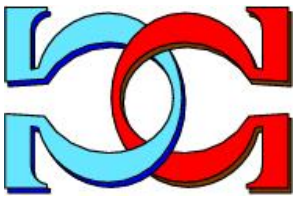

CDMTCS
Research
Report
Series

Effective Recognition and Visualization of Semantic Requirements by Perfect SQL Samples

Van Bao Tran Le
Victoria University of Wellington,
Wellington, New Zealand

Sebastian Link
University of Auckland,
Auckland, New Zealand

Flavio Ferrarotti
Victoria University of Wellington,
Wellington, New Zealand

CDMTCS-451
December 2013

Centre for Discrete Mathematics and
Theoretical Computer Science

Effective Recognition and Visualization of Semantic Requirements by Perfect SQL Samples

Van Bao Tranh Le
Victoria University of Wellington
Wellington, New Zealand
van.t.le@vuw.ac.nz

Sebastian Link
The University of Auckland, Private Bag 92019
Auckland, New Zealand
s.link@auckland.ac.nz

Flavio Ferrarotti
Victoria University of Wellington
Wellington, New Zealand
flavio.ferrarotti@vuw.ac.nz

December 17, 2013

Abstract

SQL designs result from methodologies such as UML or Entity-Relationship models, description logics, or relational normalization. Independently of the methodology, sample data is promoted by academia and industry to visualize and consolidate the designs produced. SQL table definitions are a standard-compliant encoding of their designers' perception about the semantics of an application domain. Armstrong sample data visualize these perceptions. We present a tool that computes Armstrong samples for different classes of SQL constraints. Exploiting our tool, we then investigate empirically how these Armstrong samples help design teams recognize domain semantics. New measures empower us to compute the distance between constraint sets in order to evaluate the usefulness of our tool. Extensive experiments confirm that users of our tool are likely to recognize domain semantics they would overlook otherwise. The tool thereby effectively complements existing design methodologies in finding quality schemata that process data efficiently.

Keywords: Armstrong database, Empirical measure, Functional dependency, SQL, Uniqueness constraint

1 Introduction

Design methodologies such as UML and Entity-relationship models, description logics, and relational normalization all have the ultimate goal to derive a quality database schema on which most frequent queries and most frequent updates can be processed efficiently. The output that these methodologies produce are usually relational database schemata. In practice, however, relational designs must be transformed into a standard-compliant SQL table format. While SQL is founded on the relational model of data, there is a strong disparity between them. For example, SQL permits occurrences of null markers and duplicate tuples to ease and speed-up data acquisition and processing. It is therefore a challenging task for design teams to find a suitable SQL table implementation. In particular, one that captures not only the structure but also the semantics of the application domain. Shortcomings in the acquisition of domain semantics are known to be very costly, and often require expensive data cleaning and database repair techniques once the database is in use. These are some of the reasons why there is great consensus among academia and the commercial world that the use of good data samples can greatly benefit a more complete acquisition of requirements, and also help with the validation and consolidation of schemata produced automatically by design tools, as well as deliver a justification of the designs that can be effectively communicated to different stake-holders of the database.

In essence, Armstrong samples are data samples that satisfy the constraints a design team currently perceives as semantically meaningful for the application domain, and violate all those constraints that they currently perceive meaningless. Armstrong samples are therefore visualizations of abstract sets of constraints that encode real-world semantics. Intuitively, humans can learn a lot from Armstrong samples. They therefore hold great promise for the data-driven discovery of real-world application semantics. We illustrate this promise by revisiting a classical example.

Suppose our design team has arrived at the relation schema Contact with attributes *Street*, *City*, and *ZIP*, and the set Σ of the following functional dependencies (FDs): *Street*, *City* $\rightarrow$ *ZIP* and *ZIP* $\rightarrow$ *City*. This is the classical example of a schema that is in Third normal form, but not in Boyce-Codd normal form. Normalization algorithms stop here, and cannot provide any further guidance on how to implement the relation schema within an SQL table definition. Magically, we may now produce an Armstrong *table* for the given set Σ of FDs and `NOT NULL` constraints, say on *Street* and *ZIP*, e.g. the table on the left:

Street	*City*	*ZIP*
03 Hudson St	Manhattan	10001
03 Hudson St	Manhattan	10001
70 King St	Manhattan	10001
70 King St	San Fran	94107
15 Maxwell St	San Fran	94129
46 State St	`ni`	60609

Street	*City*	*ZIP*
03 Hudson St	Manhattan	10001
70 King St	Manhattan	10001
70 King St	San Fran	94107
35 Lincoln Blvd	San Fran	94129
15 Maxwell St	`ni`	60609
15 Maxwell St	`ni`	60609

An inspection of this table shows that our specification of FDs does not exclude occurrences of duplicate tuples. More precisely, Σ does not imply any uniqueness con-

straints (UCs) over SQL tables. At this stage, our design team decides that the FD *Street, City* $\rightarrow$ *ZIP* should be replaced by the stronger UC $u(Street, City)$, meaning that there cannot be any different rows with matching total values on both *Street* and *City*. Note the interpretation of the null marker `ni` as *no information*, i.e., a value may not exist, or it may exist but is currently unknown. This is the interpretation that SQL uses [22]. The occurrence of `ni` in the table above indicates that the column *City* is nullable. An Armstrong table for the revised constraint set is shown on the right above. Looking at the last two rows of this table, the design team notices that the UC $u(Street, ZIP)$ is still not implied by the constraints specified so far. As the UC is considered to be meaningful, the designers decide to specify this constraint as well. Thus, the design team finally arrives at the following SQL table implementation

```
CREATE TABLE Contact (
  Street VARCHAR,
  City VARCHAR,
  ZIP INT,
  UNIQUE(Street,City),
  PRIMARY KEY(Street,ZIP),
  CHECK(Q = 0));
```

```
SELECT COUNT(*) FROM Contact c1                                   Q
WHERE c1.ZIP IN (
 SELECT ZIP FROM Contact c2
 WHERE c1.ZIP=c2.ZIP AND
 (c1.City <> c2.City OR
 (c1.City IS NULL AND c2.City IS NOT NULL) OR
 (c1.City IS NOT NULL AND c2.City IS NULL)));
```

where the state assertion Q on the right enforces *ZIP* $\rightarrow$ *City*.

