQL tables I_1 and I_2

The focus in [29] is on axiomatization and implication of strong/weak FDs. While Armstrong tables are claimed to exist for any set of weak/strong FDs [29], we discovered a technical error in the proof. Indeed, in the technical report [27], Example 15 on page 42 shows a set of strong and weak FDs for which no Armstrong table exists.

The principle of entity integrity has been challenged by Thalheim [41] and later by Levene and Loizou [30], both following an approach different from ours. They propose the notion of a key set. A relation satisfies a key set if, for each pair of distinct tuples, there is some key in the key set on which the two tuples are total and distinct. A certain key is equivalent to a key set consisting of all the singleton subsets of the key attributes, e.g., $c\,\langle \mathit{title}, \mathit{author}, \mathit{journal}\rangle$ corresponds to the key set $\{\{\mathit{title}\}, \{\mathit{author}\}, \{\mathit{journal}\}\}$. However, our work is different in that we study the interaction with possible keys and `NOT NULL` attributes, establish a possible world semantics, and study different problems. The implication problem for the sole class of primary keys is examined in [19]. As the key columns of primary keys are `NOT NULL`, the class of primary keys behaves differently from both possible and certain keys, see Section 3.3.

We emphasize that our findings may also be important for other data models such as XML [20] and RDF [10] where incomplete information is inherent, probabilistic databases [25] where keys can be expected to hold with some probability other than 1, and also description logics [10].

Summary. Certain keys appear to be the most natural approach to address the efficient identification of tuples in SQL tables. It is therefore surprising that they have not been considered in previous work. In this paper, we investigate the combined class of possible and certain keys under `NOT NULL` constraints. The combination of these constraints is particularly relevant to SQL, as possible keys correspond to SQL's `unique` constraint. The presence of certain keys also means that the problems studied here are substantially different from those investigated elsewhere.

3. POSSIBLE AND CERTAIN KEYS

We start with some preliminaries before introducing the notions of possible and certain keys. Subsequently, we characterize these notions syntactically, from which we derive both a simple axiomatic and a linear time algorithmic characterization of the implication problem. We show that possible and certain keys enjoy a unique minimal representation. Finally, we exploit hypergraph transversals to discover possible and certain keys from a given table.

3.1 Preliminaries

We begin with basic terminology. Let $\mathfrak{A} = \{A_1, A_2, \ldots\}$ be a (countably) infinite set of distinct symbols, called *attributes*. Attributes represent column names of tables. A *table schema* is a finite non-empty subset T of $\mathfrak{A}$. Each attribute A of a table schema T is associated with an infinite domain $\mathit{dom}(A)$ which represents the possible values that can occur in column A. In order to encompass incomplete information the domain of each attribute contains the null marker, denoted by $\perp$. As explained in Section 2 we restrict the interpretation of $\perp$ to "value unknown at present" but only to ease presentation. Although not a value, we include $\perp$ in attribute domains as a syntactic convenience.

For attribute sets X and Y we may write XY for their set union $X \cup Y$. If $X = \{A_1, \ldots, A_m\}$, then we may write $A_1 \cdots A_m$ for X. In particular, we may write A to represent the singleton $\{A\}$. A *tuple* over T is a function $t : T \to \bigcup_{A \in T} \mathit{dom}(A)$ with $t(A) \in \mathit{dom}(A)$ for all $A \in X$. For $X \subseteq T$ let $t[X]$ denote the restriction of the tuple t over T to X. We say that a tuple t is X*-total* if $t[A] \neq \perp$ for all $A \in X$. A tuple t over T is said to be a *total tuple* if it is T-total. A *table* I over T is a finite multiset of tuples over T. A table I over T is a *total table* if every tuple $t \in I$ is total. Let t, t' be tuples over T. We define *weak/strong similarity* of t, t' on $X \subseteq T$ as follows:

$$\begin{aligned} t[X] \sim_w t'[X] \quad &:\Leftrightarrow \quad \forall A \in X. \\ &\qquad (t[A] = t'[A] \vee t[A] = \perp \vee t'[A] = \perp) \\ t[X] \sim_s t'[X] \quad &:\Leftrightarrow \quad \forall A \in X. \\ &\qquad (t[A] = t'[A] \neq \perp) \end{aligned}$$

Weak and strong similarity become identical for tuples that are X-total. In such "classical" cases we denote similarity by $t[X] \sim t'[X]$. We will use the phrase t, t' *agree* interchangeably for t, t' *are similar*.

A *null-free subschema* (NFS) over the table schema T is a set T_S where $T_S \subseteq T$. The NFS T_S over T is satisfied by a table I over T if and only if I is T_S-total. SQL allows the specification of attributes as `NOT NULL`, so the set of attributes declared `NOT NULL` forms an NFS over the underlying table schema. For convenience we sometimes refer to the pair (T, T_S) as table schema.

We say that $X \subseteq T$ is a *key* for the total table I over T, denoted by $I \vdash X$, if there are no two tuples $t, t' \in I$ that have distinct tuple identities and agree on X. We will now define two different notions of keys over general tables, using a possible world semantics. Given a table I on T, a *possible world* of I is obtained by independently replacing every occurrence of $\perp$ in I with a domain value. We say that $X \subseteq T$ is a *possible/certain key* for I, denoted by $p\,\langle X\rangle$ and $c\,\langle X\rangle$ respectively, if the following hold:

$$I \vdash p\,\langle X\rangle :\Leftrightarrow X \text{ is a key for some possible world of } I$$
$$I \vdash c\,\langle X\rangle :\Leftrightarrow X \text{ is a key for every possible world of } I\ .$$

We illustrate this semantics on our running example.

Example 1. *For $T = \{title, author, journal\}$ let I denote that is, Table 1 from before. Then $I \vdash p\,\langle journal\rangle$ as $\perp$ can be replaced by a domain value different from the journals in I. Furthermore, $I \vdash c\,\langle title, journal\rangle$ as the first two rows are unique on journal, and the last row is unique on title, independently of the replacement for $\perp$. Finally, $I \vdash c\,\langle author, journal\rangle$ as the first two rows are unique on journal, and the last row is unique on author, independently of the replacement for $\perp$.*

title	*author*	*journal*
The uRNA database	*Zwieb C*	*Nucleic Acids 1997*
The uRNA database	*Zwieb C*	*Nucleic Acids 1996*
Genome wide detect.	*Ragh. R*	$\perp$

For a set Σ of constraints over table schema T we say that a table I over T *satisfies* Σ if I satisfies every $\sigma \in \Sigma$. If for some $\sigma \in \Sigma$ the table I does not satisfy σ we say that I *violates* σ (and violates Σ). A table I over (T, T_S) is a table I over T that satisfies T_S. A table I over (T, T_S, Σ) is a table I over (T, T_S) that satisfies Σ.

When discussing possible and certain keys, the following notions of strong and weak anti-keys will prove useful. Let I be a table over (T, T_S) and $X \subseteq T$. We say that X is a *strong/weak anti-key* for I, denoted by $\neg_p\langle X\rangle$ and $\neg_c\langle X\rangle$ respectively, if $p\,\langle X\rangle$ and $c\,\langle X\rangle$, respectively, do *not* hold on I. We may also say that $\neg_p\langle X\rangle$ and/or $\neg_c\langle X\rangle$ *hold* on I. We write $\neg\,\langle X\rangle$ to denote an anti-key which may be either strong or weak. A set Σ of constraints over (T, T_S) *permits* a set Π of strong and weak anti-keys if there is a table I over (T, T_S, Σ) such that every anti-key in Π holds on I. We illustrate this semantics on our running example.

