JCSE Journal of Computing Science and Engineering

Regular Paper

Journal of Computing Science and Engineering,
Vol. 6, No. 3, September 2012, pp. 193-206

Schema- and Data-driven Discovery of SQL Keys

Van Bao Tran Le
School of Information Management, The Victoria University of Wellington, Wellington, New Zealand **van.t.le@vuw.ac.nz**

Sebastian Link* **and Mozhgan Memari**
Department of Computer Science, The University of Auckland, Auckland, New Zealand
s.link@auckland.ac.nz, mmem525@aucklanduni.ac.nz

Abstract
Keys play a fundamental role in all data models. They allow database systems to uniquely identify data items, and therefore, promote efficient data processing in many applications. Due to this, support is required to discover keys. These include keys that are semantically meaningful for the application domain, or are satisfied by a given database. We study the discovery of keys from SQL tables. We investigate the structural and computational properties of Armstrong tables for sets of SQL keys. Inspections of Armstrong tables enable data engineers to consolidate their understanding of semantically meaningful keys, and to communicate this understanding to other stake-holders. The stake-holders may want to make changes to the tables or provide entirely different tables to communicate their views to the data engineers. For such a purpose, we propose data mining algorithms that discover keys from a given SQL table. We combine the key mining algorithms with Armstrong table computations to generate informative Armstrong tables, that is, key-preserving semantic samples of existing SQL tables. Finally, we define formal measures to assess the distance between sets of SQL keys. The measures can be applied to validate the usefulness of Armstrong tables, and to automate the marking and feedback of non-multiple choice questions in database courses.

Category: Smart and intelligent computing

Keywords: Algorithm; Complexity; Armstrong database; Key; Soundness; Completeness; Mining; SQL

I. INTRODUCTION

In databases, keys play a fundamental role in understanding both the structure and semantics. Given an SQL table schema, a key is a collection of columns whose values uniquely identify rows. That is, no two different rows have matching total values in each of the key columns. The concept of a key is essential for many other data models, including semantic models [1-4], object models [5], description logics [6], XML [7-9], RDF [10], and OWL [11].

The discovery of semantically meaningful SQL keys is a crucial task in many areas of modern data management, for example, in data modeling, database design, query optimization, indexing, and data integration [12]. This article is concerned with methods for semi-automated schema-driven and automated data-driven SQL key discovery.

There is great demand in industry for such methods, because they vastly simplify the job of the database administrator and thereby decrease the overall cost of database ownership. The discovery of composite keys is especially difficult, because the number of possible keys

Open Access http://dx.doi.org/10.5626/JCSE.2012.6.3.193 http://jcse.kiise.org

Received 13 June 2012, **Accepted** 20 August 2012
*Corresponding Author

[†]An earlier version of this paper has been published at DASFAA 2012.

 pISSN: 1976-4677 eISSN: 2093-8020

Table 1. An Armstrong table for SCHEDULE

C_ID	Time	L_Name	Room
11301	Mon, 10 am	Ullman	Red
11301	Tue, 02 pm	Ullman	Red
78200	Mon, 10 am	Ullman	Red
99120	Wed, 04 pm	ni	ni

ni: no information.

Table 2. Another Armstrong table

C ID	Time	L_Name	Room
11301	Mon, 10 am	Ullman	Red
11301	Tue, 02 pm	Ullman	Red
78200	Mon, 10 am	Ullman	ni
99120	Mon, 10 am	ni	Red

ni: no information.

increases exponentially with the number of columns [3, 13]. Because of the industry demand, the goal is to provide practical algorithms that exhibit good typical case behavior. Industry-leading data modeling and design tools, such as the *ERwin* data modeler, emphasize the need for good test data in order to validate the semantics of the models they produce [14]. In our context, test data is considered good if it assists data engineers with the discovery of semantically meaningful keys. This calls for algorithms that produce test data which satisfy the keys currently perceived as semantically meaningful and violate the keys currently perceived as semantically meaningless.

A. Running Example

Consider a simple database that collects basic information about the weekly schedule of courses. That is, we have a schema SCHEDULE with columns *C_ID*, *L_Name*, as well as *Time* and *Room*. The schema stores the time (including the weekday) and room in which a lecturer (identified by their lecturer name *L_Name*) gives a course (identified by its *C_ID*). An SQL definition may be given as follows:

```
CREATE TABLE Schedule (
    C_ID CHAR[5],
    L_Name VARCHAR,
    Time CHAR[15],
    Room VARCHAR,
    PRIMARY KEY (C_ID, Time);)
```

The table schema specifies additional assertions. The primary key forces rows over SCHEDULE to be NOT NULL in the *C_ID* and *Time* columns, and to be unique on their {*C_ID*, *Time*} projections. That is, no two different rows must have matching values in both the *C_ID* column and the *Time* column. A team of data engineers may wonder if the semantics of the application domain has been captured by this SQL definition. They decide to generate some good test data to discuss their current understanding with the domain experts, who do not understand database terminology. Therefore, the data engineers produce the data sample in Table 1, where ni denotes occurrences of the *no information* null marker.

Table 3. Updated table

C ID	Time	L Name	Room
11301	Mon, 10 am	Ullman	Red
11301	Tue, 02 pm	Ullman	Red
78200	Mon, 10 am	**Fagin**	**Blue**
99120	Wed, 04 pm	ni	ni

ni: no information.

The domain experts express concern about rows 1 and 3: Ullman teaches both 11301 and 78200 in the red room on Monday at 10 am. After some discussion, the domain experts emphasize that different courses should not be taught by the same lecturer in the same room at the same time. As a consequence, the team of engineers decides to specify the uniqueness constraint (UC) *u*(*Time*, *L_Name*, *Room*). They produce the data sample in Table 2 to discuss their new understanding with the domain experts.

The domain experts direct the engineers' attention to rows 1 and 3 where Ullman teaches both 11301 and 78200 on Monday at 10 am. After exchanging some ideas, the experts agree that lecturers must not teach different courses at the same time.

Moreover, from rows 1 and 4 of the data sample in Table 2, experts notice that both 11301 and 78200 are taught on Monday at 10am in the red room. It turns out that no two different courses must be taught at the same time in the same room. Therefore, the UC *u*(*Time*, *L_Name*, *Room*) is replaced by the two stronger UCs *u*(*Time*, *L_Name*) and *u*(*Time*, *Room*). The example shows the potential benefit of investigating good sample data for the discovery of semantically meaningful SQL keys. As a revised SQL table schema, the team of data engineers specify

```
CREATE TABLE Schedule' (
    C_ID CHAR[5],
    L_Name VARCHAR,
    Time CHAR[15],
    Room VARCHAR,
    PRIMARY KEY (C_ID, Time),
    UNIQUE (L_Name, Time),
    UNIQUE (Room, Time);)
```

While this approach appears to be very beneficial, the

question remains what constitutes good test data, and how to create it (automatically). In some situations the domain experts may feel very confident and would like to have the opportunity to modify values in the test data to reflect their domain knowledge, or provide the entire test data themselves. In such cases, and other situations, the data engineers require automated means to discover a representation of the keys that are satisfied by the test data. For example, when the domain experts inspect the data sample in Table 1, they may simply suggest using the data sample in Table 3 instead, with the updated values indicated by bold font.

On input of this table, a constraint mining algorithm would return `NOT NULL` constraints on *C_ID* and *Time*, as well as the three UCs *u*(*C_ID*, *Time*), *u*(*L_Name*, *Time*), and *u*(*Room*, *Time*). This, again, leads to the definition of SCHEDULE'. The question remains how to discover the SQL keys that are satisfied by a given SQL table.

B. Contributions

In this paper we will establish detailed answers to the two questions above. As our first main contribution we discuss the schema-driven discovery of SQL keys that are semantically meaningful for a given application domain. For this purpose we investigate the well-known concept of Armstrong databases for the class of SQL keys. Armstrong databases formalize the concept of good test data in the sense that they satisfy the set Σ of keys currently perceived as semantically meaningful, and violate all keys that are not implied by Σ. We characterize when a given SQL table is Armstrong with respect to a given set Σ of SQL keys. This characterization allows us to establish an algorithm that generates good test data for arbitrary sets of SQL keys. While we demonstrate that the problem of computing such Armstrong tables is precisely exponential in the number of column headers, our algorithm produces an Armstrong table whose size is at most quadratic. We also show that there are situations where the size of the Armstrong table our algorithm produces is exponentially smaller than the size of the keys given. Both extreme cases result from very specific situations, which are very unlikely to occur in practice. Indeed, in most situations that do occur in practice, our algorithm will produce Armstrong tables whose size is in the same order of magnitude as the size of the keys given.

As a second main contribution we discuss the data-driven discovery of SQL keys. For this purpose we establish two algorithms that compute the set of minimal SQL keys satisfied by a given SQL table. While the problem generally requires exponential time, our algorithms show good best case behavior. As a third main contribution we combine the schema-driven and data-driven approach to increase the effectiveness of SQL key discovery. Given a real world data set, we apply the data-driven approach to identify the minimal SQL keys satisfied by this data set, and then apply the schema-driven approach to compute an Armstrong table for the set of these minimal keys. The Armstrong table is called informative as it contains only rows of the original data set, is much smaller in size, and satisfies the same SQL keys.