Contributions. As our main contribution we investigate empirically how Armstrong samples in standard-compliant SQL format help design teams recognize domain semantics. For this contribution we need to overcome several obstacles. In previous research, Armstrong samples have been investigated in the context of the relational model of data only. Thus, Armstrong relations cannot show the delicate interactions between SQL constraints. However, we draw from our recent work where we developed algorithms to compute Armstrong tables for different classes of SQL constraints. Our first contribution in this paper is a tool that is the first to implement these algorithms. We describe the graphical user interface of the tool and its functionality. Our second contribution is the definition of several empirical measures to assess the effectiveness of using our tool. In essence, *soundness* measures how many of the as meaningful perceived constraints are actually meaningful, *completeness* measures how many of the actually meaningful constraints are perceived as meaningful, and *proximity* combines soundness and completeness. Our measures assess the quality of a constraint set with respect to a target constraint set, and therefore qualify naturally for the use in automated assessment tools, e.g., in database courses. In the example above, the target set Σ_t consists of the UCs *u(Street,City)*, *u(Street,ZIP)*, and the FD *ZIP* $\rightarrow$ *City*. The original set Σ of FDs consisting of *Street,City* $\rightarrow$ *ZIP* and *ZIP* $\rightarrow$ *City* is fully sound with respect to Σ_t since both FDs in Σ are implied by Σ_t. While Σ is also fully complete with respect to FDs, it is fully incomplete with respect to UCs. That is, every FD implied by Σ_t is also implied by Σ, but no UC implied by Σ_t is implied by Σ. In our third contribution we present an analysis of our extensive experiments with our tool. Our experiments determine what and how much design teams learn about the application domain in addition to what they know prior to inspecting Armstrong tables produced by our tool. Our analysis shows

that inspecting Armstrong tables has no impact on recognizing meaningless SQL constraints which are incorrectly perceived as meaningful, but inspecting Armstrong tables empowers design teams to recognize nearly all meaningful SQL constraints that are incorrectly perceived as meaningless. These results empirically confirm our intuition that *the satisfaction of meaningless SQL constraints is nearly impossible to be observed, and the violation of meaningful SQL constraints is almost certain to be observed in Armstrong tables.*

Organization. We summarize related work in Section 2 and define the necessary framework in Section 3. Our tool is presented in Section 4, and an overview of the experimental design is given in Section 5. Our measures are defined in Section 6, and a quantitative and qualitative analysis is presented in Section 7. We conclude in Section 8 where we also briefly comment on future work.

2 Related work

Armstrong databases have been regarded intuitively as a conceptual tool helpful with the acquisition of domain semantics [5, 17, 18, 19, 20]. Theoretical work on computational and structural properties of Armstrong databases in the relational model of data and the Entity-Relationship model are manifold, including [2, 3, 6, 12, 18, 21] with survey papers [5, 17].

One of the most important extensions of Codd's basic relational model [4] is partial information to cope with the high demand for the correct handling of such information in real-world applications. In the literature many interpretations of null markers have been proposed such as "missing" or "value unknown at present", "no information", and "inapplicable". Our tool can handle constraints under the two most popular interpretations: null marker occurrences interpreted as "value unknown at present" are denoted by *unk*, and occurrences that are interpreted as "no information" are denoted by *ni*. As the latter is the one used by SQL, our presentation in this paper will mostly focus on the "no information" interpretation.

In recent research we have established a theory of Armstrong tables for different classes of `NOT NULL` constraints, uniqueness constraints (UCs) and functional dependencies (FDs) under occurrences of either the *unk* marker, or the *ni* marker [7, 9, 10, 11, 14]. The present article is the first to present a tool for the computation of Armstrong tables under different classes of constraints and the two different null marker interpretations. The tool subsumes those implemented for relational databases as an idealized special case [18, 19].

Our tool forms the critical basis for our empirical investigations into the usefulness of Armstrong tables. We utilize the tool to compute Armstrong tables on-the-fly and in response to inputs by design teams. The teams then inspect the Armstrong tables together with domain experts in order to consolidate their perception of the domain semantics in form of a set of SQL constraints. For the class of FDs over strictly relational databases our previous research has shown that the use of Armstrong relations is likely to increase the recognition of meaningful FDs, but unlikely to increase the recognition of meaningless FDs [13]. Since the interaction of SQL constraints is more involved than that for their

idealized relational counterparts, one would naturally assume that sample data becomes even more useful. In the current paper we therefore extend the experimental framework from [13] to investigate the usefulness, and exploit our tool in actual experiments. Our new measures do not just address the single class of FDs, but the two classes of UCs and FDs. This is necessary as `NOT NULL` constraints, UCs and FDs interact non-trivially over SQL data, in contrast to relational data where all attributes are `NOT NULL` and UCs are simply subsumed by FDs. In addition to the measures of *soundness*, *completeness* and *proximity*, we introduce here their *relative versions*. These illustrate best our findings: almost all UCs and FDs that can possibly be recognized are actually recognized by inspecting Armstrong tables, and that Armstrong tables do not have an impact on recognizing meaningless UCs and FDs incorrectly perceived as meaningful.