Example 2. *For $T = \{title, author, journal\}$ let I denote the table over T from Table 1. Then $I \vdash \neg_p\langle title, author\rangle$ as the first two rows will agree on title and author in every possible world. Furthermore, $I \vdash \neg_c\langle journal\rangle$ as $\perp$ could be replaced by either of the two journals listed in I, resulting in possible worlds that violate the key $\{journal\}$.*

3.2 Syntactic Characterization

The following result characterizes the semantics of possible and certain keys syntactically, using strong and weak similarity. The characterization shows, in particular, that possible keys capture the `unique` constraint in SQL. Therefore, we have established a first formal semantics for the SQL `unique` constraint, based on Codd's null marker interpretation of "value exists, but unknown". Moreover, Theorem 1 provides a foundation for developing efficient algorithms that effectively exploit our keys in data processing.

Theorem 1. *$X \subseteq T$ is a possible (certain) key for I iff no two tuples in I with distinct tuple identities are strongly (weakly) similar on X.*

We illustrate the syntactic characterization of our key semantics from Theorem 1 on our running example.

Example 3. *Suppose I denotes Table 1 over $T = \{title, author, journal\}$. Then $I \vdash p\,\langle journal\rangle$ as no two different I-tuples strongly agree on journal. Furthermore, $I \vdash c\,\langle title, journal\rangle$ as no two different I-tuples weakly agree on title and journal. Finally, $I \vdash c\,\langle author, journal\rangle$ as no two different I-tuples weakly agree on author and journal.*

3.3 Implication

Many data management tasks, including data profiling, schema design, transaction processing, and query optimization, benefit from the ability to decide the implication problem of semantic constraints. In our context, the implication problem can be defined as follows. Let (T, T_S) denote the schema under consideration. For a set $\Sigma \cup \{\varphi\}$ of constraints over (T, T_s) we say that Σ *implies* φ, denoted by $\Sigma \models \varphi$, if and only if every table over (T, T_S) that satisfies Σ also satisfies φ. The *implication problem* for a class $\mathcal{C}$ of constraints is to decide, for an arbitrary (T, T_S) and an arbitrary set $\Sigma \cup \{\varphi\}$ of constraints in $\mathcal{C}$, whether Σ implies φ. For possible and certain keys the implication problem can be characterized as follows.

Theorem 2. *Let Σ be a set of possible and certain keys. Then Σ implies $c\,\langle X\rangle$ iff $c\,\langle Y\rangle \in \Sigma$ for some $Y \subseteq X$ or $p\,\langle Z\rangle \in \Sigma$ for some $Z \subseteq X \cap T_S$. Furthermore, Σ implies $p\,\langle X\rangle$ iff $c\,\langle Y\rangle \in \Sigma$ or $p\,\langle Y\rangle \in \Sigma$ for some $Y \subseteq X$.*

Thus, a given certain key is implied by Σ iff it contains a certain key from Σ or its `NOT NULL` columns contain a possible key from Σ. Similarly, a given possible key is implied by Σ iff it contains a possible or certain key from Σ. We exemplify Theorem 2 on our running example.

Example 4. *Let $T = \{title, author, journal\}$ be our table schema, $T_S = \{title, author\}$ our NFS, and let Σ consist of $p\,\langle journal\rangle$, $c\,\langle title, journal\rangle$ and $c\,\langle author, journal\rangle$. Theorem 2 shows that Σ implies $c\,\langle title, author, journal\rangle$ and $p\,\langle title, journal\rangle$, but neither $c\,\langle journal\rangle$ nor $p\,\langle title, author\rangle$. This is independently confirmed by Table 1, which satisfies $c\,\langle title, author, journal\rangle$ and $p\,\langle title, journal\rangle$, but it violates $c\,\langle journal\rangle$ and $p\,\langle title, author\rangle$.*

A consequence of Theorem 2 is that the implication of our keys can be decided with just one scan over the input.

Theorem 3. *The implication problem for the class of possible and certain keys can be decided in linear time.*

Theorem 3 indicates that the semantics of our keys can be exploited efficiently in many data management tasks. We illustrate this on the following example.

Example 5. *Suppose we want to find all distinct combinations of authors and journals from the current instance over table schema $T = \{title, author, journal\}$ where $T_S = \{title, author\}$ denotes our NFS. For the query*

```
SELECT DISTINCT author, journal
FROM T
WHERE journal IS NOT NULL
```

an optimizer may check if the certain key $c\langle author, journal\rangle$ is implied by the set Σ of specified constraints on (T, T_S) together with the additional `NOT NULL` constraint on journal, enforced by the `WHERE` clause. If that is indeed the case, as for the set Σ in Example 4, the `DISTINCT` clause is superfluous and can be omitted, thereby saving the expensive operation of duplicate removal.

3.4 Axiomatization

An axiomatic characterization of the implication problem provides us with a tool for gaining an understanding of and reasoning about the interaction of possible and certain keys and `NOT NULL` constraints. The following result presents a sound and complete axiomatization for this combined class of constraints. The definitions of *sound* and *complete* sets of axioms are standard [1].

Theorem 4. *The following axioms are sound and complete for the implication of possible and certain keys.*

$$\textit{p-Extension: } \frac{p\langle X\rangle}{p\langle XY\rangle} \qquad \textit{c-Extension: } \frac{c\langle X\rangle}{c\langle XY\rangle}$$

$$\textit{Weakening: } \frac{c\langle X\rangle}{p\langle X\rangle} \qquad \textit{Strengthening: } \frac{p\langle X\rangle}{c\langle X\rangle}\, X \subseteq T_S$$

We note that Theorem 4 also constitutes an invaluable foundation for establishing further important results about our class of keys. A simple application of the inference rules is shown on our running example.

Example 6. *Let $T = \{title, author, journal\}$ be our table schema, $T_S = \{title, author\}$ our NFS, and let Σ consist of $p\langle journal\rangle$, $c\langle title, journal\rangle$ and $c\langle author, journal\rangle$. Then $p\langle author, journal\rangle$ can be inferred from Σ by a single application of p-Extension to $p\langle journal\rangle$, or by a single application of Weakening to $c\langle author, journal\rangle$.*

3.5 Minimal Covers

A *cover* of Σ is a set Σ' where every element is implied by Σ and which implies every element of Σ. Hence, a cover is just a representation. Minimal representations of constraint sets are of particular interest in database practice. Firstly, they justify the use of valuable resources, for example, by limiting the validation of constraints to those necessary. Secondly, people find minimal representations easier to work with than non-minimal ones. For the class of possible and certain keys, a unique minimal representation exists.

We say that i) $p\langle X\rangle \in \Sigma$ is *non-minimal* if $\Sigma \models p\langle Y\rangle$ for some $Y \subset X$ or $\Sigma \models c\langle X\rangle$, and ii) $c\langle X\rangle \in \Sigma$ is *non-minimal* if $\Sigma \models c\langle Y\rangle$ for some $Y \subset X$. We call a key σ with $\Sigma \models \sigma$ *minimal* iff σ is not non-minimal. We call $\sigma \in \Sigma$ *redundant* iff $\Sigma \setminus \{\sigma\} \models \sigma$, and *non-redundant* otherwise. We call Σ *minimal* (*non-redundant*) iff all keys in Σ are minimal (non-redundant), and *non-minimal* (*redundant*) otherwise.

Due to the logical equivalence of $p\langle X\rangle$ and $c\langle X\rangle$ for $X \subseteq T_S$, certain keys can be both minimal and redundant while possible keys can be both non-minimal and non-redundant.

Theorem 5. *The set Σ_{min} of all minimal possible and certain keys w.r.t. Σ is a non-redundant cover of Σ. Indeed, Σ_{min} is the only minimal cover of Σ.*

We may hence talk about *the* minimal cover of Σ, and illustrate this concept on our running example.