As the final contribution, we define formal measures that can be applied to 1) empirically validate the usefulness of our Armstrong tables for the acquisition of semantically meaningful SQL keys, and 2) automate the feedback and marking of database exam questions.

C. Organization

We summarize related work in Section II, and provide preliminary definitions in Section III. We investigate structural and computational properties of Armstrong tables for the class of SQL keys in Section IV. In Section V we study the SQL key mining problem. Section VI combines the data-driven and schema-driven approaches to the discovery of SQL keys. Our formal measures of usefulness and their applications are discussed in Section VII. Finally, in Section VIII, we conclude and briefly comment on future work.

II. RELATED WORK

Data dependencies have been studied thoroughly in the relational model of data, cf. [15-17]. Dependencies are essential to the design of the target database, the maintenance of the database during its lifetime, and all major data processing tasks [15, 17]. These applications also provide strong motivation for developing data mining algorithms to discover the data dependencies that are satisfied by a given database. Armstrong databases are a useful design aid for data engineers, which can help with the consolidation of data dependencies [18, 19] and schema mappings [20], the design of databases [21] and the creation of concise test data [22].

In the relational model of data, the class of keys is subsumed by the class of functional dependencies. The structural and computational properties of Armstrong relations have been investigated in the relational model of data for the class of keys [3, 23], and the more general class of functional dependencies [21, 24]. The mining of keys and functional dependencies in relations has also received considerable attention in the relational model [12, 25, 26]. The concept of informative Armstrong databases was introduced in [22] as semantic samples of existing databases.

One of the most important extensions of Codd's basic relational model [27] is incomplete information [28, 29]. This is mainly due to the high demand for the correct handling of such information in real-world applications. Approaches to deal with incomplete information comprise incomplete relations, or-relations or fuzzy relations. In this paper we focus on incomplete bags, and the most

popular interpretation of a null marker as "no information" [29, 30]. This is the general case of SQL tables where duplicate rows and null markers are permitted to occur in columns that are specified as null-able. Relations are idealized special SQL tables where no duplicate rows can occur and all columns are specified NOT NULL. Recently, Armstrong tables have been investigated for their combined class of SQL keys and functional dependencies [31]. In this article, we establish optimizations that arise from the focus on the sole class of SQL keys. This provides insight into the trade-off between the expressiveness of data dependency classes and the structural and computational properties of Armstrong tables. The insight can be used by data engineers to make a more informed decision about the complexity of the design process for the target database. For example, it may appear that the most important requirements of an application domain can already be captured by SQL keys alone, without the need for functional dependencies. In this case, engineers can utilize the algorithms developed in the current article, which result in computations that are more resource-efficient than those developed for the combined class of keys and functional dependencies [31]. Note that Hartmann et al. [1] studies Codd keys, which require tuples to have unique and total projections on key attributes. In contrast, the present article studies SQL keys, which require total tuples to have unique projections on key attributes. Hartmann et al. [1] does also not discuss any key mining algorithms or measures to assess the difference of key sets, nor does it discuss informative Armstrong databases. To the best of the authors' knowledge, no previous work has addressed the mining of keys from SQL tables. Our results reveal that the methods developed for the idealized special case of relations [26] can be adapted to the use for SQL tables. We are also unaware of any studies related to informative Armstrong databases except [22], measures that capture the difference between sets of keys, nor their utilization for assessing the usefulness of Armstrong tables or database exam questions.

The present article is an extension of the conference paper [32]. The present article contains all the proofs of our results as well as additional motivation and discussion. In addition, the concept of informative Armstrong tables was not discussed in the conference paper.

III. THE SQL TABLE MODEL

In this section we introduce the basic definitions necessary and sufficient for the development of our results in subsequent sections. In particular, we introduce a model that is very similar to SQL's data definition capabilities. The interpretation of null marker occurrences as "no information" follows the SQL interpretation, and the formal model introduced by Zaniolo [29]. The class of SQL keys under consideration is the exact class of uniqueness constraints defined in the SQL standard [33].

Let $\mathfrak{H} = \{H_1, H_2, ...\}$ be a countably infinite set of symbols, called *column headers* or *headers* for short. A *table schema* is a finite non-empty subset T of $\mathfrak{H}$. Each header H of a table schema T is associated with a countably infinite domain $dom(H)$ of the possible values that can occur in column H. To encompass partial information, every column can have a null marker, denoted by ni $\in dom(H)$. The intention of ni is to signify "no information".

For header sets X and Y, we may write XY for $X \cup Y$. If $X = \{H_1, \ldots, H_m\}$, then we may write $H_1 \cdots H_m$ for X. In particular, we may write simply H to represent the singleton $\{H\}$. A *row* over T (T-row or simply row, if T is understood) is a function $r : T \to \cup_{H \in T} dom(H)$ with $r(H) \in dom(H)$ for all $H \in T$. The null marker occurrence $r(H)$ = ni associated with a header H in a row r means that there is no information about $r(H)$. That is, $r(H)$ may not exist or $r(H)$ exists but is unknown. For $X \subseteq T$ let $r(X)$ denote the restriction of the row r over T to X. A *table* t over T is a finite multi-set of rows over T. For a row r over T and a set $X \subseteq T$, r is said to be X-total if for all $H \in X$, $r(H) \neq$ ni. Similarly, a table t over T is said to be X-total, if every row r of t is X-total. A table t over T is said to be a *total table* if it is T-total.

A UC over a table schema T is an expression $u(X)$ where $X \subseteq T$. A table t over T is said to *satisfy* the UC $u(X)$ over T, denoted by $\models_t u(X)$, if for all $r_1, r_2 \in t$, if $r_1(X) = r_2(X)$ and r_1, r_2 are X-total, then $r_1 = r_2$. The semantics is that of SQL's UC [33], and it reduces to that of a key for total tables [15].

Following Atzeni and Morfuni [30] a *null-free subschema* (NFS) over the table schema T is an expression $nfs(T_s)$ where $T_s \subseteq T$. The NFS $nfs(T_s)$ over T is satisfied by a table t over T, denoted by $\models_t nfs(T_s)$, if and only if t is T_s-total. SQL allows the specification of column headers as NOT NULL. NFSs occur in everyday database practice: the set of headers declared NOT NULL forms an NFS over the underlying table schema.

In schema design and maintenance data dependencies are normally specified as semantic constraints on the tables intended to be instances of the schema. During the design process or the lifetime of a database one usually needs to determine further dependencies, which are implied by the given ones. Let T be a table schema, $nfs(T_s)$ an NFS, and $\Sigma \cup \{\varphi\}$ be a set of UCs over T. We say that Σ *implies* φ in the presence of $nfs(T_s)$, denoted by $\Sigma \models_{T_s} \varphi$, if every T_s-total table t over T that satisfies Σ also satisfies φ. If Σ does not imply φ in the presence of $nfs(T_s)$ we may also write $\Sigma \not\models_{T_s} \varphi$. Let $\Sigma^*_{T_s} = \{\varphi \mid \Sigma \models_{T_s} \varphi\}$ be the semantic closure of Σ. If we do not want to emphasize the presence of the NFS $nfs(T_s)$ we may simply write $\Sigma \models \varphi$, $\Sigma \not\models \varphi$ or Σ^*, respectively. The next result explains why minimal UCs are important. Indeed, for a set $\Sigma \cup \{u(X)\}$ of UCs, and an NFS $nfs(T_s)$ over T we call $u(X)$ *minimal* if and only if $\Sigma \models_{T_s} u(X)$ and for all $u(Y)$ over T

where $\Sigma \models_{T_s} u(Y)$ and $Y \subseteq X$ we have $Y = X$.

THEOREM 1. *Let T be a table schema, $nfs(T_s)$ an NFS, and $\Sigma \cup \{\varphi\}$ be a set of UCs over T. Then, Σ implies φ in the presence of $nfs(T_s)$ if and only if there is some $u(Y) \in \Sigma$ such that $Y \subseteq X$.*

Proof. Suppose first that there is some $u(Y) \in \Sigma$ such that $Y \subseteq X$. We need to show that Σ implies φ in the presence of $nfs(T_s)$. Assume to the contrary that there is some T_s-total table t that satisfies Σ, but violates φ. Consequently, there are two different T_s-total rows $r_1, r_2 \in t$ such that $r_1(X) = r_2(X)$. Since $Y \subseteq X$ we conclude that t violates $u(Y) \in \Sigma$, a contradiction. Consequently, Σ implies φ in the presence of $nfs(T_s)$.