3 Preliminaries

In this section we define the syntax and semantics for the different classes of constraints under different interpretations of null markers. While an exact understanding of the semantics is not necessary to appreciate our main results, they are required to fully appreciate the features implemented in our tool.

Let $\mathfrak{H} = \{H_1, H_2, \ldots\}$ be a countably infinite set of symbols, called *(column) headers*. A *table schema* is a finite non-empty subset T of $\mathfrak{H}$. Each header H of a table schema T is associated with an infinite domain $dom(H)$ of the possible values that can occur in column H. To encompass partial information every column may contain occurrences of a null marker, $\mathtt{ni} \in dom(H)$.

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 H to represent $\{H\}$. A *row* over T is a function $r : T \to \bigcup_{H \in T} dom(H)$ with $r(H) \in dom(H)$ for all $H \in T$. For $X \subseteq T$ let $r(X)$ denote the restriction of the row r over T to X. An *SQL table* t over T is a finite multi-set of rows over T. For rows r_1 and r_2 over T, r_1 *subsumes* r_2 if for all $H \in T$, $r_1(H) = r_2(H)$ or $r_2(H) = \mathtt{ni}$.

For a row r over T and a set $X \subseteq T$, r is said to be X-total if for all $H \in X$, $r(H) \neq \mathtt{ni}$. Similar, an SQL table t over T is said to be X-total, if every row r of t is X-total. An SQL table t over T is said to be *total* if it is T-total.

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 an SQL table t over T, denoted by $\models_t nfs(T_s)$, if and only if t is T_s-total. In practice, the NFS consists of those attributes declared `NOT NULL` in the SQL table definition.

An *SQL functional dependency* (SFD) over a table schema T is an expression $X \to Y$ where $X, Y \subseteq T$. An SQL table t over T satisfies the SFD $X \to Y$ if for all rows $r, r' \in t$ the following holds: if $r(X) = r'(X)$ and r, r' are both X-total, then $r(Y) = r'(Y)$ [16]. An *SQL uniqueness constraint* (SUC) over table schema T is an expression $u(X)$ where $X \subseteq T$. An SQL table t satisfies the SUC $u(X)$ if for all rows $r, r' \in t$ the following holds: if $r(X) = r'(X)$ and both r and r' are X-total, then $r = r'$. For examples, both SQL tables from the introduction satisfy *ZIP* $\to$ *City*. While the left table violates every SUC, the right table satisfies u(*Street,City*) but violates u(*Street,ZIP*).

Let $\mathcal{C}$ be a class of constraints, for example, the combined class of `NOT NULL` constraints, SUCs and SFDs. We say for a set $\Sigma \cup \{\varphi\}$ of constraints from $\mathcal{C}$ over table schema T that Σ *implies* φ, denoted by $\Sigma \models \varphi$, if for every SQL table t over T that satisfies every constraint in Σ, t also satisfies φ. For example, the right SQL table from the introduction shows that the set Σ consisting of *ZIP* $\rightarrow$ *City*, *u*(*Street,City*), and the NFS *nfs*(*Street,ZIP*) does not imply *u*(*Street,ZIP*).

For a set Σ of constraints in $\mathcal{C}$ over table schema T, we say that an SQL table t over T is $\mathcal{C}$-*Armstrong* for Σ if t satisfies every constraint in Σ, and violates every constraint in $\mathcal{C}$ over T that is not implied by Σ. For example, the table on the right from the introduction is Armstrong for the set Σ containing *ZIP* $\rightarrow$ *City*, *u*(*Street,City*), and *nfs*(*Street,ZIP*). By inspecting this table, we know that Σ does not imply *City* $\rightarrow$ *Street* nor *u*(*Street,ZIP*), but does imply *Street,ZIP* $\rightarrow$ *City* and *u*(*Street,City*).

For our experiments we will focus on SQL constraints exclusively. Constraints, however, can also be defined on tables that feature the Codd null marker `unk`, instead of `ni`. In that case we speak of *Codd tables*. For a Codd table t over T, the set $Poss(t)$ of all possible worlds relative to t is defined by

$$Poss(t) = \{t' \mid t' \text{ is a table over } T \text{ and there is a bijection } b : t \rightarrow t' \text{ such that } \forall r \in t, r \text{ is sub- sumed by } b(r) \text{ and } b(r) \text{ is } T\text{-total}\}.$$

A *Codd functional dependency* (CFD) over table schema T is an expression $\diamond(X \rightarrow Y)$ where $X, Y \subseteq T$. A Codd table t over T satisfies $\diamond(X \rightarrow Y)$ if there is some $p \in Poss(t)$ such that for all rows $r, r' \in p$ the following holds: if $r(X) = r'(X)$, then $r(Y) = r'(Y)$. A *Codd uniqueness constraint* (CUC) over table schema T is an expression $\diamond u(X)$ where $X \subseteq T$. A Codd table t satisfies $\diamond u(X)$ if there is some $p \in Poss(t)$ such that for all rows $r, r' \in p$ the following holds: if $r(X) = r'(X)$ and both r and r' are X-total, then $r = r'$. The notions of implication and Armstrong tables, defined in the context of SQL tables above, are defined analogously in the context of Codd tables.

Algorithms to compute $\mathcal{C}$-Armstrong tables were recently developed for the classes $\mathcal{C}$ of `NOT NULL` constraints and i) SUCs in [10], ii) SUCs and SFDs in [9], iii) CUCs in [14], and iv) CUCs and CFDs in [7]. Our tool implements all of these algorithms, but for the experiments we focus on the class ii) above.