Example 7. *Let $T = \{title, author, journal\}$ be our table schema, $T_S = \{title, author\}$ our NFS, let Σ' consist of $p\langle journal\rangle$, $c\langle title, journal\rangle$, $c\langle author, journal\rangle$, $p\langle author, journal\rangle$ and $c\langle title, author, journal\rangle$. The minimal cover Σ of Σ' consists of $p\langle journal\rangle$, $c\langle title, journal\rangle$ and $c\langle author, journal\rangle$.*

A strong anti-key $\neg_p\langle X\rangle$ is *non-maximal* if $\neg\langle Y\rangle$ with $X \subset Y$ is an anti-key implying $\neg_p\langle X\rangle$ on (T, T_S). A weak anti-key $\neg_c\langle X\rangle$ is *non-maximal* if $\neg\langle Y\rangle$ with $X \subset Y$ is an anti-key or $\neg_p\langle X\rangle$ is a strong anti-key. Anti-keys are *maximal* unless they are non-maximal. We denote the set of maximal strong anti-keys by $\mathcal{A}^s_{max}$, the set of maximal weak anti-keys by $\mathcal{A}^w_{max}$ and their disjoint union by $\mathcal{A}_{max}$.

3.6 Key Discovery

Our next goal is to discover all certain and possible keys that hold in a given table I over (T, T_S). If T_S is not given it is trivial to find a maximal T_S. The discovery of constraints from data reveals semantic information useful for database design and administration, data exchange, integration and profiling. Combined with human expertise, meaningful constraints that are not specified or meaningless constraints that hold accidentally may be revealed.

Keys can be discovered from total tables by computing the agree sets of all pairs of distinct tuples, and then computing the transversals for their complements [13]. On general tables we distinguish between strong and weak agree sets, motivated by our notions of strong and weak similarity.

Given two tuples t, t' over T, the *weak (strong) agree set* of t, t' is the (unique) maximal subset $X \subseteq T$ such that t, t' are weakly (strongly) similar on X. Given a table I over T, we denote by $\mathcal{AG}^w(I), \mathcal{AG}^s(I)$ the set of all maximal agree sets of distinct tuples in I:

$$\mathcal{AG}^w(I) := \max\{X \mid \exists t \neq t' \in I . t[X] \sim_w t'[X]\}$$
$$\mathcal{AG}^s(I) := \max\{X \mid \exists t \neq t' \in I . t[X] \sim_s t'[X]\}$$

We shall simply write $\mathcal{AG}^w, \mathcal{AG}^s$ when I is clear from the context. Complements and transversals are standard notions for which we now introduce notation [13].

Let X be a subset of T and $\mathcal{S}$ a set of subset of T. We use the following notation for complements and transversals:

$$\overline{X} := T \setminus X$$
$$\overline{\mathcal{S}} := \{\overline{X} \mid X \in \mathcal{S}\}$$
$$Tr(\mathcal{S}) := \min\{Y \subseteq T \mid \forall X \in \mathcal{S} . Y \cap X \neq \emptyset\}$$

Our main result on key discovery shows that the certain (possible) keys that hold in I are the transversals of the complements for all weak (strong) agree sets in I.

Theorem 6. *Let I be a table over (T, T_S), and Σ_I the set of all certain and/or possible keys that hold on I. Then*

$$\Sigma := \{c\langle X\rangle \mid X \in Tr(\overline{\mathcal{AG}^w})\} \cup \{p\langle X\rangle \mid X \in Tr(\overline{\mathcal{AG}^s})\}$$

is a cover of Σ_I.

Our goal was to show that elegant tools such as hypergraph transversals can be used to discover our keys in real-world data sets. Results of those experiments are presented in Section 6. Optimizations and scalability of our method to big data are left for future research, see [35]. We illustrate the discovery method on our running real-world example.

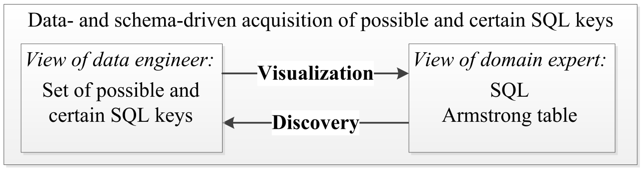
Data- and schema-driven acquisition of possible and certain SQL keys
View of data engineer:
Set of possible and certain SQL keys
Visualization
Discovery
View of domain expert:
SQL Armstrong table

Figure 1: Acquisition Framework for SQL Keys

Example 8. *Let I denote Table 1 over table schema $T = \{title, author, journal\}$. Then $T_S = \{title, author\}$ can be easily computed. Then $\mathcal{AG}^w(I)$ consists of $\{title,author\}$ and $\{journal\}$, with complements $\{journal\}$ and $\{title,author\}$ in $\overline{\mathcal{AG}^w}$, and transversals $\{title,journal\}$ and $\{author,journal\}$. $\mathcal{AG}^s(I)$ consists of $\{title,author\}$, with complement $\{journal\}$ in $\overline{\mathcal{AG}^s}$, and transversal $\{journal\}$. Therefore, a cover of the set of possible and certain keys that holds on I consists of $c\langle title, journal\rangle$, $c\langle author, journal\rangle$, and $p\langle journal\rangle$.*

4. ARMSTRONG TABLES

Armstrong tables are widely regarded as a user-friendly, exact representation of abstract sets of constraints [5, 15, 18, 33]. For a class $\mathcal{C}$ of constraints and a set Σ of constraints in $\mathcal{C}$, a $\mathcal{C}$-Armstrong table I for Σ satisfies Σ and violates all the constraints in $\mathcal{C}$ not implied by Σ. Therefore, given an Armstrong table I for Σ the problem of deciding for an arbitrary constraint in $\mathcal{C}$ whether Σ implies φ reduces to the problem of verifying whether φ holds on I. The ability to compute an Armstrong table for Σ provides us with a data sample that is a perfect summary of the semantics embodied in Σ. Unfortunately, classes $\mathcal{C}$ cannot be expected at all to enjoy Armstrong tables. That is, there are sets Σ for which no $\mathcal{C}$-Armstrong table exists [15]. Classically, keys and even functional dependencies enjoy Armstrong relations [5, 33]. However, for possible and certain keys under `NOT NULL` constraints the situation is much more involved. Nevertheless, we will characterize when Armstrong tables exist, and establish structural and computational properties for them.

4.1 Acquisition Framework

Applications benefit from the ability of data engineers to acquire the keys that are semantically meaningful in the domain of the application. For that purpose data engineers communicate with domain experts. We establish two major tools to improve communication, as shown in the framework of Figure 1. Here, data engineers use our algorithm to visualize abstract sets Σ of keys as an Armstrong table I_Σ, which is inspected jointly with domain experts. Domain experts may change I_Σ or provide new data samples. Our discovery algorithm from Theorem 6 is then used to discover the keys that hold in the given sample.

4.2 Definition and Motivating Examples

An instance I over (T, T_S) is a *pre-Armstrong table* for (T, T_S, Σ) if for every key σ over T, σ holds on I iff $\Sigma \vDash \sigma$. We call I an *Armstrong table* if it is pre-Armstrong and for every `NULL` attribute $A \in T \setminus T_S$ there exists a tuple $t \in I$ with $t[A] = \bot$. Our first example presents a case where a pre-Armstrong table but no Armstrong tables exist.

Example 9. *Let $(T, T_S, \Sigma) = (AB, A, \{c\langle B\rangle\})$. The following is easily seen to be a pre-Armstrong table:*

A	B
0	0
0	1

Now let I be an instance over (T, T_S, Σ) with $t \in I$ such that $t[B] = \bot$. Then the existence of any other tuple $t' \in I$ with $t' \neq t$ would violate $c\langle B\rangle$, so $I = \{t\}$. But that means $c\langle A\rangle$ holds on I even though $\Sigma \nvDash c\langle A\rangle$, so I is not Armstrong.

We now present a case where no pre-Armstrong tables exist.