For the reverse direction we present the contraposition. That is, we assume that for all $u(Y) \in \Sigma$ we have $Y \not\subseteq X$, and show that under this assumption Σ does not imply φ in the presence of $nfs(T_s)$. For the latter, we construct a T_s-total table t that satisfies Σ but violates φ. Let t be the table over T consisting of two rows r_1 and r_2 over T where r_1 and r_2 are XT_s-total and $r_1(X) = r_2(X)$, and $r_1(H) = \texttt{ni} = r_2(H)$ for all $H \in T - XT_s$, and $r_1(H) \neq r_2(H)$ for all $H \in (T - X) \cap T_s$. The construction ensures that t violates $u(X)$. Let $u(Y) \in \Sigma$. We have assumed that $Y \not\subseteq X$, that is, $Y \cap (T - X) \neq \emptyset$. Again, the construction ensures that t satisfies $u(Y)$. This concludes the proof. □

In the relational model of data, keys enjoy a simple axiomatization [17] by two axioms. The set axiom says that for every relation schema R, the set R of all attributes forms a key. The superkey rule states that for every set K of attributes in R that forms a key over R, every superset $K' \subseteq R$ of K also forms a key over R. For SQL we have just shown that this axiomatization becomes even simpler. The proof of Theorem 1 has just shown that the superkey rule is sound and complete for the implication of SQL keys. In particular, the set axiom is no longer sound since we can have two duplicate tuples in a multiset.

EXAMPLE 1. We can capture the SQL table schema of the running example as the table schema SCHEDULE = *CTLR* with SCHEDULE_s = *CT*. Let Σ consist of the two UCs $u(CT)$ and $u(LTR)$. It follows that Σ does not imply any of the following UCs: $u(CLR)$, $u(LT)$ nor $u(TR)$. For instance, the SCHEDULE_s-total table in Table 2 satisfies Σ and violates every of the three UCs. As an application of Theorem 1 we see that neither of *CLR*, *LT* nor *TR* is a superset of *CT* or *LTR*. □

IV. SCHEMA-DRIVEN SQL KEY DISCOVERY

In this section we investigate the structural and computational properties of suitable data to test the semantic meaningfulness of uniqueness constraints over the SQL table schemata. For this purpose, we use Armstrong tables to formalize the notion of suitable test data. Having introduced the concepts of strong agree sets and anti-keys, we characterize when an arbitrarily given SQL table is Armstrong for an arbitrarily given set of uniqueness constraints. The characterization is then used to compute Armstrong tables. Finally, we derive results on the time and space complexity associated with the computation of Armstrong tables.

A. Key Concepts

The official concept of an Armstrong database was originally introduced by Fagin [16]. We require our tables to be Armstrong with respect to uniqueness constraints and the NFS. Intuitively, an Armstrong table satisfies the given constraints and violates the constraints in the given class that are not implied by the given constraints. This results in the following definition.

DEFINITION 1. *Let Σ be a set of UCs, and $nfs(T_s)$ an NFS over table schema T. A table t over T is said to be Armstrong for Σ and $nfs(T_s)$ if and only if i) t satisfies Σ, ii) t violates every UC $\varphi \notin \Sigma^*_{T_s}$, iii) t is T_s-total, and iv) for every $H \in T - T_s$, t is not H-total.* □

Through the use of Theorem 1, it is easy to see that a table t is Armstrong for Σ and $nfs(T_s)$ if and only if 1) for all $u(X)$ over T, t satisfies $u(X)$ if and only if there is some $u(Y) \in \Sigma$ such that $Y \subseteq X$, and 2) for all $nfs(T'_s)$ over T, t satisfies $nfs(T'_s)$ if and only if $T'_s \subseteq T_s$.

EXAMPLE 2. Let SCHEDULE = *CTLR* with SCHEDULE_s = *CT*. Let Σ consist of the two UCs $u(CT)$ and $u(LTR)$. The data sample in Table 2 is Armstrong for Σ and $nfs(\text{SCHEDULE}_s)$. For example, it violates the UCs: $u(CLR)$, $u(LT)$ and $u(TR)$, as well as every NFS $nfs(T'_s)$ where T'_s is not contained in SCHEDULE_s. □

A natural question to ask is how we can characterize the structure of tables that are Armstrong. With this in mind, we introduce the formal notion of strong agree sets for pairs of distinct rows, and tables.

DEFINITION 2. *For two rows r_1 and r_2 over table schema T where $r_1 \neq r_2$ we define the strong agree set of r_1 and r_2 as the set of all column headers over T on which r_1 and r_2 have a matching total value, i.e., $ag^s(r_1,r_2) = \{H \in T \mid r_1(H) = r_2(H)$ and $r_1(H) \neq \texttt{ni} \neq r_2(H)\}$. Furthermore, the strong agree set of a table t over table schema T is defined as $ag^s(t) = \{ag^s(r_1, r_2) \mid r_1, r_2 \in t \wedge r_1 \neq r_2\}$.*

EXAMPLE 3. Let SCHEDULE = *CTLR* with SCHEDULE_s = *CT*. Let Σ consist of the two UCs $u(CT)$ and $u(LTR)$. Let t denote the data sample in Table 2. The strong agree set of t consists of *CLR*, *LT*, *TR*, *L*, *R*, and *T*. □

For a table t to be Armstrong for Σ and $nfs(T_s)$, t must violate all UCs $u(X)$ not implied by Σ in the presence of $nfs(T_s)$. Instead of violating all UCs, it suffices to violate those ones that are maximal with the property that they

are not implied by Σ in the presence of $nfs(T_s)$. This motivates the following definition.

DEFINITION 3. *Let Σ be a set of UCs, and $nfs(T_s)$ an NFS over table schema T. The set Σ^{-1} of all anti-keys with respect to Σ and $nfs(T_s)$ is defined as* $\Sigma^{-1} = \{a(X) \mid X \subseteq T \wedge \Sigma \not\models_{T_s} u(X) \wedge \forall H \in (T - X)(\Sigma \models_{T_s} u(XH))\}$. □

Hence, an anti-key for Σ is given by a maximal set of column headers, which does not form a uniqueness constraint implied by Σ.

EXAMPLE 4. Let SCHEDULE = *CTLR* with SCHEDULE_s = *CT*. Let Σ consist of the two UCs $u(CT)$ and $u(LTR)$. The set Σ^{-1} of anti-keys with respect to Σ and SCHEDULE_s consists of $a(CLR)$, $a(LT)$ and $a(TR)$. □

B. Structure of Armstrong Tables

The concepts from the last sub-section are sufficient to establish a characterization of Armstrong tables for the class of UCs over SQL table schemata.

THEOREM 2. *Let Σ be a set of UCs, and $nfs(T_s)$ an NFS over the table schema T. For all tables t over T it holds that t is an Armstrong table for Σ and $nfs(T_s)$ if and only if the following three conditions are satisfied: for all $a(X) \in \Sigma^{-1}$ we have $X \in ag^s(t)$; for all $u(X) \in \Sigma$ and for all $Y \in ag^s(t)$ we have $X \not\subseteq Y$; total$(t) = \{H \in T \mid \forall r \in t(r(H) \neq \texttt{ni})\} = T_s$.*

Proof. First, we show that the three conditions are sufficient for t to be an Armstrong table for Σ and $nfs(T_s)$. Suppose that t is such that the three conditions are satisfied. It follows immediately from the last condition that t satisfies $nfs(T'_s)$ if and only if $T'_s \subseteq T_s$. Let $u(X) \in \Sigma$. If there were two rows $r_1, r_2 \in t$ such that $r_1 \neq r_2$ and $X \subseteq ag^s(r_1,r_2) = Y$, then $Y \in ag^s(t)$. This, however, would violate the second condition. Hence, t satisfies $u(X)$. Since $u(X) \in \Sigma$ was an arbitrary choice we conclude that t satisfies Σ. Let $u(Y) \notin \Sigma^*$. Then there is some $a(X) \in \Sigma^{-1}$ such that $Y \subseteq X$ holds. From the first condition we conclude that $Y \subseteq ag^s(r_1,r_2)$ for some $r_1, r_2 \in t$ with $r_1 \neq r_2$. Consequently, t violates every uniqueness constraint not implied by Σ.