4 A Tool to Recognize and Visualize Domain Semantics

In this section we present the functionality of our tool. It is the first to implement recently published algorithms for computing Armstrong tables [7, 9, 10, 14]. It is a Web application, developed in the .NET framework, and operates on common Internet browsers such as Firefox, Google Chrome, and Internet Explorer. We invite the reader to visit `http://armstrongtable.sim.vuw.ac.nz` to experiment with the tool. A screenshot of the GUI is shown in Figure 1.

The workflow of the tool exploits the following components. Firstly, the user selects a context which fixes the interpretation of occurring null markers and the class of constraints. If `ni` is selected, the user chooses from the class of SQL UCs, or the combined

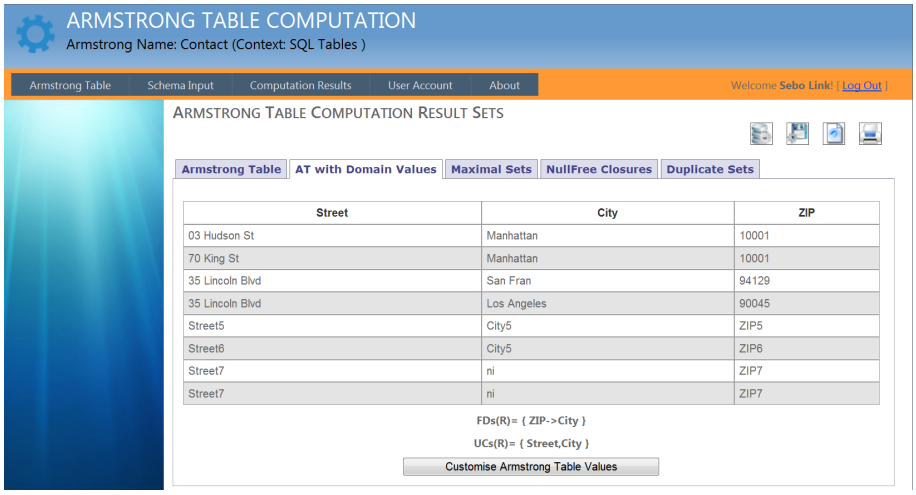

Figure 1: GUI of web-based tool: Armstrong table with some customized domain values

class of SQL UCs and SQL FDs. Otherwise, the user chooses from the class of Codd UCs, or the combined class of Codd UCs and Codd FDs.

Secondly, the user specifies i) the table schema and which column headers are declared `NOT NULL`, ii) the sets of constraints for the classes of constraints from the context, and iii) optionally, some designated domain values to populate the Armstrong table.

Thirdly, the user choose from several algorithms. Foremost, a $\mathcal{C}$-Armstrong table can be computed for the set Σ of input constraints from $\mathcal{C}$. The tool populates the table by either domain values provided or by artificial values. The user can replace values in the table by new values such that the new table is also $\mathcal{C}$-Armstrong for Σ. Depending on the class of constraints selected, other algorithms include the computation of closures for a collection of column headers, the set of anti-keys, the set of maximal sets for each column header, and duplicate sets. These notions are defined in our previous work [7, 9, 10, 14].

Finally, users of the tool have the option to export the table to an XML file, or print it out. The XML file may be used in many other common applications, including Microsoft Office, OpenOffice, or Apple's iWork.

5 Experimental Design

Our aim is to evaluate how well Armstrong tables help design teams recognize domain semantics. We thus ask how much domain semantics they learn by inspecting Armstrong tables *in addition* to what they can learn without them.

5.1 Overview

Naturally, our experiment consists of two phases. In the first phase, each design team i is given the same application domain in form of a table schema and `NOT NULL` constraints. A natural language description accompanies the schema definition, and domain experts are available to answer questions about the domain. The experts have no database

background and do not answer questions about database constraints. After internal discussion and consultation with the experts, each team i writes down a cover Σ_1^i of SQL UCs and FDs that they perceive as meaningful for the domain. For each team i, our tool computes an Armstrong table for Σ_1^i and the `NOT NULL` constraints.

In the second phase, each team i revises their constraint set Σ_1^i with the help of the Armstrong table for Σ_1^i. Again, domain experts can be consulted and the Armstrong table can be used to communicate with them. Teams are provided with Armstrong tables for their revised constraint sets until they are happy with the outcome. The final constraint set is denoted by Σ_2^i.

For each team i, we use different measures (see the next section) to compare Σ_1^i with our target set Σ_t of constraints, and to compare Σ_2^i with Σ_t. If the latter comparison results in a higher similarity than the former comparison, we conclude that Armstrong tables are indeed useful with respect to the measure. The comparison between Σ_1^i and Σ_t gives us a baseline of what team i can still possibly learn by inspecting Armstrong tables. Similarly, the comparison between Σ_2^i and Σ_t tells us how much team i has actually learned by inspecting Armstrong tables. Graphs will illustrate for each team the difference between how much they could possibly learn and how much they have actually learned.

5.2 Design teams and domain experts

The experiments involved 50 design teams from three universities: the University of Auckland, the Victoria University of Wellington, and the Lotus University of Vietnam. The students took third year database courses, were familiar with the semantics of SQL constraints, and that the Armstrong tables satisfy the constraints they currently perceive as meaningful and violate every constraint they currently perceive as meaningless. Each team consisted of 2 or 3 students.

The authors of this paper acted as domain experts for the given application domain. They were present during the experiments to clarify questions by the teams. Teams were not given advice about the specification of their constraint sets. For example, questions like "Does this UC make sense?" were not answered. In practice, there may not exist a unique target set, because the available information may not identify a unique constraint set. For conducting the experiment, however, the presence of the domain experts and their consensus on the target set guarantees that the quality of constraint sets can be measured transparently.