Example 10. *Let $(T, T_S) = (ABCD, \emptyset)$ and*

$$\Sigma = \{c\langle AB\rangle, c\langle CD\rangle, p\langle AC\rangle, p\langle AD\rangle, p\langle BC\rangle, p\langle BD\rangle\}$$

Then a pre-Armstrong table I must disprove the certain keys $c\langle AC\rangle$, $c\langle AD\rangle$, $c\langle BC\rangle$, $c\langle BD\rangle$ while respecting the possible keys $p\langle AC\rangle$, $p\langle AD\rangle$, $p\langle BC\rangle$, $p\langle BD\rangle$. In each of these four cases, we require two tuples t, t' which are weakly but not strongly similar on the corresponding key sets (e.g. $t[AC] \sim_w t'[AC]$ but $t[AC] \not\sim_s t'[AC]$). This is only possible when t or t' are $\bot$ on one of the key attributes. This ensures the existence of tuples $t_{AC}, t_{AD}, t_{BC}, t_{BD}$ with

$$t_{AC}[A] = \bot \lor t_{AC}[C] = \bot, \quad t_{AD}[A] = \bot \lor t_{AD}[D] = \bot,$$
$$t_{BC}[B] = \bot \lor t_{BC}[C] = \bot, \quad t_{BD}[B] = \bot \lor t_{BD}[D] = \bot$$

If $t_{AC}[A] \neq \bot$ and $t_{AD}[A] \neq \bot$ it follows that $t_{AC}[C] = \bot$ and $t_{AD}[D] = \bot$. But this means $t_{AC}[CD] \sim_w t_{AD}[CD]$, contradicting $c\langle CD\rangle$. Hence there exists $t_A \in \{t_{AC}, t_{AD}\}$ with $t_A[A] = \bot$. Similarly we get $t_B \in \{t_{BC}, t_{BD}\}$ with $t_B[B] = \bot$. Then $t_A[AB] \sim_w t_B[AB]$ contradicting $c\langle AB\rangle$. In fact, if $t_{AC} = t_{AD}$ then $t_{AC}[CD] = (\bot, \bot)$ is weakly similar to every other tuple.

4.3 Structural Characterization

Motivated by our examples we will now characterize the cases for which Armstrong tables do exist, and show how to construct them whenever possible. While the research has been technically challenging, it has significant rewards in theory and practice. In practice, Armstrong tables facilitate the acquisition of requirements, see Figure 1. This is particularly appealing to our keys since the experiments in Section 6 confirm that i) key sets for which Armstrong tables do not exist are rare, and ii) keys that represent real application semantics can be enforced efficiently in SQL database systems. Our findings illustrate the impact of nulls on the theory of Armstrong databases, and have revealed a technical error in previous research [29], see Section B of the appendix in the technical report [27].

In case that $\Sigma \vDash c\langle\emptyset\rangle$ there is an Armstrong table with a single tuple only. Otherwise every pre-Armstrong table contains at least two tuples. We will now characterize the existence of (pre-)Armstrong tables. The core challenge was to characterize reconcilable situations between the given possible keys and null-free subschema, and the implied maximal weak anti-keys. In fact, possible keys require some possible world in which all tuples disagree on some key attribute while weak anti-keys require some possible world in which some tuples agree on all key attributes. So, whenever a weak anti-key contains a possible key, this situation is only reconcilable by the use of the $\bot$ marker. However, such $\bot$ occurrence can cause unintended weak similarity of tuples, as seen in Example 10. Consequently, standard Armstrong table construction techniques [5, 18, 33], which essentially deal with each anti-constraint in isolation, cannot be applied here. We introduce some more notation to present the characterization. Let $\mathcal{V}, \mathcal{W} \subseteq \mathcal{P}(T)$ be two sets of sets. The *cross-union* of $\mathcal{V}$ and $\mathcal{W}$ is defined as $\mathcal{V} \uplus \mathcal{W} := \{V \cup W \mid V \in \mathcal{V}, W \in \mathcal{W}\}$. We abbreviate the cross-union of a set $\mathcal{W}$ with itself by $\mathcal{W}^{\times 2} := \mathcal{W} \uplus \mathcal{W}$.

Theorem 7. *Let $\Sigma \not\models c\,\langle\emptyset\rangle$. There exists a pre-Armstrong table for (T, T_S, Σ) iff there exists a set $\mathcal{W} \subseteq \mathcal{P}(T \setminus T_S)$ with the following properties:*

i) Every element of $\mathcal{W}^{\times 2}$ forms a weak anti-key.

ii) For every maximal weak anti-key $\neg_c\langle X\rangle \in \mathcal{A}^w_{max}$ there exists $Y \in \mathcal{W}^{\times 2}$ with $Y \cap X' \neq \emptyset$ for every possible key $p\,\langle X'\rangle \in \Sigma$ with $X' \subseteq X$.

There exists an Armstrong table for (T, T_S, Σ) iff i) and ii) hold as well as

iii) $\bigcup \mathcal{W} = T \setminus T_S$.

We have exploited Theorem 7 to devise a *general* construction of (pre-)Armstrong tables whenever they exist. In the construction, every maximal anti-key is represented by two new tuples. Strong anti-keys are represented by two tuples that strongly agree on the anti-key, while weak anti-keys are represented by two tuples that weakly agree on some suitable attributes of the anti-key as determined by the set $\mathcal{W}$ from Theorem 7. Finally, the set $\mathcal{W}$ permits us to introduce $\perp$ in nullable columns that do not yet feature an occurrence of $\perp$. For the construction we assume without loss of generality that all attributes have integer domains.

Construction 1 (Armstrong Table). Let $\mathcal{W} \subseteq \mathcal{P}(T \setminus T_S)$ satisfy conditions *i)* and *ii)* of Theorem 7. We construct an instance I over (T, T_S) as follows.

I) For every strong anti-key $\neg_p\langle X\rangle \in \mathcal{A}^s_{max}$ add tuples $t^s_X, t^{s\prime}_X$ to I with

$$t^s_X[X] = (i,\ldots,i) \qquad t^{s\prime}_X[X] = (i,\ldots,i)$$
$$t^s_X[T \setminus X] = (j,\ldots,j) \qquad t^{s\prime}_X[T \setminus X] = (k,\ldots,k)$$

where i, j, k are distinct integers not used previously.

II) For every weak anti-key $\neg_c\langle X\rangle \in \mathcal{A}^w_{max}$ we add tuples $t^w_X, t^{w\prime}_X$ to I with

$$t^w_X[X \setminus Y_1] = (i,\ldots,i) \qquad t^{w\prime}_X[X \setminus Y_2] = (i,\ldots,i)$$
$$t^w_X[X \cap Y_1] = (\perp,\ldots,\perp) \qquad t^{w\prime}_X[X \cap Y_2] = (\perp,\ldots,\perp)$$
$$t^w_X[T \setminus X] = (j,\ldots,j) \qquad t^{w\prime}_X[T \setminus X] = (k,\ldots,k)$$

where $Y_1, Y_2 \in \mathcal{W}$ meet the condition for $Y = Y_1 \cup Y_2$ in *ii)* of Theorem 7, and i, j, k are distinct integers not used previously.

III) If condition *iii)* of Theorem 7 also holds for $\mathcal{W}$, then for every $A \in T \setminus T_S$ for which there is no t in I with $t[A] = \perp$, we add a tuple t_A to I with

$$t_A[T \setminus A] = (i,\ldots,i) \qquad t_A[A] = \perp$$

where i is an integer not used previously.

Indeed, Construction 1 yields a (pre-)Armstrong table whenever it exists.

Theorem 8. *Let $\Sigma \not\models c\,\langle\emptyset\rangle$ and I constructed via Construction 1. Then I is a pre-Armstrong table over (T, T_S, Σ). If condition iii) of Theorem 7 holds for $\mathcal{W}$, then I is an Armstrong table.*

We illustrate Construction 1 on our running example and show how Table 1 can be derived from it.

Example 11. *Consider our running example where $T = \{article, author, journal\}$, $T_S = \{article, author\}$ and Σ consists of the certain key $c\,\langle article, journal\rangle$, the certain key $c\,\langle author, journal\rangle$ and the possible key $p\,\langle journal\rangle$. This gives us the maximal strong and weak anti-keys*

$$\mathcal{A}_{max} = \{\neg_p\langle author, article\rangle, \neg_c\langle journal\rangle\}$$

with the set $\mathcal{W} = \{journal\}$ meeting the conditions of Theorem 7. Now Construction 1 produces the Armstrong table

article	*author*	*journal*
0	0	0
0	0	1
1	1	$\perp$
2	2	$\perp$

.