Showing that the three conditions hold necessarily whenever t is an Armstrong table for Σ and $nfs(T_s)$ still remains. Suppose that t is an Armstrong table for Σ and $nfs(T_s)$. The last condition follows immediately from the fact that t satisfies $nfs(T'_s)$ if and only if $T'_s \subseteq T_s$. Since t satisfies Σ there cannot be any $Y \in ag^s(t)$ and $u(X) \in \Sigma$ such that $X \subseteq Y$ holds. We conclude that the second condition is satisfied. It remains to show that the first condition is satisfied, too. Let $a(X) \in \Sigma^{-1}$. We need to show that $X \in ag^s(t)$. From $a(X) \in \Sigma^{-1}$ it follows that $u(X) \notin \Sigma^*$. As t violates $u(X)$ it follows that there are $r_1, r_2 \in t$ such that $r_1 \neq r_2$ and $X \subseteq Y = ag^s(r_1,r_2)$. Also, from $a(X) \in \Sigma^{-1}$ it follows that for all $H \in T - X$ we have $u(XH) \in \Sigma^*$, and thus that t satisfies $u(XH)$. Suppose that there is some $H \in Y - X$. Then t satisfies $u(XH)$ and, therefore, $r_1(H) = \texttt{ni} = r_2(H)$ or $r_1(H) \neq r_2(H)$. This, however, is a contradiction as $H \in Y = ag^s(r_1,r_2)$. Consequently, $X = Y \in ag^s(t)$. □

EXAMPLE 5. Let SCHEDULE = *CTLR* with SCHEDULE_s = *CT*. Let Σ consist of the two UCs $u(CT)$ and $u(LTR)$, and let t denote the data sample in Table 2. Recall from the previous examples that $\Sigma^{-1} = \{a(CLR), a(LT), a(TR)\}$, and $ag^s(t) = \{CLR, LT, TR, L, R, T\}$. Since t satisfies the three conditions of Theorem 2, it follows that t is an Armstrong table for Σ and SCHEDULE_s. □

C. Computation of Armstrong Tables

We will now use the characterization of Theorem 2 to compute Armstrong tables for an arbitrarily given set Σ of UCs and an arbitrarily given NFS $nfs(T_s)$ over an arbitrarily given table schema T. A great part of the computation is devoted to determining the set Σ^{-1}. For this purpose, we borrow concepts from hyper-graphs. Indeed, to compute Σ^{-1} we generate the simple hyper-graph $\mathcal{H} = (V, E)$ with the vertex set $V = T$ and the set $E = \{X \mid u(X) \in \Sigma\}$ of hyper-edges. In fact, based on Theorem 1 we assume without loss of generality that Σ consists of minimal UCs only. If not, then we remove all those UCs from Σ that are not minimal. From this we obtain $\Sigma^{-1} = \{a(T - X) \mid X \in Tr(\mathcal{H})\}$ where $Tr(\mathcal{H})$ denotes the minimal transversals of the hyper-graph $\mathcal{H}$, i.e., the set of minimal sets $X \subseteq T$ that have a non-empty intersection with every hyper-edge of $\mathcal{H}$ [34].

LEMMA 1. *Let Σ be a set of UCs over table schema T. Then, $\Sigma^{-1} = \{a(T - X) \mid X \in Tr(\mathcal{H})\}$.*

Proof. Recall that $Tr(\mathcal{H}) = \{X \subseteq T \mid \forall u(Z) \in \Sigma(Z \cap X \neq \emptyset) \wedge (\exists Y \subseteq X(\forall u(Z) \in \Sigma(Z \cap Y \neq \emptyset)) \Rightarrow Y = X)\}$.

We show first that if $X \in Tr(\mathcal{H})$, then $a(T - X) \in \Sigma^{-1}$. First, it follows that $\Sigma \not\models_{T_s} u(T - X)$ since otherwise there would be some $u(Z) \in \Sigma$ such that $Z \subseteq T - X$. This, however, would mean that $Z \cap X = \emptyset$, which contradicts the hypothesis that $X \in Tr(\mathcal{H})$. It remains to show that for all $H \in X$, $\Sigma \models_{T_s} u((T - X)H)$. Assume the opposite, i.e., there is some $H \in X$ such that $\Sigma \not\models_{T_s} u((T - X)H)$. Then, there cannot be any $u(Z) \in \Sigma$ such that $Z \subseteq (T - X)H = T - (X - H)$. Hence, for all $u(Z) \in \Sigma$ we have $Z \cap (X - H) \neq \emptyset$. This, however, contradicts the minimality of $X \in Tr(\mathcal{H})$. We have shown that $a(T - X) \in \Sigma^{-1}$.

Now, we show that if $a(X) \in \Sigma^{-1}$, then $T - X \in Tr(\mathcal{H})$. From $a(X) \in \Sigma^{-1}$ we conclude that $\Sigma \not\models_{T_s} u(X)$. Hence, for all $u(Z) \in \Sigma((T - X) \cap Z \neq \emptyset)$. From $a(X) \in \Sigma^{-1}$ we know that for all $H \in T - X$, $\Sigma \models_{T_s} u(XH)$. Hence, for all $H \in T - X$ there is some $u(Z) \in \Sigma$ such that $Z \subseteq XH$. Thus, for all $H \in T - X$ there is some $u(Z) \in \Sigma$ such that $(T - XH) \cap Z = \emptyset$. That is, $T - X \in Tr(\mathcal{H})$. □

Now, we have a complete strategy for computing Armstrong tables, which we summarize in Algorithm 1. For

each column header H, let $c_{H,0}$, $c_{H,1}$, ... denote mutually distinct domain values from $dom(H)$. Lines 2-4 deal with the special case where Σ contains the empty key $u(\emptyset)$, saying that a table can have at most one row. Here, we just need to return a table consisting of one row with null marker occurrences in columns which are null-able. Otherwise, we start off with a row r_0 with total values in every column (line 6). In lines 8 and 9 we use Lemma 1 to compute the set of anti-keys with respect to Σ and $nfs(T_s)$. In lines 11-15 we introduce for each anti-key $a(Y)$ a new row r_i, which has a strong agree set Y with r_0. In lines 16-20 we may need to introduce an additional row that has null value occurrences in null-able columns that do not feature null value occurrences in previously generated rows.

Algorithm 1 Armstrong table computation

1: **procedure** ARMSTRONG($T, \Sigma, nfs(T_s)$)
2: **if** $u(\emptyset) \in \Sigma$ **then return** ▷ special case
3: $t := \{r_0\}$ such that
4: $\forall H \in T,\ r_0(H) := \begin{cases} c_{H,0} & \text{, if } H \in T_s \\ \texttt{ni} & \text{, else} \end{cases}$
5: **else**
6: $t := \{r_0\}$ where $\forall H \in T,\ r_0(H) = \{c_{H,0}\}$
7: **end if**
8: $\mathcal{H} := (T, \{X \mid u(X) \in \Sigma\})$ ▷ hypergraph
9: $\Sigma^{-1} := \{a(T - X) \mid X \in Tr(\mathcal{H})\}$ ▷ Lemma 1
10: $i := 1$
11: **for all** $a(Y) \in \Sigma^{-1}$ **do** ▷ add agree sets
12: $t := t \cup \{r_i\}$ where $\forall H \in T$,
13: $r_i(H) := \begin{cases} c_{H,0} & \text{, if } H \in Y \\ c_{H,i} & \text{, if } H \in T_s - Y \\ \texttt{ni} & \text{, else} \end{cases}$
14: $i := i + 1$
15: **end for**
16: $total(t) := \{H \in T \mid \forall r \in t(r(H) \neq \texttt{ni})\}$
17: **if** $total(t) - T_s \neq \emptyset$ **then** ▷ add null markers
18: **return** $t := t \cup \{r_i\}$ where $\forall H \in T$,
19: $r_i(H) := \begin{cases} \texttt{ni} & \text{, if } H \in total(t) - T_s \\ c_{H,i} \end{cases}$
20: **else**
21: **return** t
22: **end if**
23: **end procedure**

The correctness of Algorithm 1 follows from Lemma 1 and Theorem 2.

THEOREM 3. *On input* $(T, \Sigma, nfs(T_s))$*, Algorithm 1 computes a table* t *over* T *that is Armstrong for* Σ *and* $nfs(T_s)$.

Proof. It suffices to verify that the output of Algorithm 1 satisfies the conditions in Theorem 2. Indeed, Lemma 1 reveals that Algorithm 1 computes the anti-keys correctly. The main construction of Algorithm 1 in lines 11-15 guarantees that the output satisfies the first condition. The construction also ensures that every strong agree set from the output of Algorithm 1 is contained in some anti-key. This means that no UC from Σ can be contained in some strong agree set. This shows that the output satisfies the second condition. Finally, lines 17-19 ensure that the output satisfies the third condition. □

EXAMPLE 6. Let SCHEDULE = CTLR with SCHEDULE_s = *CT*. Let Σ consist of the two UCs $u(CT)$ and $u(LTR)$. On input (SCHEDULE, Σ, SCHEDULE_s), Algorithm 1 would compute the following Armstrong table:

C_ID	Time	L_Name	Room
$c_{H,0}$	$c_{T,0}$	$c_{L,0}$	$c_{R,0}$
$c_{H,0}$	$c_{T,1}$	$c_{L,0}$	$c_{R,0}$
$c_{H,1}$	$c_{T,0}$	$c_{L,0}$	ni
$c_{H,2}$	$c_{T,0}$	ni	$c_{R,0}$

A suitable value substitution yields the data sample in Table 2. □

D. Complexity Considerations

In this subsection, we investigate properties regarding the space and time complexity for computing Armstrong tables. We will demonstrate that the user-friendly representation of a set of SQL keys in the form of an Armstrong table comes, in the worst case, at a price. In fact, the number of rows in a minimum-sized Armstrong table can be exponential in the total number of column headers that occur in Σ. Because of this result we cannot, in the worst case, design an algorithm for generating Armstrong tables in polynomial time.

1) Worst-case time-complexity

The following result follows straight from Theorem 2 and the correctness of Algorithm 1.

PROPOSITION 1. *Let* Σ *be a set of UCs and* $nfs(T_s)$ *be some NFS over table schema* T*. Let* t *be an Armstrong table for* Σ *and* $nfs(T_s)$*. Then* $|\Sigma^{-1}| \leq |ag^s(t)| \leq \binom{|t|}{2}$.