5.3 Application domain and target

As application domain we used the schema

$$\text{WORK}=\{P(roj),E(mp),D(ate),R(ole),H(rs)\}$$

with the following description. The schema records information about the number *Hrs* of hours (e.g., *5*) that an employee *Emp* (e.g., Dilbert) works on a project *Proj* (e.g., *Blue*) in a role *Role* (e.g., *Programmer*) at a day *Date* (e.g., *Oct 5*). The null-free subschema of WORK is $nfs(Emp, Date)$, i.e., `ni` must not occur in the *Emp* and *Date* columns.

Σ_t contains $nfs(Emp, Date)$ and the following SQL UCs and FDs. The UC *u(Emp,Date)* states that there cannot be different rows with the same employee and the same date. The FD *Proj,Emp* $\rightarrow$ *Role* says that in each project, each employee has a unique role. The FD *Project,Role* $\rightarrow$ *Hrs* says that the project and role together determine the number of hours. Finally, the FD *Proj,Emp* $\rightarrow$ *Hrs* says that the project and employee together determine the number of hours. The last FD is not implied by the other UCs and FDs in Σ_t [11].

5.4 Limitations

These include issues such as students acting as database designers, the familiarity of students with the application domain, the number and size of the application domain, time constraints, the assumption that domain experts are present, and that there is consensus among them. The discussion of these limitations from the idealized relational case [13] is also valid for SQL constraints.

6 Quality measures

We define new measures to compare constraint sets, and illustrate these on the following running example from our actual experiment. Design team 8 handed in the following sets of constraints:

- $\Sigma_1^8 = \{$*u(Emp,Proj,Date)*; *Proj,Role* $\rightarrow$ *Hrs*$\}$
- $\Sigma_2^8 = \{$*u(Emp,Date)*; *Proj,Role* $\rightarrow$ *Hrs*; *Proj,Emp* $\rightarrow$ *Role*$\}$

Our measures will enable us to compare these sets to the target set Σ_t. As the NFS $nfs(Emp, Date)$ is given, it belongs to all sets Σ_j^i, $i = 1, \ldots, 50$ and $j = 1, 2$.

Soundness measures which of the constraints perceived meaningful by a team, are actually meaningful. Here, actually meaningful are the constraints implied by the target set Σ_t. The UCs implied by a set Σ of UCs and FDs and $nfs(T_s)$ is defined as $s(\Sigma) := \{u(X) \mid \Sigma \models u(X)\}$. Soundness for UCs is thus the ratio between the as meaningful perceived UCs that are implied by Σ_t and all the as meaningful perceived UCs:

$$sound^u_{\Sigma_t}(\Sigma) = \frac{\mid s(\Sigma) \cap s(\Sigma_t) \mid}{\mid s(\Sigma) \mid},$$

and $sound^u_{\Sigma_t}(\Sigma) := 1$, if $s(\Sigma) = \emptyset$. For the measures of FDs we exploit the notion of a closure $X^*_\Sigma = \{H \in T \mid \Sigma \models X \rightarrow H\}$ for a set X of headers under Σ. Let $\mathcal{P}_0(T)$ denote the set of all non-empty, proper subsets of T. Then the soundness for FDs is the ratio between the header sets in $\mathcal{P}_0(T)$ whose closure under Σ is contained in the closure under Σ_t, and $\mathcal{P}_0(T)$:

$$sound^f_{\Sigma_t}(\Sigma) = \frac{\mid \{X \in \mathcal{P}_0(T) \mid X^*_\Sigma \subseteq X^*_{\Sigma_t}\} \mid}{\mid \mathcal{P}_0(T) \mid}.$$

For our running example we obtain $sound^u_{\Sigma_t}(\Sigma_1^8) = 1$, $sound^f_{\Sigma_t}(\Sigma_1^8) = 30/30 = 1$, and $sound^u_{\Sigma_t}(\Sigma_2^8) = 8/8 = 1$, $sound^f_{\Sigma_t}(\Sigma_2^8) = 1$.

Completeness measures which of the actually meaningful constraints are also perceived as meaningful by a team. Completeness for UCs is thus the ratio between the as meaningful perceived UCs that are implied by Σ_t and all the actually meaningful UCs:

$$complete^u_{\Sigma_t}(\Sigma) = \frac{\mid s(\Sigma) \cap s(\Sigma_t) \mid}{\mid s(\Sigma_t) \mid},$$

and $complete^u_{\Sigma_t}(\Sigma) := 1$, if $s(\Sigma_t) = \emptyset$. Completeness for FDs is the ratio between the header sets in $\mathcal{P}_0(T)$ whose closure under Σ_t is contained in the closure under Σ, and $\mathcal{P}_0(T)$:

$$complete^f_{\Sigma_t}(\Sigma) = \frac{\mid \{X \in \mathcal{P}_0(T) \mid X^*_{\Sigma_t} \subseteq X^*_{\Sigma}\} \mid}{\mid \mathcal{P}_0(T) \mid}.$$

For our running example, we obtain $complete^u_{\Sigma_t}(\Sigma^8_1) = 4/8 = 0.5$, $complete^f_{\Sigma_t}(\Sigma^8_1) = 26/30 = 0.8$, and $complete^u_{\Sigma_t}(\Sigma^8_2) = 8/8 = 1$, $complete^f_{\Sigma_t}(\Sigma^8_2) = 29/30 \approx 0.97$.