In this specific case, we can remove either the third or the fourth tuple. While those two tuples have weak agree set $\{journal\}$, both of them have the same weak agree set with the first and the second tuple, too. After removal of the third or fourth tuple and suitable substitution we obtain Table 1.

The characterization of Theorem 7 is difficult to test in general, due to the large number of candidate sets $\mathcal{W}$. However, there are some cases where testing becomes simple.

Theorem 9. *Let $\Sigma \not\models c\,\langle\emptyset\rangle$.*

i) If $\Sigma \not\models c\,\langle X\rangle$ for every $X \subseteq T \setminus T_S$ then there exists an Armstrong table for (T, T_S, Σ).

ii) If $\Sigma \models c\,\langle X\rangle$ for any $X \subseteq T \setminus T_S$ with $|X| \leq 2$ then there does not exist an Armstrong table for (T, T_S, Σ).

4.4 Computational Characterization

We present a worst-case double exponential time algorithm for computing Armstrong tables whenever possible. Our experiments in Section 6 show that these worst cases do not arise in practice, and our computations are very efficient. For example, our algorithm requires milliseconds when brute force approaches would require $2^{2^{20}}$ operations. While the exact complexity of the Armstrong table existence problem remains an open question, we establish results which suggest that the theoretical worst-case time bound is difficult to improve upon.

Algorithm. Our goal is to compute the set $\mathcal{W}$ of Theorem 7 whenever it exist. For this purpose, we first observe that maximal anti-keys can be constructed using transversals.

Lemma 1. *Let Σ^P, Σ^C be the sets of possible and certain keys in Σ. For readability we will identify attribute sets with the keys or anti-keys induced by them. Then*

$$\mathcal{A}^s_{max} = \{\, X \in \overline{Tr(\Sigma^C \cup \Sigma^P)} \mid$$
$$\neg(X \subseteq T_S \wedge \exists Y \in \mathcal{A}^w_{max}.\, X \subset Y)\,\}$$
$$\mathcal{A}^w_{max} = \overline{Tr(\Sigma^C \cup \{X \in \Sigma^P \mid X \subseteq T_S\})} \setminus \mathcal{A}^s_{max}$$

For computing $\mathcal{A}^s_{max}$ and $\mathcal{A}^w_{max}$ by Lemma 1, we note that anti-keys for which strictly larger weak anti-keys exist can never be set-wise maximal weak anti-keys. Hence, the set of all maximal weak anti-keys can be computed as

$$\mathcal{A}^w_{max} = \overline{Tr(\Sigma^C \cup \{X \in \Sigma^P \mid X \subseteq T_S\})} \setminus \overline{Tr(\Sigma^C \cup \Sigma^P)}$$

The difficult part in deciding existence of (and then computing) an Armstrong table given (T, T_S, Σ) is to check existence of (and construct) a set $\mathcal{W}$ meeting the conditions of Theorem 7, in cases where Theorem 9 does not apply. Blindly testing all subsets of $\mathcal{P}(T \setminus T_S)$ becomes infeasible as soon as $T \setminus T_S$ contains more than 4 elements (for 5 elements, up to $2^{32} \approx 4,000,000,000$ sets would need to be checked). To ease discussion we will rephrase condition ii) of Theorem 7. Let $\mathcal{W} \subseteq \mathcal{P}(T)$. We say that

- $\mathcal{W}$ *supports* $Y \subseteq T$ if $Y \subseteq Z$ for some $Z \in \mathcal{W}$.
- $\mathcal{W}$ $\vee$*-supports* $\mathcal{V} \subseteq \mathcal{P}(T)$ if $\mathcal{W}$ supports *some* $Y \in \mathcal{V}$.
- $\mathcal{W}$ $\wedge\vee$*-supports* $\mathcal{T} \subseteq \mathcal{P}(\mathcal{P}(T))$ if $\mathcal{W}$ $\vee$-supports *every* $\mathcal{V} \in \mathcal{T}$.

We write $Y \Subset \mathcal{W}$ to indicate that $\mathcal{W}$ supports Y.

Lemma 2. *Let $\mathcal{T}_X$ be the transversal set in $T \setminus T_S$ of all possible keys that are subsets of X, and $\mathcal{T}$ the set of all such transversals for all maximal weak antikeys:*

$$\mathcal{T}_X := Tr(\{X' \cap (T \setminus T_S) \mid p\langle X'\rangle \in \Sigma_{min} \wedge X' \subseteq X\})$$
$$\mathcal{T} := \{\mathcal{T}_X \mid X \in \mathcal{A}^w_{max}\}$$

Then condition ii) of Theorem 7 can be rephrased as follows:

ii') $\mathcal{W}^{\times 2}$ $\wedge\vee$-supports $\mathcal{T}$.

We propose the following: For each $\mathcal{T}_X \in \mathcal{T}$ and each minimal transversal $t \in \mathcal{T}_X$ we generate all non-trivial bi-partitions $\mathcal{B}_\mathcal{X}$ (or just a trivial partition for transversals of cardinality < 2). We then add to $\mathcal{W}$ one such bi-partition for every $\mathcal{T}_X$ to ensure condition ii), and combine them with all single-attribute sets $\{A\} \subseteq T \setminus T_S$ to ensure condition iii). This is done for every possible combination of bipartitions until we find a set $\mathcal{W}$ that meets condition i), or until we have tested them all. We then optimize this strategy as follows: If a set $\mathcal{P}_X$ is already $\vee$-supported by $\mathcal{W}^{\times 2}$ (which at the time of checking will contain only picks for some sets $\mathcal{T}_X$), we may remove $\mathcal{P}_X$ from further consideration, as long as we keep all current picks in $\mathcal{W}$. In particular, since all single-attribute subsets of $T \setminus T_S$ are added to $\mathcal{W}$, we may ignore all $\mathcal{P}_X$ containing a set of size 2 or less. We give the algorithm thus developed in pseudo-code as Algorithm 1.

Theorem 10. *Algorithm Armstrong-Set is correct.*

We illustrate the construction with an example.

Example 12. *Let $(T, T_S) = (ABCDE, \emptyset)$ and*

$$\Sigma = \left\{ \begin{array}{l} p\langle A\rangle, p\langle B\rangle, p\langle CD\rangle, \\ c\langle ABE\rangle, c\langle ACE\rangle, c\langle ADE\rangle, c\langle BCE\rangle \end{array} \right\}$$

Neither condition of Theorem 9 is met, so Algorithm Armstrong-Set initializes/computes $\mathcal{W}$, $\mathcal{A}^w_{max}$ and $\mathcal{T}$ as

$$\begin{aligned} \mathcal{W} &= \{A, B, C, D, E\} \\ \mathcal{A}^w_{max} &= \{ABCD, AE, BDE, CDE\} \\ \mathcal{T} &= \{\{ABC, ABD\}\} \end{aligned}$$

Then, in method Extend-Support, $\mathcal{T}_x = \{ABC, ABD\}$ which is not $\vee$-supported by $\mathcal{W}^{\times 2}$. The non-trivial bi-partitions (Y_1, Y_2) of ABC are

$$\{(A, BC), (AC, B), (AB, C)\}.$$

None of these are suitable for extending $\mathcal{W}$, as the extension $(\mathcal{W} \cup \{Y_1, Y_2\})^{\times 2}$ contains the certain keys BCE, ACE and ABE, respectively. The non-trivial bi-partitions of the second set ABD are $\{(A, BD), (AD, B), (AB, D)\}$. While (AD, B) and (AB, D) are again unsuitable, (A, BD) can be used to extend $\mathcal{W}$ to the Armstrong set

$$\begin{aligned} \mathcal{W}' &= \textit{Extend-Support}(\{A, BD, C, E\}, \emptyset) \\ &= \{A, BD, C, E\} \end{aligned}$$

If we add $c\langle BDE\rangle$ to the sample schema, then (A, BD) becomes unsuitable as well, so that no Armstrong set exists.