Proof. The first condition of Theorem 2 states that for all $a(X) \in \Sigma^{-1}$ we have $X \in ag^s(t)$. If we add the second condition of Theorem 2, then we derive that for all $Y \in ag^s(t)$ there is at most one $X \in \Sigma^{-1}$ such that $X \subseteq Y$. This shows that

$$|\Sigma^{-1}| \leq |ag^s(t)|.$$

Finally, $|ag^s(t)| \leq \binom{|t|}{2}$ since every distinct pair of distinct tuples in t has precisely one agree set. □

Let the size of an Armstrong table be defined as the number of rows that it contains. It is a practical question to ask how many rows a minimum-sized Armstrong table requires. An Armstrong table t for Σ and $nfs(T_s)$ is said to be *minimum-sized* if there is no Armstrong table t' for Σ and $nfs(T_s)$ such that $|t'| < |t|$. That is, for a minimum-sized Armstrong table for Σ and $nfs(T_s)$ there is no Armstrong

table for Σ and $nfs(T_s)$ with a smaller number of rows.

We recall what we mean by *precisely exponential* [24]. Firstly, it means that there is an algorithm for computing an Armstrong table, given a set Σ of UCs and an NFS $nfs(T_s)$, where the running time of the algorithm is exponential in the size of the keys. Secondly, it means that there is a set Σ of UCs and an NFS $nfs(T_s)$ in which the number of rows in each minimum-sized Armstrong table for Σ and $nfs(T_s)$ is exponential.

PROPOSITION 2. *The complexity of finding an Armstrong table, given a set of UCs and an NFS, is precisely exponential in the size of the UCs*.

Proof. The time complexity of Algorithm 1 is proportional to the time-complexity of computing the set Σ^{-1} of anti-keys with respect to Σ and $nfs(T_s)$. The computation of the anti-keys via the hypergraph transversals in lines 8 and 9 requires time exponential in the size of the UCs given [35]. Therefore, Algorithm 1 runs in time exponential in the size of the UCs given.

It remains to show that there is a set Σ of UCs for which the number of rows in each Armstrong table for Σ is exponential in the size of the UCs given. According to Theorem 1 it suffices to find a set Σ of UCs such that Σ^{-1} is exponential in the size of the UCs. Such a set is given by

$$\Sigma = \{u(H_{2i-1}, H_{2i}) \mid i = 1, \ldots, n\}.$$

The set Σ^{-1} consists of those anti-keys that contain precisely one column header for each element of Σ. Therefore, Σ^{-1} contains precisely 2^n elements with $2n$ being the size of Σ. □

2) Minimum-sized Armstrong tables

Despite the general worst-case exponential complexity in the size of the keys, Algorithm 1 is a fairly simple algorithm that is, as we now demonstrate, quite conservative in its use of time.

PROPOSITION 3. *Let Σ be a set of UCs, $nfs(T_s)$ an NFS over table schema T. Let t be a minimum-sized Armstrong table for Σ and $nfs(T_s)$. Then* $\frac{\sqrt{1+8\cdot|\Sigma^{-1}|}}{2} \leq |t| \leq |\Sigma^{-1}| + 2$.

Proof. The upper bound follows immediately from Theorem 3. The lower bound follows from Theorem 1. Indeed, it follows that $|\Sigma^{-1}| \leq \binom{|t|}{2}$. That is, $|\Sigma^{-1}| \leq \frac{|t|\cdot(|t|-1)}{2}$. Consequently, $\frac{\sqrt{1+8\cdot|\Sigma^{-1}|}}{2} \leq |t|$. □

Note the upper bound in Proposition 3. In general, the focus on UCs can yield Armstrong tables with a substantially smaller number of rows than Armstrong tables for more expressive classes such as functional dependencies. The reason is that we do not need to violate any functional dependencies that are not implied by the given set. In practice, this is desirable for the validation of schemata known to be in Boyce-Codd normal form, for example. Such schemata are often the result of entity-relationship modeling. Applying the algorithm from [31], designed for UCs and functional dependencies, to our running example would yield an Armstrong table with 12 rows. Instead, Algorithm 1, designed for UCs only, produces an Armstrong table with just 4 rows. In general, we can conclude that Algorithm 1 always computes an Armstrong table of reasonably small size.

COROLLARY 1. *On input* $(T, \Sigma, nfs(T_s))$, *Algorithm 1 computes an Armstrong table for* Σ *and* $nfs(T_s)$ *whose size is at most quadratic in the size of a minimum-sized Armstrong table for* Σ *and* $nfs(T_s)$. □

3) Size of representations

We show that, in general, there is no superior way of representing the information inherent in a set of UCs and a null-free subschema.

THEOREM 4. *There is some set* Σ *of UCs and an NFS* $nfs(T_s)$ *such that* Σ *has size* $\mathcal{O}(n)$, *and the size of a minimum-sized Armstrong table for* Σ *and* $nfs(T_s)$ *is* $\mathcal{O}(2^{n/2})$. *There is some table schema T, some NFS* $nfs(T_s)$ *and some set* Σ *of UCs over T such that there is an Armstrong table for* Σ *and* $nfs(T_s)$ *where the number of rows is in* $\mathcal{O}(n)$, *and the size of the minimal UCs implied by* Σ *in the presence of* $nfs(T_s)$ *is in* $\mathcal{O}(2^n)$.

Proof. For the first claim let $T = H_1, \ldots, H_{2n}$, $T_s = T$ and $\Sigma = \{u(H_{2i-1}H_{2i}) \mid i = 1, \ldots, n\}$. Then $\Sigma^{-1} = \{a(X_1 \cdots X_n) \mid \forall i = 1, \ldots, n(X_i \in \{H_{2i-1}, H_{2i}\})\}$.

For the second claim let $T = H_1 \cdots H_{2n}$, $T_s = T$ and $\Sigma = \{u(X_1 \cdots X_n) \mid \forall i = 1, \ldots, n(X_i \in \{H_{2i-1}, H_{2i}\})\}$. Then, the set of minimal UCs implied by Σ is Σ itself. Since $\Sigma^{-1} = \{a(T - (H_{2i-1}H_{2i})) \mid i = 1, \ldots, n\}$ there is an Armstrong table for Σ and $nfs(T_s)$ where the number of rows is in $\mathcal{O}(n)$. □

We can perceive that the representation in form of an Armstrong table can offer tremendous space savings over the representation as a UC set, and vice versa. It seems intuitive to use the representation as Armstrong tables for the discovery of semantically meaningful constraints, and the representation as constraint sets for the discovery of semantically meaningless constraints. This intuition has been confirmed empirically for the class of functional dependencies over relations [18].

V. DATA-DRIVEN SQL KEY DISCOVERY

In this section we will establish algorithms for the automated discovery of uniqueness constraints from given SQL tables. Such algorithms have direct applications in schema design, query optimization, and the semantic sampling of databases [22, 26, 31, 36, 37]. In requirements engineering, for example, these algorithms can be utilized to discover semantically meaningful uniqueness constraints

from sample data, which domain experts provide.

A. Mining by Pairwise Comparison of Rows

Our first algorithm gradually inspects all pairs of rows for the given table, and adjusts the set of minimal UCs accordingly. More specifically, for every given pair of different rows r_1, r_2 (line 3), and for every UC $u(X)$ satisfied by the rows inspected so far (line 4), we replace $u(X)$ by $u(XH)$ whenever r_1, r_2 violate $u(X)$ and H is a column in $T - X$ such that r_1, r_2 satisfy $u(XH)$ (lines 5-10). Lines 12-14 remove non-minimal UCs. Note that the output of Algorithm 2 is $uc(t) = \emptyset$ whenever t contains any duplicate rows.

Algorithm 2 Mining of uniqueness constraints by pairwise row comparisons

```
1:  procedure MINE-BY-ROW((T, t))
2:      uc(t) := {u(∅)}
3:      for all r1, r2 ∈ t such that r1 ≠ r2 do
4:          for all u(X) ∈ uc(t) do
5:              if X ⊆ ag^s(r1, r2) then
6:                  uc(t) := uc(t) − {u(X)}
7:                  for all H ∈ T − ag^s(r1, r2) do
8:                      uc(t) := uc(t) ∪ {u(XH)}
9:                  end for
10:             end if
11:         end for
12:         while u(X), u(Y) ∈ uc(t) with X ⊆ Y do
13:             uc(t) := uc(t) − {u(Y)}
14:         end while
15:     end for
16: end procedure
```

THEOREM 5. *On input* (T, t)*, Algorithm 2 computes the set of minimal UCs satisfied by table* t *over* T *in time* $\mathcal{O}(|T|^2 \times |t|^2 \times m_t^2)$ *where* m_t *denotes the maximum number of minimal UCs satisfied by any subset* $s \subseteq t$.