Proximity measures how close two sets of constraints are. For UCs it is the ratio between the as meaningful perceived UCs that are implied by Σ_t and all the actually meaningful and all the as meaningful perceived UCs:

$$prox^u(\Sigma, \Sigma_t) = \frac{\mid s(\Sigma) \cap s(\Sigma_t) \mid}{\mid s(\Sigma) \cup s(\Sigma_t) \mid},$$

and $prox^u(\Sigma, \Sigma_t) := 1$, if $s(\Sigma_t) \cup s(\Sigma_t) = \emptyset$. The complement $dist^u(\Sigma, \Sigma_t) =\mid (s(\Sigma) \cup s(\Sigma_t)) - (s(\Sigma) \cap s(\Sigma_t)) \mid$ defines a metric on equivalent sets of UCs. For FDs, completeness is the ratio between the header sets in $\mathcal{P}_0(T)$ whose closure under Σ_t is the same as the closure under Σ, and $\mathcal{P}_0(T)$:

$$prox^f(\Sigma, \Sigma_t) = \frac{\mid \{X \in \mathcal{P}_0(T) \mid X^*_{\Sigma_t} = X^*_{\Sigma}\} \mid}{\mid \mathcal{P}_0(T) \mid}.$$

Similar to UCs, $dist^f(\Sigma, \Sigma_t) =\mid \{X \in \mathcal{P}_0(T) \mid X^*_{\Sigma_t} \neq X^*_{\Sigma}\} \mid$ defines a metric on equivalent sets of FDs. For our example, we obtain $prox(\Sigma^8_1, \Sigma_t) = 4/8 = 0.5$, $prox^f(\Sigma^8_1, \Sigma_t) = 26/30 = 0.8$, and $prox^u(\Sigma^8_2, \Sigma_t) = 8/8 = 1$, $prox^f(\Sigma^8_2, \Sigma_t) = 29/30 \approx 0.97$.

For the best insight into the usefulness of Armstrong tables we present, for each of the measures, the difference between how much Armstrong tables can *possibly* and *actually* improve the measurement. In what follows we refer by $measure(\Sigma)$ to one of $sound^u_{\Sigma_t}(\Sigma)$, $sound^f_{\Sigma_t}(\Sigma)$, $complete^u_{\Sigma_t}(\Sigma)$, $complete^f_{\Sigma_t}(\Sigma)$, $prox^u(\Sigma, \Sigma^t)$ and $prox^f(\Sigma, \Sigma^t)$. Then we define $possible\text{-}gain\text{-}measure^i := (1 - measure(\Sigma^i_1)) \cdot 100\%$, and $actual\text{-}gain\text{-}measure^i := (measure(\Sigma^i_2) - measure(\Sigma^i_1)) \cdot 100\%$. Note that actual gains can also be negative. For our running example we obtain

- $possible\text{-}gain\text{-}UC\text{-}sound^8 = 0\%$,
- $actual\text{-}gain\text{-}UC\text{-}sound^8 = 0\%$,
- $possible\text{-}gain\text{-}FD\text{-}sound^8 = 0\%$,
- $actual\text{-}gain\text{-}FD\text{-}sound^8 = 0\%$,

- *possible-gain-UC-complete*$^8 = 50\%$,
- *actual-gain-UC-complete*$^8 = 50\%$,
- *possible-gain-FD-complete*$^8 = 20\%$, and
- *actual-gain-FD-complete*$^8 = 17\%$.

7 Data Analysis

We analyze our experiments with 50 design teams quantitatively and qualitatively. Due to lack of space we will focus on our main findings and will not analyze proximity separately, as it combines soundness and completeness.

7.1 Quantitative Analysis

The following tables show some statistics for the actual gains teams achieved by inspecting Armstrong tables. Means are arithmetic means.

Gain-in-soundness in percent				
Class	*Min*	*Mean*	*Median*	*Max*
UCs	-33	4	0	44
FDs	-13	0	0	14

Gain-in-completeness in percent				
Class	*Min*	*Mean*	*Median*	*Max*
UCs	0	24	25	75
FDs	0	11	12	27

The statistics confirm our intuition that Armstrong tables are unlikely to help design teams recognize meaningless constraints that they perceive meaningful prior to inspecting a corresponding Armstrong table. While a small number of teams have added new meaningless constraints (a negative gain in soundness), a small number of different teams have removed meaningless constraints (a positive gain). On average, these gains and losses even each other out. The statistics also confirm our intuition that Armstrong tables are likely to help design teams recognize meaningful constraints they perceive meaningless prior to inspecting an Armstrong table. Indeed, no meaningful constraints are removed at all, but a significant number of meaningful constraints is added on average.

Figures 2 to 5 strengthen these observation. For soundness it shows that actual gains are similar to no gains. For completeness it shows that possible and actual gains nearly coincide.

7.2 Qualitative Analysis

A qualitative analysis of the soundness confirms that only few teams add or remove meaningless constraints after inspecting Armstrong tables. Indeed, three teams added $u(EP)$ and one team added $u(EPR)$, and $PE \to D$, $PD \to E$, and $ER \to P$ were each added by at most one team. Three teams removed $u(EP)$ and one team removed $u(DERH)$. $PR \to E$, $PR \to D$, $ER \to P$, $DH \to E$, $R \to P$ were each removed by at most one team.