Algorithm 1 Armstrong-Set

```
Input: $T, T_S, \Sigma$
Output: $\mathcal{W} \subseteq \mathcal{P}(T \setminus T_S)$ meeting conditions i) to iii) of
    Theorem 7 if such $\mathcal{W}$ exists, $\bot$ otherwise
 1: if $\nexists c\langle X\rangle \in \Sigma$ with $X \subseteq T \setminus T_S$ then
 2:    return $\{T \setminus T_S\}$
 3: if $\exists c\langle X\rangle \in \Sigma$ with $X \subseteq T \setminus T_S$ and $|X| \leq 2$ then
 4:    return $\bot$
 5: $\mathcal{W} := \{\{A\} \mid A \in T \setminus T_S\}$
 6: $\mathcal{A}^w_{max} := \overline{Tr(\Sigma^C \cup \{X \in \Sigma^P \mid X \subseteq T_S\})} \setminus \overline{Tr(\Sigma^C \cup \Sigma^P)}$
 7: $\mathcal{T} := \{\ Tr(\{X' \cap (T \setminus T_S) \mid p\langle X'\rangle \in \Sigma_{min} \wedge X' \subseteq X\}) \mid$
          $X \in \mathcal{A}^w_{max}\ \}$
 8: $\mathcal{T} := \mathcal{T} \setminus \{\mathcal{T}_X \in \mathcal{T} \mid \exists Y \in \mathcal{T}_X.\ |Y| \leq 2\}$
 9: return Extend-Support($\mathcal{W}, \mathcal{T}$)

Subroutine Extend-Support($\mathcal{W}, \mathcal{T}$)
Input: $\mathcal{W} \subseteq \mathcal{P}(T \setminus T_S)$ meeting conditions i) and iii) of
    Theorem 7, $\mathcal{T} \subseteq \mathcal{P}(\mathcal{P}(T \setminus T_S))$
Output: $\mathcal{W}' \supseteq \mathcal{W}$ meeting conditions i) and iii) of Theorem
    7 such that $\mathcal{W}'^{\times 2}$ $\wedge\vee$-supports $\mathcal{T}$ if such $\mathcal{W}'$ exists, $\bot$
    otherwise
10: if $\mathcal{T} = \emptyset$ then
11:    return $\mathcal{W}$
12: $\mathcal{T} := \mathcal{T} \setminus \{\mathcal{T}_X\}$ for some $\mathcal{T}_X \in \mathcal{T}$
13: if $\mathcal{W}^{\times 2}$ $\vee$-supports $\mathcal{T}_X$ then
14:    return Extend-Support($\mathcal{W}, \mathcal{T}$)
15: for all $Y \in \mathcal{T}_X$ do
16:    for all non-trivial bipartitions $Y = Y_1 \cup Y_2$ do
17:       if $(\mathcal{W} \cup \{Y_1, Y_2\})^{\times 2}$ contains no certain key then
18:          $\mathcal{W}' :=$ Extend-Support($\mathcal{W} \cup \{Y_1, Y_2\}, \mathcal{T}$)
19:          if $\mathcal{W}' \neq \bot$ then
20:             return $\mathcal{W}'$
21: return $\bot$
```

Hardness in theory. The main complexity of constructing Armstrong relations for keys in the relational model goes into the computation of all anti-keys. This is just the starting point for Algorithm 1, line 6. When computing the possible combinations of bi-partitions in line 16, Algorithm 1 becomes worst-case double exponential. We provide evidence that suggests this worst-case bound may be difficult to improve upon. The key/anti-key satisfiability problem asks: Given a schema (T, T_S, Σ) and a set $\mathcal{A}^w \subseteq \mathcal{P}(T)$ of weak anti-keys, does there exist a table I over (T, T_S, Σ) such that every element of $\mathcal{A}^w$ is a weak anti-key for I?

The pre-Armstrong existence problem can be reduced to the key/anti-key satisfiability problem by computing the set of maximal weak anti-keys w.r.t. Σ. We followed this approach in Algorithm 1 that computes a set $\mathcal{W}$ of Theorem 7 whenever such a set exists. Therefore, by showing key/anti-key satisfiability to be NP-complete, we show that any attempt to decide Armstrong existence more efficiently will likely need to take a very different route. As $\mathcal{W}$ can be exponential in the size of Σ, any such approach must not compute $\mathcal{W}$ at all. The NP-hard problem we reduce to the key/anti-key satisfiability problem is *monotone 1-in-3 SAT* [39], which asks: Given a set of 3-clauses without negations, does there exist a truth assignment such that every clause contains exactly one true literal?

THEOREM 11. *The key/anti-key satisfiability problem is NP-complete.*

Despite the poor theoretical worst-case bounds of the problem, our experiments in Section 6 show that our algorithm is typically very efficient.

5. ENFORCING CERTAIN KEYS

In practice, an efficient enforcement of certain keys requires index structures. Finding suitable index structures is non-trivial as weak similarity is not transitive. Hence, classical indices will not work directly. Nevertheless we present an index scheme, based on multiple classical indices, which allows us to check certain keys efficiently, provided there are few nullable key attributes. While an efficient index scheme seems elusive for larger sets of nullable attributes, our experiments from Section 6 suggest that most certain keys in practice have few nullable attributes.

Let (T, T_S) be a table schema and $X \subseteq T$. A *certain-key-index* for $c\langle X\rangle$ consists of a collection of indices $\mathfrak{I}_Y$ on subsets Y of X which include all `NOT NULL` attributes:

$$\mathfrak{I}_{c\langle X\rangle} := \{\mathfrak{I}_Y \mid X \cap T_S \subseteq Y \subseteq X\}$$

Here we treat $\perp$ as regular value for the purpose of indexing, i.e., we do index tuples with $\perp$ "values". When indexing a table I, each tuple in I is indexed in each $\mathfrak{I}$.

Obviously, $|\mathfrak{I}_{c\langle X\rangle}| = 2^n$, where $n := |X \setminus T_S|$, which makes maintenance efficient only for small n. When checking if a tuple exists that is weakly similar to some given tuple, we only need to consult a single index, but within that index we must perform up to 2^n lookups.

THEOREM 12. *Let t be a tuple on (T, T_S) and $\mathfrak{I}_{c\langle X\rangle}$ a certain-key-index for table I over (T, T_S). Define*

$$K := \{A \in X \mid t[A] \neq \perp\} .$$

Then the existence of a tuple in I weakly similar to t can be checked with 2^k lookups in $\mathfrak{I}_K$, where $k := |K \setminus T_S|$.

As $|K \setminus T_S|$ is bounded by $|X \setminus T_S|$, lookup is efficient whenever indexing is efficient.

EXAMPLE 13. *Consider schema $(ABCD, A, \{c\langle ABC\rangle\})$ with table I over it:*

I =

A	B	C	D
1	$\perp$	$\perp$	1
2	2	$\perp$	2
3	$\perp$	3	$\perp$
4	4	4	$\perp$

The certain-key-index $\mathfrak{I}_{c\langle ABC\rangle}$ for $c\langle ABC\rangle$ consists of:

$\mathfrak{I}_A$

A
1
2
3
4

$\mathfrak{I}_{AB}$

A	B
1	$\perp$
2	2
3	$\perp$
4	4

$\mathfrak{I}_{AC}$

A	C
1	$\perp$
2	$\perp$
3	3
4	4

$\mathfrak{I}_{ABC}$

A	B	C
1	$\perp$	$\perp$
2	2	$\perp$
3	$\perp$	3
4	4	4

These tables represent attribute values that we index by, using any standard index structure such as B-trees. When checking whether tuple $t := (2, \perp, 3, 4)$ is weakly similar on ABC to some tuple $t' \in I$ (and thus violating $c\langle ABC\rangle$ when inserted), we perform lookup on $\mathfrak{I}_{AC}$ for tuples t' with $t'[AC] \in \{(2, 3), (2, \perp)\}$.