Proof. Algorithm 2 works correctly. A formal proof can be established using induction over the number of rows in the input table t. When t contains no row or just one row, then Algorithm 2 returns $uc(t) = \{u(\emptyset)\}$ - which is the unique minimal UC satisfied by t. Suppose we know that Algorithm 2 correctly computes the set of minimal UCs satisfied by table t with n rows, and that $t' := t \cup \{r'\}$. Then, the new row r' is compared with every other row r in t. Whenever a UC $u(X)$ is violated, it is replaced by the set of uniqueness constraints $u(XH)$ where r' and r do not strongly agree on H. This ensures that table t' satisfies all the uniqueness constraints in $uc(t')$. Finally, lines 12-14 ensure that $uc(t')$ does not contain any uniqueness constraints that are not minimal.

For the complexity bound note that there are $|t|^2$ pairs of rows to inspect. For each pair, $uc(t)$ has at most m_t elements, and for each element there are at most $m_t \cdot |T|^2$ operations to perform. □

EXAMPLE 7. Let t, t' be the data samples over SCHEDULE from Tables 2 and 3, respectively. The following table shows the evolution of the minimal UCs by gradually adding a new pair of rows until the entire table has been explored.

rows	$uc(t)$	$uc(t')$
1,2	*T*	*T*
1,3	*TR, TC*	*TR, TL, TC*
1,4	*TRL,TC*	*TR,TL,TC*
2,3	*TRL,TC*	*TR,TL,TC*
2,4	*TRL,TC*	*TR,TL,TC*
3,4	*TRL,TC*	*TR,TL,TC*

The UCs are those explained in the introductory section of this article. □

B. Mining by Exploration of Hyper-Graph Transversals

Again, our next algorithm uses the concept of hypergraph transversals. Indeed Algorithm 3 computes the complements of the strong agree sets for the given input table (lines 2 and 3). These complements are called weak disagree sets. Lines 4 and 5 compute the necessary weak disagree sets, that is, those weak agree sets that are minimal. For the hypergraph where the node set is T and the edge set consists of all necessary weak disagree sets, Algorithm 3 uses any (preferably the most efficient) algorithm to compute the set of all minimal transversals of the hypergraph. Lemma 2 shows that this set contains all minimal UCs satisfied by the input table t. Algorithm 3 is compact and benefits from any progress on the popular problem of computing minimal hypergraph transversals [34].

The following lemma explains the soundness of Algorithm 3.

LEMMA 2. *Let* t *be a table over table schema* T*. Then, for all* $X \subseteq T$, $X \in Tr(T, nec\text{-}disag^w(t))$ *if and only if* $\models_t u(X)$ *and for all* $H \in X$, $\not\models_t u(X - H)$.

Proof. We begin by showing that the two conditions are necessary for every $X \in Tr(T, nec\text{-}disag^w(t))$.

Algorithm 3 Mining of uniqueness constraints by exploring hypergraph transversals

```
1: procedure MINE-BY-TRANSVERSALS((T, t))
2:     disag^w(t) := {T − ag^s(r1, r2) |
3:                                  r1, r2 ∈ t ∧ r1 ≠ r2}
4:     nec-disag^w(t) := {X ∈ disag^w(t) |
5:                          ¬∃Y ∈ disag^w(t)(Y ⊂ X)}
6:     H := (T, nec-disag^w(t))
7:     uc(t) := {u(X) | X ∈ Tr(H)}
8: end procedure
```

First, if t violated $u(X)$, then there were two rows $r_1, r_2 \in t$ such that $r_1 \neq r_2$ and $X \subseteq ag^s(r_1, r_2)$. Hence, $X \cap disag^w(r_1, r_2) = \emptyset$ and there would be some $Y \in nec\text{-}disag^w(t)$ such that $Y \subseteq disag^w(r_1, r_2)$. Thus, $X \cap Y = \emptyset$ which contradicts the hypothesis that $X \in Tr(T, nec\text{-}disag^w(t))$. We conclude that t *satisfies* $u(X)$. Suppose there was some $H \in X$ such that $|=_t u(X - H)$. Then, it would follow that for all $r_1, r_2 \in t$ such that $r_1 \neq r_2$ it held that $(X - H) \cap disag^w(r_1, r_2) \neq \emptyset$. This, however, would violate the minimality of $X \in Tr(T, nec\text{-}disag^w(t))$. Therefore, it holds that for all $H \in X$, $|\neq_t u(X - H)$.

Now, we demonstrate that the two conditions are sufficient for X to be in $Tr(T, nec\text{-}disag^w(t))$. From $|=_t u(X)$ follows that for all $r_1, r_2 \in t$ where $r_1 \neq r_2$ we have $X \cap disag^w(r_1, r_2) \neq \emptyset$. From $|\neq_t u(X - H)$ for all $H \in X$ it follows that for all $H \in X$ there are some $r_1, r_2 \in t$ such that $r_1 \neq r_2$ and $X - H \subseteq ag^s(r_1, r_2)$. The last condition means that $(X - H) \cap disag^w(r_1, r_2) = \emptyset$ holds. Consequently, for all $H \in X$ there is some $Z \in nec\text{-}disag^w(t)$ such that $Z \subseteq disag^w(r_1, r_2)$, and thus $(X - H) \cap Z = \emptyset$. We conclude that $X \in Tr(T, nec\text{-}disag^w(t))$. □

THEOREM 6. *On input* (T, t), *Algorithm 3 computes the set of minimal UCs satisfied by table* t *over* T *in time* $\mathcal{O}(m + n)$ *where* $m := |T|^2 \times |t|^2 \times |nec\text{-}disag^w(t)|$ *and* $n := \left(\prod_{X \in nec\text{-}disag^w(t)} |X| \right)^2$.

Proof. The correctness of Algorithm 3 follows directly from the description of the algorithm and Lemma 2.

The collection $nec\text{-}disag^w(t)$ can be computed in time $\mathcal{O}(m)$: there are $|t|^2$ pairs of rows to consider, and for each new set in $disag^w(t)$ one can check with at most $|T|^2 \times |nec\text{-}disag^w(t)|$ operations whether the set is also in $nec\text{-}disag^w(t)$. The computation of the set $Tr(T, nec\text{-}disag^w(t))$ of minimal transversals can be done in time $\mathcal{O}(n)$ [34]. Note that the hypergraph $(T, nec\text{-}disag^w(t))$ is simple, that is, there are no different sets X, Y in $nec\text{-}disag^w(t)$ where $X \subseteq Y$ holds. □

EXAMPLE 8. Let t, t' be the data samples over SCHEDULE from Tables 2 and 3, respectively.

The next table shows the steps for applying Algorithm 3 to (SCHEDULE, t) and (SCHEDULE, t'), respectively.

	t	t'
$ag^s(\cdot)$	*CLR, TL, TR, L, R, T*	*CLR, T,* $\emptyset$
$nec\text{-}disag^w(\cdot)$	*T, CR, CL*	*T, CLR*
$u(\cdot)$	*CT, TLR*	*CT, LT, RT*

The UCs are those explained in the introductory section of this article. □

VI. INFORMATIVE ARMSTRONG TABLES

Here, we combine the schema- and data-driven approach to the discovery of SQL keys, as given in Sections IV and V, respectively. The disadvantage of the schema-driven approach is that the Armstrong tables, when generated by our algorithm, contain only artificial values. It is therefore doubtful that such tables illustrate the current perceptions of the data engineers to the domain experts who inspect the table. Of course, the data engineers may substitute real domain values for the artificial values before they present the table to the domain experts, or they generate the Armstrong tables on the basis of some real domain values. However, the table may still not be a good representative of the real world application domain, since some of the combinations of the values may never occur in practice. We will now outline a solution to this problem, which overcomes this limitation. Here, the assumption that is necessary is that some real world data set t is available, for example, in the form of legacy data. Under this assumption, we can apply the data-driven algorithms from Section V to mine the set $uc(t)$ of minimal uniqueness constraints (and `NOT NULL` constraints) satisfied by t. Subsequently, we apply the schema-driven algorithm from Section IV to compute an Armstrong table t' for $uc(t)$ and the null-free subschema satisfied by t. The Armstrong table t' can be populated with suitable rows from t, that is, t' is a subset of the table t. Thus, the number of rows in t' will be much smaller than the number of rows in t. Therefore, t' constitutes an invaluable semantic sample of the data set t, semantic in the sense that it satisfies the same set of uniqueness and `NOT NULL` constraints as t.