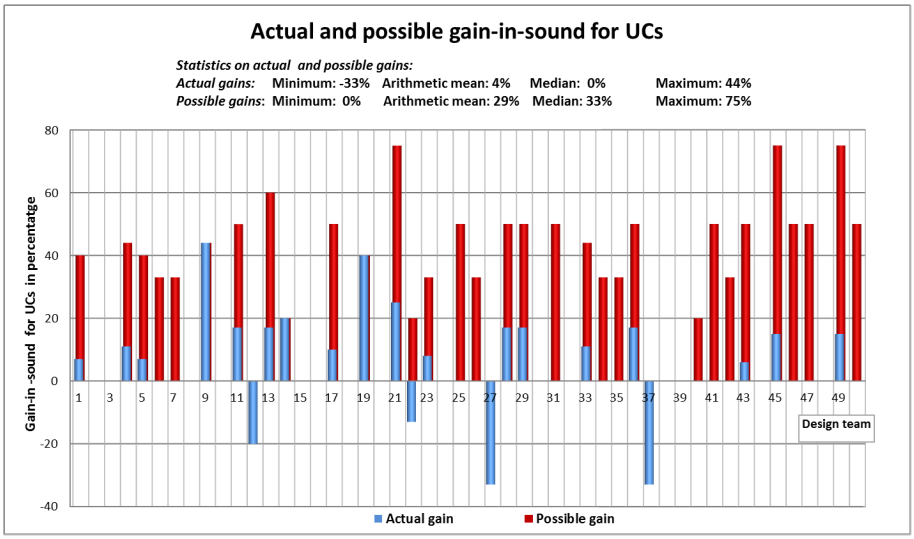

Figure 2: Actual and Possible Gains for Soundness of Uniqueness Constraints

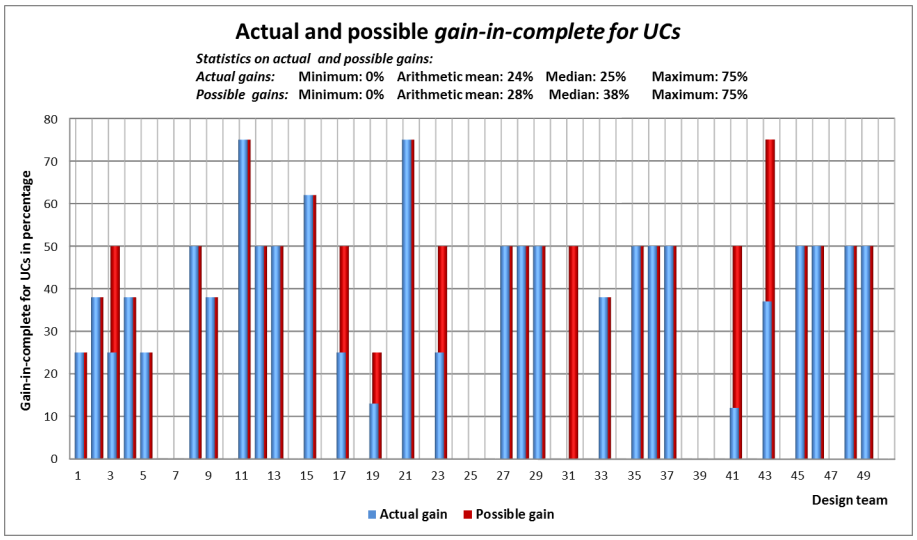

Figure 3: Actual and Possible Gains for Completeness of Uniqueness Constraints

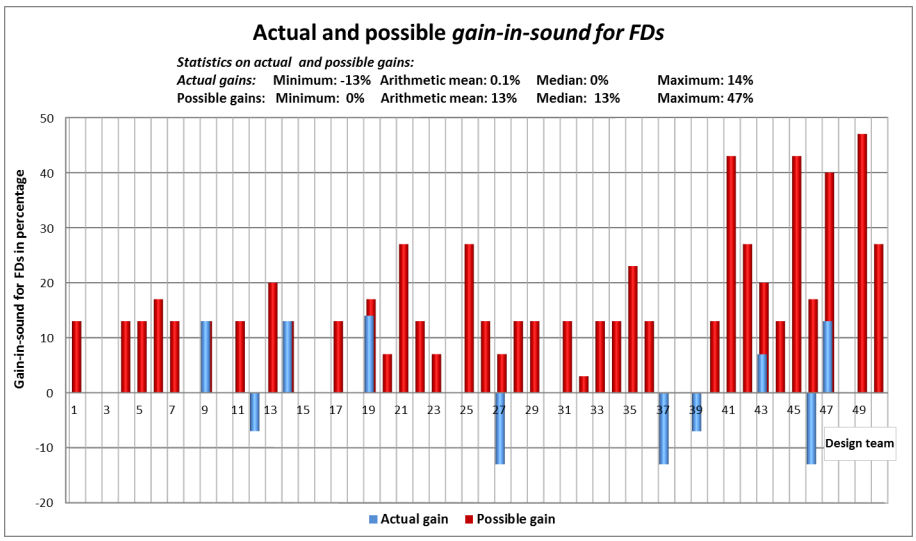

Figure 4: Actual and Possible Gains for Soundness of Functional Dependencies

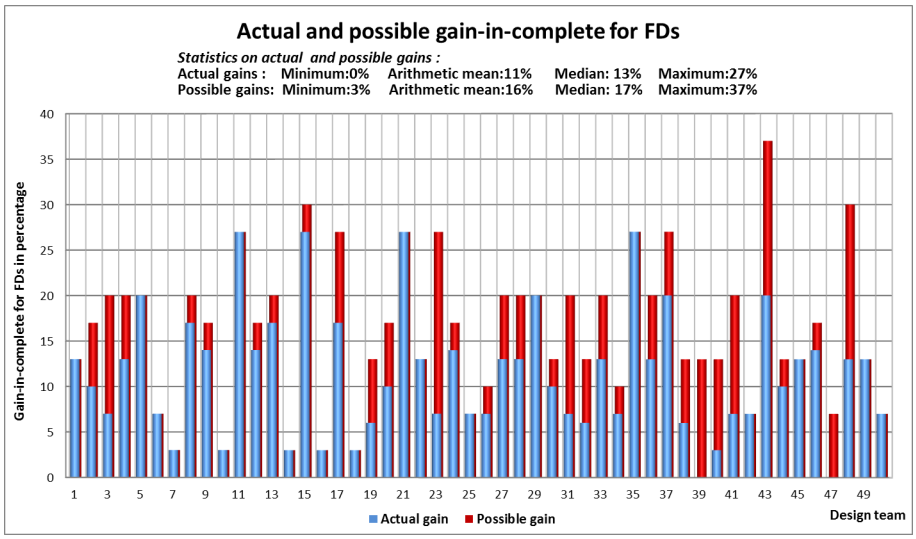

Figure 5: Actual and Possible Gains for Completeness of Functional Dependencies

A qualitative analysis of the completeness confirms that significant numbers of teams add meaningful constraints after inspecting Armstrong tables. Indeed, 22 teams added $u(ED)$, and 41 teams added some meaningful FD: 18 teams added $PR \to H$, 15 teams added $PE \to H$, and 14 teams added $EP \to R$. The numbers also suggest that there is no particular pattern on which meaningful FD is added. Importantly, no team removed any meaningful constraint.

8 Conclusion and Future Work

Conceptual methodologies produce relational approximations of database designs that still need to be converted into real-world SQL table definitions. In particular, the semantics of the application domain must be encoded in form of SQL constraints. We have investigated how much perfect SQL sample data, in form of Armstrong tables, help design teams recognize uniqueness constraints and functional dependencies that are meaningful for the underlying application domain. For this purpose we developed a tool that implements recent algorithms for the computation of Armstrong tables. The tool was then used in our experiments to create Armstrong tables for sets of these SQL constraints that design teams perceive as meaningful. New measures were exploited to confirm empirically that the inspection of Armstrong tables is likely to help design teams recognize nearly all meaningful SQL constraints they did not recognize before, but unlikely to help them recognize any actually meaningless SQL constraints they perceive as meaningful. The results extend previous findings for purely relational functional dependencies. They suggest to use our tool as early as possible during requirements acquisition and to exploit it for the consolidation and visualization of database designs produced by popular conceptual design methodologies.

In future work we will address the limitations of our experiments, include other classes of SQL constraints such as cardinality and referential constraints [8, 15, 21], gather data on how the size of Armstrong tables affects the recognition of domain semantics, implement our measures for the automated assessment and feedback of exercises in database courses, and combine our approach with techniques from natural language processing [1]. In particular, our results may not apply in this form when different constraints or constraints on several tables are considered, such as foreign keys.