6. EXPERIMENTS

We conducted several experiments to evaluate various aspects of our work. Firstly, we discovered possible and certain keys in publicly available databases. Secondly, we tested our algorithms for computing and deciding the existence of Armstrong tables. Lastly, we considered storage space and time requirements for our index scheme. We used the following data sets for our experiments: i) GO-termdb (Gene Ontology) at `geneontology.org/`, ii) IPI (International Protein Index) at `ebi.ac.uk/IPI`, iii) LMRP (Local Medical Review Policy) from `cms.gov/medicare-coverage-database/`, iv) PFAM (protein families) at `pfam.sanger.ac.uk/`, and v) RFAM (RNA families) at `rfam.sanger.ac.uk/`. These were chosen for their availability in database format.

6.1 Key Discovery

Examining the schema definition does not suffice to decide what types of key constraints hold or should hold on a database. Certain keys with $\perp$ occurrences cannot be expressed in current SQL databases, so would be lost. Even constraints that could and should be expressed are often not declared. Furthermore, even if `NOT NULL` constraints are declared, one frequently finds that these are invalid, resulting in work-arounds such as empty strings. We therefore mined data tables for possible and certain keys, with focus on finding certain keys containing $\perp$ occurrences. In order to decide whether a column is `NOT NULL`, we ignored any schema-level declarations, and instead tested whether a column contained $\perp$ occurrences. To alleviate string formatting issues (such as "Zwieb C." vs "Zwieb C.;") we normalized strings by trimming non-word non-decimal characters, and interpreting the empty string as $\perp$. This pre-processing was necessary because several of our test data sets were available only in CSV format, where $\perp$ occurrences would have been exported as empty strings. Tables containing less than two tuples were ignored. In the figures reported, we exclude tables `pfamA` and `lcd` from the PFAM and LMRP data sets, as they contain over 100,000 (`pfamA`) and over 2000 (`lcd`) minimal keys, respectively, almost all of which appear to be 'by chance', and thus would distort results completely. Table 5 lists the number of minimal keys of each type discovered in the 130 tables. We distinguish between possible and certain keys with `NULL` columns, and keys not containing `NULL` columns. For the latter, possible and certain keys coincide.

Two factors are likely to have a significant impact on the figures in Table 5. First, constraints may only hold accidentally, especially when the tables examined are small. For example, Table 1 of the RFAM data set satisfies $c\langle title, journal\rangle$ and $c\langle author, journal\rangle$, with `NULL` column *journal*. Only

Key Type	Occurrences
Certain Keys with `NULL`	43
Possible Keys with `NULL`	237
Keys without `NULL`	87

Table 5: Keys found by type

the first key appears to be sensible. Second, constraints that should sensibly hold may well be violated due to lack of enforcement. Certain keys with $\perp$ occurrences, which cannot easily be expressed in existing systems, are likely to suffer from this effect more than others. We thus consider our results qualitative rather than quantitative. Still, they indicate that certain keys do appear in practice, and may benefit from explicit support by a database system.

6.2 Armstrong Tables

We applied Algorithm 1 and Construction 1 to compute Armstrong tables for the 130 tables we mined possible and certain keys for. As all certain keys contained some `NOT NULL` column, Theorem 9 applied in all cases, and each Armstrong table was computed within a few milliseconds. Each Armstrong table contained only 7 tuples on average.

We also tested our algorithms against 1 million randomly generated table schemas and sets of keys. Each table contained 5-25 columns, with each column having a 50% chance of being `NOT NULL`, and 5-25 keys of size 1-5, with equal chance of being possible or certain. To avoid overly silly examples, we removed certain keys with only 1 or 2 `NULL` attributes and no `NOT NULL` attributes. Hence, case ii) of Theorem 9 never applies.

We found that in all but 85 cases, an Armstrong table existed, with average computation time again in the order of a few milliseconds. However, note that for larger random schemas, transversal computation can become a bottleneck.

Together these results support our intuition that although Algorithm 1 is worst-case double exponential, such cases need to be carefully constructed and arise neither in practice nor by chance (at least not frequently).

6.3 Indexing

The efficiency of our index scheme for certain keys with $\perp$ occurrences depends directly on the number of `NULL` attributes in the certain key. Thus the central question becomes how many `NULL` attributes occur in certain keys in practice. For the data sets in our experiments, having 43 certain keys with $\perp$ occurrences, the distribution of the number of these occurrences is listed in Table 6.

#NULLs	Frequency
1	39
2	4

Table 6: `NULL` columns in certain keys

Therefore, certain keys with $\perp$ occurrences contain mostly only a single attribute on which $\perp$ occurs (requiring 2 standard indices), and never more than two attributes (requiring 4 standard indices). Assuming these results generalize to other data sets, we conclude that certain key with $\perp$ occurrences can be enforced efficiently in practice.

We used triggers to enforce certain keys under different combinations of B-tree indices, and compared these to the enforcement of the corresponding primary keys. The experiments were run in MySQL version 5.6 on a Dell Latitude E5530, Intel core i7, CPU 2.9GHz with 8GB RAM on a 64-bit operating system. For all experiments the schema was $(T = ABCDE, T_S = A)$ and the table I over T contained 100M tuples. In each experiment we inserted 10,000 times one tuple and took the average time to perform this operation. This includes the time for maintaining the index structures involved. We enforced $c\langle X\rangle$ in the first experiment for $X = AB$, in the second experiment for $X = ABC$ and in the third experiment for $X = ABCD$, incrementing the number of `NULL` columns from 1 to 3. The distribution of permitted $\perp$ occurrences was evenly spread amongst the 100M tuples, and also amongst the 10,000 tuples to be inserted. Altogether, we run each of the experiments for 3 index structures: i) $\mathfrak{I}_X$, ii) $\mathfrak{I}_{T_S}$, iii) $\mathfrak{I}_{c\langle X\rangle}$. The times were compared against those achieved under declaring a primary key $PK(X)$ on X, where we had 100M X-total tuples. Our results are shown in Table 7. All times are in milliseconds.

Index	$T = ABCDE, T_S = A$		
	$X = AB$	$X = ABC$	$X = ABCD$
$PK(X)$	0.451	0.491	0.615
$\mathfrak{I}_X$	0.764	0.896	0.977
$\mathfrak{I}_{T_S}$	0.723	0.834	0.869
$\mathfrak{I}_{c\langle X\rangle}$	0.617	0.719	1.143

Table 7: Average Times to Enforce Keys on X

Hence, certain keys can be enforced efficiently as long as we involve the columns in T_S in some index. Just having $\mathfrak{I}_{T_S}$ ensures a performance similar to that of the corresponding primary key. Indeed, the `NOT NULL` attributes of a certain key suffice to identify most tuples uniquely. Our experiments confirm these observations even for certain keys with three `NULL` columns, which occur rarely in practice. Of course, $\mathfrak{I}_{T_S}$ cannot guarantee the efficiency bounds established for $\mathfrak{I}_{c\langle X\rangle}$ in Theorem 12. We stress the importance of indexing: enforcing $c\langle X\rangle$ on our data set without an index resulted in a performance loss in the order of 10^4.

7. CONCLUSION AND FUTURE WORK

Primary key columns must not feature null markers and therefore oppose the requirements of modern applications, inclusive of high volumes of incomplete data. We studied keys over SQL tables in which the designated null marker may represent missing or unknown data. Both interpretations can be handled by a possible world semantics. A key is possible (certain) to hold on a table if some (every) possible world of the table satisfies the key. Possible keys capture SQL's `unique` constraint, and certain keys generalize SQL's primary keys by permitting null markers in key columns. We established solutions to several computational problems related to possible and certain keys under `NOT NULL` constraints. These include axiomatic and linear-time algorithmic characterizations of their implication problem, minimal representations of keys, discovery of keys from a given table, and structural and computational properties of Armstrong tables. Experiments confirm that our solutions work effectively and efficiently in practice. This also applies to enforcing certain keys, by utilizing known index structures. Our findings from public data confirm that certain keys have only few columns with null marker occurrences. Certain keys thus achieve the goal of Codd's rule for entity integrity while accommodating the requirements of modern applications. This distinct advantage over primary keys comes only at a small price in terms of update performance.