Table 4. Real-world table t_{real}

C_ID	Time	L_Name	Room
COMPSCI 210	Mon, 2 pm	Manoharan	OGH
COMPSCI 210	Tue, 2 pm	Ye	OGH
COMPSCI 210	Wed, 1 pm	ni	OGH
COMPSCI 215	Mon, 3 pm	Russello	Eng1439
COMPSCI 215	Tue, 3 pm	Sheehan	Eng1439
COMPSCI 215	Wed, 4 pm	Ye	MLT1
COMPSCI 220	Tue, 10 am	Speidel	PLT1
COMPSCI 220	Wed, 11 am	ni	PLT1
COMPSCI 220	Fri, 1 pm	Welch	ni
COMPSCI 225	Mon, 11 am	ni	HSB2
COMPSCI 225	Tue, 12 pm	Nies	OGH
COMPSCI 225	Fri, 10 am	ni	ni
COMPSCI 230	Tue, 11 am	ni	OGH
COMPSCI 230	Thu, 11 am	Thomborson	OGH
COMPSCI 230	Fri, 2 pm	Chang	ni

Table 5. Informative Armstrong table t_{inf} for t_{real}

C_ID	Time	L_Name	Room
COMPSCI 210	Tue, 2 pm	Ye	OGH
COMPSCI 210	Wed, 1 pm	ni	OGH
COMPSCI 215	Wed, 4 pm	Ye	MLT1
COMPSCI 230	Fri, 2 pm	Chang	ni

DEFINITION 4. *Let t be a table over table schema T, and let $\mathcal{C}$ denote a class of constraints. A $\mathcal{C}$-informative Armstrong table for t is a table t' over T such that i) $t' \subseteq t$ and ii) for all constraints φ of $\mathcal{C}$ over T, t' satisfies φ if and only if t satisfies φ.* □

In the context of this paper, we are interested in informative Armstrong tables with respect to the class $\mathcal{C}$ of uniqueness and NOT NULL constraints. For an illustration of the concept we will now look at a small example.

The data sample t_{real} in Table 4 shows some aspects of the Semester 2 timetable for 200 level courses in the Department of Computer Science, the University of Auckland, one month before the beginning of Semester 2 in 2012. Suppose we would like to compute an informative Armstrong table for t_{real} with respect to the class of uniqueness and NOT NULL constraints. We may apply Algorithm 2 or Algorithm 3 to input (SCHEDULE, t), and compute the set $uc(t)$ as

$$\{u(\text{Time}), u(\text{C\_ID,L\_Name}), u(\text{Room,L\_Name})\}.$$

It is not difficult to compute the maximal null-free subschema $nfs(T_s)$ satisfied by t_{real}. We would simply start with T and gradually inspect row by row in t_{real}. For each row r we remove any column $H \in T_s$ from T_s whenever $r(H)$ = ni. For the data sample t_{real} from Table 4 we obtain T_s = {C_ID, Time}. Using the algorithms from Section IV we compute the set $uc(t)^{-1}$ of anti-keys as

$$\{a(\text{L\_Name}), a(\text{C\_ID,Room})\}.$$

Now, finding rows in t_{real} whose strong agree sets are these anti-keys, and which violate any null-free subschema violated by t_{real} still remains. One such table t_{inf} is shown in Table 5. The first two rows have a strong agree set {C_ID,Room}, the first and third row have strong agree {L_Name}, and the fourth row ensures that t' is total exactly where t_{real} is. The example illustrates the potential savings in terms of size that informative Armstrong tables have over the original data samples. For original data samples that are large, the informative Armstrong tables will be considerably smaller. For example, in the case of relational databases, De Marchi and Petit [22] have reported that their informative Armstrong database contained only 0.6% of the tuples in the original database. One can imagine that the inspection for semantic samples of smaller size is much more effective than the inspection of the original data set.

VII. EMPIRICAL MEASURES OF USEFULNESS

In this section we describe how the usefulness of applying Armstrong tables for the discovery of semantically meaningful SQL keys can be measured. For this purpose, we will first introduce different measures of usefulness, and then describe a detailed example illustrating how the marking and feedback for non-multiple choice questions in database courses can be automated.

A. Soundness, Completeness, and Proximity

Measuring the usefulness of applying Armstrong tables for the discovery of semantically meaningful SQL keys appears to be non-trivial. However, one may conduct a two-phase experiment where database design teams are given an application domain and are asked to specify the set of UCs they perceive as semantically meaningful. In the first phase, they are not given any help, except natural language descriptions by domain experts. In the second phase, our algorithm can be used to produce Armstrong tables for the set of UCs the teams perceive currently as semantically meaningful. The Armstrong tables may be inspected together with the domain experts, and when new UCs are identified a corresponding Armstrong table may be repeatedly produced. For an experiment or assignment, one may specify a target set Σ_t and possibly a target NFS $nfs(T_s^t)$. One may then measure the quality of the output sets (Σ_1, T_s^1) of the first phase against the target sets (Σ_t, T_s^t), and the output sets (Σ_2, T_s^2) of the second phase against the target sets (Σ_t, T_s^t). If there is an increase in quality, then Armstrong tables, indeed, appear to be useful. The question remains how to measure the quality of the output sets against the target sets. For this purpose we propose three measures. Let $min(\Sigma)$ denote the UCs in Σ that are minimal. *Soundness* measures, which of the (minimal) UCs and headers currently perceived as semantically meaningful, are actually semantically meaningful:

$$sound_{\Sigma_t, T_s^t}(\Sigma, T_s) = \frac{|min(\Sigma) \cap \Sigma_t^*| + |T_s \cap T_s^t|}{|min(\Sigma)| + |T_s|}.$$

If $\Sigma = \emptyset$ and $T_s = \emptyset$, we define $sound_{\Sigma_t, T_s^t}(\Sigma, T_s) = 1$. *Completeness* measures, which of the actually semanti-

Table 6. Armstrong table as feedback

C_ID	Time	L_Name	Room
11301	Mon, 10 am	Ullman	Red
55505	Mon, 2 pm	Ullman	Red
55505	Tue, 9 am	Fagin	Red
55505	Tue, 2 pm	Fagin	Blue
77707	Tue, 2 pm	Gottlob	Blue
ni	Tue, 2 pm	Gottlob	Green

cally meaningful (minimal) UCs and headers in NFSs, are currently perceived as semantically meaningful:

$$complete_{\Sigma_t, T_s^t}(\Sigma, T_S) = \frac{\left|\Sigma^* \cap min(\Sigma_t)\right| + \left|T_s \cap T_s^t\right|}{\left|min(\Sigma_t)\right| + \left|T_s^t\right|}.$$

If $\Sigma_t = \emptyset$ and $T_s^t = \emptyset$, we define $complete_{\Sigma_t, T_s^t}(\Sigma, T_S) = 1$. Finally, *proximity* $prox((\Sigma, T_s), (\Sigma_t, T_s^t))$ combines soundness and completeness and is defined as:

$$\frac{\left|(min(\Sigma) \cap \Sigma_t^*) \cup (\Sigma^* \cap min(\Sigma_t))\right| + \left|T_s \cap T_s^t\right|}{\left|min(\Sigma) \cup min(\Sigma_t)\right| + \left|T_s \cup T_s^t\right|}.$$

If $\Sigma = \emptyset = \Sigma_t$ and $T_s = \emptyset = T_s^t$, we define $prox((\Sigma,T_s), (\Sigma_t,T_s^t)) = 1$.

In database courses one may use Armstrong tables as automated feedback to solutions. Our measures can be applied to automatically mark non-multiple choice questions. This can reduce errors and save time in assessing course work.

B. A Detailed Example

Again, we will use our running example where SCHEDULE consists of the four attributes C(_ID), L(_Name), R(oom), and T(ime). Suppose we describe, in natural language, the application domain to the students of a database course. Then, we ask them to identify the uniqueness and NOT NULL constraints that they perceive to be semantically meaningful for this application domain. The students may ask questions in natural language to clarify their perceptions about the application domain. The lecturers of the course act as domain experts who will provide answers to questions in natural language that are consistent with the target set $\Sigma_t = \{u(CT), u(LT), u(RT)\}$, $\text{SCHEDULE}_s^t = CT$ over the table schema SCHEDULE.

Suppose one group of students returns as an answer to the question the set $\Sigma = \{u(CT), u(LRT), u(CLR)\}$ and $T_s = LRT$. One can then automatically compute $sound_{\Sigma_t, T_s^t}(\Sigma, T_S) = 1/2$, $complete_{\Sigma_t, T_s^t}(\Sigma, T_S) = 2/5$, and $prox((\Sigma, T_s), (\Sigma_t, T_s^t)) = 1/3$. Furthermore, one may also return an Armstrong table for Σ and $nfs(T_s)$, for example, the one in Table 6.

The students inspect the table and may make several observations. For example, Fagin teaches 55505 and Gottlob teaches 77707 at the same time in the same room. The students decide to ask the domain expert whether different courses can be taught in the same room at the same time. The domain experts indicate that this is impossible, and the students decide to replace the UC $u(LRT)$ by the UC $u(RT)$ in response. In addition, the students notice that Gottlob teaches in different rooms at the same time. As this is impossible, they decide to specify the UC $u(LT)$. They now submit their revised constraint set $\Sigma' = \{u(CT), u(RT), u(LT), u(CLR)\}$ and $T_s = LRT$. Again, one can then automatically compute $sound_{\Sigma_t, T_s^t}(\Sigma', T_s) = 4/7$, $complete_{\Sigma_t, T_s^t}(\Sigma', T_s) = 4/5$, and $prox((\Sigma', T_s), (\Sigma_t, T_s^t)) = 1/2$. Hence, inspecting the Armstrong table results in an improvement of 1/14 in soundness, 2/5 in completeness, and 1/6 in proximity.