Acknowledgement. This research is supported by the Marsden fund council from Government funding, administered by the Royal Society of New Zealand. We express our sincere gratitude to Pavle Mogin and Hui Ma for their kind assistance during the data gathering process.

References

[1] Albrecht, M., Buchholz, E., Düsterhöft, A., Thalheim, B.: An informal and efficient approach for obtaining semantic constraints using sample data and natural language processing. In: Libkin, L., Thalheim, B. (eds.) Semantics in Databases. LNCS, vol. 1358, pp. 1–28. Springer, Heidelberg (1998)

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

[3] Beskales, G., Ilyas, I., Golab, L.: Sampling the repairs of functional dependency violations under hard constraints. PVLDB 3(1), 197–207 (2010)

[4] Codd, E.F.: A relational model of data for large shared data banks. Commun. ACM 13(6), 377–387 (1970)

[5] Fagin, R.: Armstrong databases. Tech. Rep. RJ3440(40926), IBM Research Laboratory, San Jose, California, USA (1982)

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

[7] Ferrarotti, F., Hartmann, S., Le, V., Link, S.: Codd table representations under weak possible world semantics. In: Hameurlain, A., Liddle, S.W., Schewe, K.D., Zhou, X. (eds.) DEXA 2011, Part I. LNCS, vol. 6860, pp. 125–139. Springer, Heidelberg (2011)

[8] Ferrarotti, F., Hartmann, S., Link, S.: Efficiency frontiers of XML cardinality constraints. Data Knowl. Eng., http://dx.doi.org/10.1016/j.datak.2012.09.004 (2013)

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

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

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

[12] Hartmann, S., Link, S., Trinh, T.: Constraint acquisition for Entity-Relationship models. Data Knowl. Eng. 68(10), 1128–1155 (2009)

[13] Langeveldt, W.D., Link, S.: Empirical evidence for the usefulness of Armstrong relations in the acquisition of meaningful functional dependencies. Inf. Syst. 35(3), 352–374 (2010)

[14] Le, V., Link, S., Memari, M.: Schema- and data-driven discovery of SQL keys. JCSE 6(3), 193–206 (2012)

[15] Liddle, S.W., Embley, D.W., Woodfield, S.N.: Cardinality constraints in semantic data models. Data Knowl. Eng. 11(3), 235–270 (1993)

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

[17] Link, S.: Armstrong databases: Validation, communication and consolidation of conceptual models with perfect test data. In: Ghose, A., Ferrarotti, F. (eds.) APCCM 2012. pp. 3–20. Australian Computer Society (2012)

[18] Mannila, H., Räihä, K.J.: Design of Relational Databases. Addison-Wesley (1992)

[19] Silva, A., Melkanoff, M.: A method for helping discover the dependencies of a relation. In: Advances in Data Base Theory. pp. 115–133 (1979)

[20] Thalheim, B.: Entity-Relationship modeling. Springer (2000)

[21] Thalheim, B.: Fundamentals of cardinality constraints. In: Pernul, G., Min Tjoa, A. (eds.) ER 1992. LNCS, vol. 645, pp. 7–23. Springer, Heidelberg (1992)

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