Our research encourages various future work. It will be both interesting and challenging to revisit different notions of keys, including our possible and certain keys, in the presence of finite domains. The exact complexity of deciding the existence of Armstrong tables should be determined. It is worth investigating optimizations to reduce the time complexity of computing Armstrong tables, and their size. Evidently, the existence and computation of Armstrong relations for sets of weak and strong functional dependencies requires new attention [29]. Approximate and scalable discovery is an important problem as meaningful keys may be violated. This line of research has only been started for total [22, 40] and possible keys [22]. Other index schemes may prove valuable to enforce certain keys. All application areas of keys deserve new attention in the light of possible and certain keys. It is interesting to combine possible and certain keys with possibilistic or probabilistic keys [8, 26].

8. ACKNOWLEDGEMENTS

We thank Mozhgan Memari for carrying out the experiments for key enforcement. We thank Arnaud Durand and Georg Gottlob for their comments on an earlier draft. This research is partially supported by the Marsden Fund Council from New Zealand Government funding, by the Natural Science Foundation of China (Grant No. 61472263) and the Australian Research Council (Grants No. DP140103171).

9. REFERENCES

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[2] S. Abiteboul and V. Vianu. Transactions and integrity constraints. In *PODS*, pages 193–204, 1985.

[3] M. Arenas, L. E. Bertossi, and J. Chomicki. Consistent query answers in inconsistent databases. In *SIGMOD*, pages 68–79, 1999.

[4] P. Bailis, A. Ghodsi, J. M. Hellerstein, and I. Stoica. Bolt-on causal consistency. In *SIGMOD*, pages 761–772, 2013.

[5] C. Beeri, M. Dowd, R. Fagin, and R. Statman. On the structure of Armstrong relations for functional dependencies. *J. ACM*, 31(1):30–46, 1984.

[6] L. E. Bertossi. *Database Repairing and Consistent Query Answering*. Morgan & Claypool Publishers, 2011.

[7] J. Biskup. *Security in Computing Systems - Challenges, Approaches and Solutions*. Springer, 2009.

[8] P. Brown and S. Link. Probabilistic keys for data quality management. In *CAiSE*, pages 118–132, 2015.

[9] A. Calì, D. Calvanese, and M. Lenzerini. Data integration under integrity constraints. In *Seminal Contributions to Information Systems Engineering*, pages 335–352. 2013.

[10] D. Calvanese, W. Fischl, R. Pichler, E. Sallinger, and M. Simkus. Capturing relational schemas and functional dependencies in RDFS. In *AAAI*, pages 1003–1011, 2014.

[11] E. F. Codd. *The Relational Model for Database Management, Version 2*. Addison-Wesley, 1990.

[12] S. Doherty. The future of enterprise data. `http://insights.wired.com/profiles/blogs/the-future-of-enterprise-data#axzz2owCB8FFn`, 2013.

[13] T. Eiter and G. Gottlob. Identifying the minimal transversals of a hypergraph and related problems. *SIAM J. Comput.*, 24(6):1278–1304, 1995.

[14] R. Fagin. A normal form for relational databases that is based on domains and keys. *ACM Trans. Database Syst.*, 6(3):387–415, 1981.

[15] R. Fagin. Horn clauses and database dependencies. *J. ACM*, 29(4):952–985, 1982.

[16] R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data exchange: semantics and query answering. *Theor. Comput. Sci.*, 336(1):89–124, 2005.

[17] W. Fan, F. Geerts, and X. Jia. A revival of constraints for data cleaning. *PVLDB*, 1(2):1522–1523, 2008.

[18] S. Hartmann, M. Kirchberg, and S. Link. Design by example for SQL table definitions with functional dependencies. *VLDB J.*, 21(1):121–144, 2012.

[19] S. Hartmann, U. Leck, and S. Link. On Codd families of keys over incomplete relations. *Comput. J.*, 54(7):1166–1180, 2011.

[20] S. Hartmann and S. Link. Efficient reasoning about a robust XML key fragment. *ACM Trans. Database Syst.*, 34(2), 2009.

[21] S. Hartmann and S. Link. The implication problem of data dependencies over SQL table definitions. *ACM Trans. Database Syst.*, 37(2):13, 2012.

[22] A. Heise, Jorge-Arnulfo, Quiane-Ruiz, Z. Abedjan, A. Jentzsch, and F. Naumann. Scalable discovery of unique column combinations. *PVLDB*, 7(4):301–312, 2013.

[23] I. Ileana, B. Cautis, A. Deutsch, and Y. Katsis. Complete yet practical search for minimal query reformulations under constraints. In *SIGMOD*, pages 1015–1026, 2014.

[24] I. F. Ilyas, V. Markl, P. J. Haas, P. Brown, and A. Aboulnaga. CORDS: automatic discovery of correlations and soft functional dependencies. In *SIGMOD*, pages 647–658, 2004.

[25] A. K. Jha, V. Rastogi, and D. Suciu. Query evaluation with soft keys. In *PODS*, pages 119–128, 2008.

[26] H. Köhler, U. Leck, S. Link, and H. Prade. Logical foundations of possibilistic keys. In *JELIA*, pages 181–195, 2014.

[27] H. Köhler, U. Leck, S. Link, and X. Zhou. Possible and certain SQL keys. CDMTCS-452, The University of Auckland, 2013.

[28] P. Koutris and J. Wijsen. The data complexity of consistent query answering for self-join-free conjunctive queries under primary key constraints. In *PODS*, pages 17–29, 2015.

[29] M. Levene and G. Loizou. Axiomatisation of functional dependencies in incomplete relations. *Theor. Comput. Sci.*, 206(1-2):283–300, 1998.

[30] M. Levene and G. Loizou. A generalisation of entity and referential integrity. *ITA*, 35(2):113–127, 2001.

[31] Y. E. Lien. On the equivalence of database models. *J. ACM*, 29(2):333–362, 1982.

[32] J. Liu, J. Li, C. Liu, and Y. Chen. Discover dependencies from data - A review. *IEEE TKDE*, 24(2):251–264, 2012.

[33] H. Mannila and K.-J. Räihä. *Design of Relational Databases*. Addison-Wesley, 1992.

[34] J. Melton. ISO/IEC 9075-2: 2003 (SQL/foundation). ISO standard, 2003.

[35] F. Naumann. Data profiling revisited. *SIGMOD Record*, 42(4):40–49, 2013.

[36] R. Pochampally, A. D. Sarma, X. L. Dong, A. Meliou, and D. Srivastava. Fusing data with correlations. In *SIGMOD*, pages 433–444, 2014.

[37] K. A. Ross, D. Srivastava, and S. Sudarshan. Materialized view maintenance and integrity constraint checking: Trading space for time. In *SIGMOD*, pages 447–458, 1996.

[38] B. Saha and D. Srivastava. Data quality: The other face of big data. In *ICDE*, pages 1294–1297, 2014.

[39] T. J. Schaefer. The complexity of satisfiability problems. In *STOC*, pages 216–226, 1978.

[40] Y. Sismanis, P. Brown, P. J. Haas, and B. Reinwald. GORDIAN: Efficient and scalable discovery of composite keys. In *VLDB*, pages 691–702, 2006.

[41] B. Thalheim. On semantic issues connected with keys in relational databases permitting null values. *Elektr. Informationsverarb. Kybern.*, 25(1/2):11–20, 1989.

[42] C. Zaniolo. Database relations with null values. *J. Comput. System Sci.*, 28(1):142–166, 1984.

[43] K. Zellag and B. Kemme. Consad: a real-time consistency anomalies detector. In *SIGMOD*, pages 641–644, 2012.