VIII. CONCLUSION AND FUTURE WORK

We investigated the data- and schema-driven discovery of SQL keys. We established insights into structural and computational properties of Armstrong tables. These can increase the discovery of semantically meaningful SQL keys in practice, leading to better schema designs and improved data processing. In addition, we also established algorithms to automatically discover SQL keys in given SQL tables. These have applications in requirement acquisition, schema design and query optimization. We combined the data- and schema-driven approaches to compute informative Armstrong tables, which are effective semantic samples of given data sets. Moreover, we defined formal measures to assess the difference between sets of SQL keys. These can be applied to validate the usefulness of Armstrong tables and to database education. Our findings also apply to Codd's null marker interpretation *value exists but unknown*, using Levene and Loizou's possible world semantics [38]. In the future we plan to implement our results in a design aid, to test our measures in applications, and to address the class of foreign keys.

ACKNOWLEDGMENTS

This research is supported by the Marsden Fund Council from Government funding, administered by the Royal Society of New Zealand.

REFERENCES

1. S. Hartmann, U. Leck, and S. Link, 'On Codd families of keys over incomplete relations," *The Computer Journal*, vol. 54, no. 7, pp. 1166-1180, 2011.
2. B. Thalheim, "On semantic issues connected with keys in relational databases permitting null values," *Journal of Information Processing and Cybernetics*, vol. 25, no. 1-2, pp. 11-20, 1989.
3. B. Thalheim, "The number of keys in relational and nested relational databases," *Discrete Applied Mathematics*, vol. 40, no. 2, pp. 265-282, 1992.
4. G. E. Weddell, "Reasoning about functional dependencies generalized for semantic data models," *ACM Transactions on Database Systems*, vol. 17, no. 1, pp. 32-64, 1992.
5. V. L. Khizder and G. E. Weddell, "Reasoning about uniqueness constraints in object relational databases," *IEEE Transactions on Knowledge and Data Engineering*, vol. 15, no. 5, pp. 1295-1306, 2003.
6. D. Toman and G. E. Weddell, "On keys and functional dependencies as first-class citizens in description logics," *Journal of Automated Reasoning*, vol. 40, no. 2-3, pp. 117-

132, 2008.
7. P. Buneman, S. Davidson, W. Fan, C. Hara, and W. C. Tan, "Keys for XML," *Computer Networks*, vol. 39, no. 5, pp. 473-487, 2002.
8. S. Hartmann and S. Link, "Efficient reasoning about a robust XML key fragment," *ACM Transactions on Database Systems*, vol. 34, no. 2, article no. 10, 2009.
9. S. Hartmann and S. Link, "Numerical constraints on XML data," *Information and Computation*, vol. 208, no. 5, pp. 521-544, 2010.
10. G. Lausen, "Relational databases in RDF: keys and foreign keys," *Semantic Web, Ontologies and Databases, Lecture Notes in Computer Science vol. 5005*, V. Christophides et al. editors, Heidelberg: Springer, pp. 43-56, 2008.
11. B. C. Grau, I. Horrocks, B. Motik, B. Parsia, P. Patel-Schneider, and U. Sattler, "OWL 2: the next step for OWL," *Web Semantics*, vol. 6, no. 4, pp. 309-322, 2008.
12. Y. Sismanis, P. Brown, P. J. Haas, and B. Reinwald, "GORDIAN: efficient and scalable discovery of composite keys," *Proceedings of the 32nd International Conference on Very Large Data Bases*, Seoul, Korea, 2006, pp. 691-702.
13. J. Demetrovics, "On the number of candidate keys," *Information Processing Letters*, vol. 7, no. 6, pp. 266-269, 1978.
14. CA Technologies, CA ERwin data modeler: methods guide r7.3, https://support.ca.com/cadocs/0/e002961e.pdf.
15. S. Abiteboul, R. Hull, and V. Vianu, *Foundation of Database*, Reading, MA: Addison-Wesley, 1995.
16. R. Fagin, "Armstrong databases," IBM Research Laboratory, San Jose, CA, *Research report RJ3440-40926*, 1982.
17. B. Thalheim, *Dependencies in Relational Databases*, Stuttgart: B. G. Teubner, 1991.
18. W. D. Langeveldt and S. Link, "Empirical evidence for the usefulness of Armstrong relations in the acquisition of meaningful functional dependencies," *Information Systems*, vol. 35, no. 3, pp. 352-374, 2010.
19. S. Link, "Armstrong databases: validation, communication and consolidation of conceptual models with perfect test data," *Proceedings of the 9th Asia-Pacific Conference on Conceptual Modelling*, Melbourne, Australia, 2012, pp. 3-19.
20. B. Alexe, B. T. Cate, P. G. Kolaitis, and W. C. Tan, "Characterizing schema mappings via data examples," *ACM Transactions on Database Systems*, vol. 36, no. 4, article no. 23, 2011.
21. H. Mannila and K. J. Räihä, "Design by example: an application of Armstrong relations," *Journal of Computer and System Sciences*, vol. 33, no. 2, pp. 126-141, 1986.
22. F. de Marchi and J. M. Petit, "Semantic sampling of existing databases through informative Armstrong databases," *Information Systems*, vol. 32, no. 3, pp. 446-457, 2007.
23. J. Demetrovics, "On the equivalence of candidate keys with Sperner systems," *Acta Cybernetica*, vol. 4, no. 3, pp. 247-252, 1979.
24. C. Beeri, M. Dowd, R. Fagin, and R. Statman, "On the structure of Armstrong relations for functional dependencies," *Journal of the ACM*, vol. 31, no. 1, pp. 30-46, 1984.
25. Y. Huhtala, J. Karkkainen, P. Porkka, and H. Toivonen, "TANE: an efficient algorithm for discovering functional and approximate dependencies," *The Computer Journal*, vol. 42, no. 2, pp. 100-111, 1999.
26. H. Mannila and K. J. Räihä, "Algorithms for inferring functional dependencies from relations," *Data and Knowledge Engineering*, vol. 12, no. 1, pp. 83-99, 1994.
27. E. F. Codd, "A relational model of data for large shared data banks," *Communications of the ACM*, vol. 13, no. 6, pp. 377-387, 1970.
28. T. Imielinski and W. Lipski Jr, "Incomplete information in relational databases," *Journal of the ACM*, vol. 31, no. 4, pp. 761-791, 1984.
29. C. Zaniolo, "Database relations with null values," *Journal of Computer and System Sciences*, vol. 28, no. 1, pp. 142-166, 1984.
30. P. Atzeni and N. M. Morfuni, "Functional dependencies and constraints on null values in database relations," *Information and Control*, vol. 70, no. 1, pp. 1-31, 1986.
31. S. Hartmann, M. Kirchberg, and S. Link, 'Design by example for SQL table definitions with functional dependencies," *The VLDB Journal*, vol. 21, no. 1, pp. 121-144, 2012.
32. V. B. T. Le, S. Link, and M. Memari, "Discovery of keys from SQL tables," *Database Systems for Advanced Applications, Lecture Notes in Computer Science vol. 7238*, S. Lee et al. editors, Heidelberg: Springer Berlin, pp. 48-62, 2012.
33. C. J. Date, *Database Design and Relational Theory: Normal Forms and All That Jazz*, Sebastopol, CA: O'Reilly Media, 2012.
34. T. Eiter and G. Gottlob, "Identifying the minimal transversals of a hypergraph and related problems," *SIAM Journal on Computing*, vol. 24, no. 6, pp. 1278-1304, 1995.
35. J. Demetrovics and V. D. Thi, "Keys, antikeys and prime attributes," *Annales Universitatis Scientiarum Budapestinensis de Rolando Eotvos Nominatae Sectio Computatorica*, vol. 8, pp. 35-52, 1987.
36. S. Hartmann and S. Link, "When data dependencies over SQL tables meet the logics of paradox and S-3," *Proceedings of the 29th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, Indianapolis, IN, 2010, pp. 317-326.
37. S. Hartmann and S. Link, "The implication problem of data dependencies over SQL table definitions: axiomatic, algorithmic and logical characterizations," *ACM Transactions on Database Systems*, vol. 37, no. 2, article no. 13, 2012.
38. M. Levene and G. Loizou, "Axiomatisation of functional dependencies in incomplete relations," *Theoretical Computer Science*, vol. 206, no. 1-2, pp. 283-300, 1998.

Van Bao Tran Le

Van Le received a Master's degree in computer science from Ho Chi Minh City University of Technology, Vietnam. Currently, she is a PhD student of Information Systems at the Victoria University of Wellington. Her main area of research concerns the acquisition and visualization of semantics in data.

Sebastian Link

Sebastian Link received a PhD in Information Systems from Massey University in 2005. Currently, he is Associate Professor of Computer Science at the University of Auckland. His research interests include conceptual modeling, semantics in databases, foundations of markup languages, and the application of discrete mathematics to computer science. Sebastian has published more than 75 research papers, and served as a reviewer for numerous conferences and journals. He is a member of the editorial board of the journal Information Systems.

Mozhgan Memari

Mozhgan Memari received a Master's degree in information technology management from Alzahra University in Tehran, Iran. Currently, she is a PhD student of Computer Science at the University of Auckland. Her main area of research pertains to semantics in incomplete databases.