SQL-Sampler: A Tool to Visualize and Consolidate Domain Semantics by Perfect SQL Sample Data

Van Tran Bao Le[1] **Sebastian Link**[2] **Flavio Ferrarotti**[1]

[1] School of Information Management,
Victoria University of Wellington, New Zealand
Email: van.t.le@vuw.ac.nz, flavio.ferrarotti@vuw.ac.nz

[2] Department of Computer Science,
University of Auckland, Auckland, New Zealand
Email: s.link@auckland.ac.nz

Abstract

SQL database designs can result from methodologies such as UML or Entity-Relationship modeling, Description Logic specifications, or relational normalization. Independently from the methodology, the use of good sample data is promoted by academia and commercial database design tools to visualize, validate and consolidate the database designs produced. Unfortunately, advice on what constitutes good sample data, or support to create good sample data are hard to come by. Armstrong databases provide a right notion of sample data that perfectly represent the domain semantics encoded in the form of SQL constraints. We present a tool that computes Armstrong sample tables for different classes of SQL constraints, and different interpretations of null markers. Armstrong tables illustrate the perceptions of an SQL database design about the semantics of an application domain. The tool exemplifies the impact of various design choices on Armstrong tables. These include the expressiveness of the classes of SQL constraints considered, and the semantics of null markers. Armstrong tables complement existing database design methodologies. In particular, they provide data samples that guide the transfer from relational approximations of an application domain to an actual real-life SQL table design.

1 Introduction

Classical database design comprises a variety of methodologies, including conceptual approaches with UML or Entity-Relationship modeling, Description Logic specifications or relational normalization. The output of these approaches is usually a database schema within Codd's relational model of data. The ultimate classical goal, however, is to design an SQL database schema. Relational database schemata constitute only approximations of the target SQL database schema. The reason is that SQL provides features not available in the relational model. In SQL tables it is possible that duplicate and partial information can occur. This makes data processing more efficient as duplicate removal is often considered to be too expensive, and the occurrence of null markers provides simple yet efficient means for partial information to enter the database. Due to these features, the interaction of SQL constraints is delicate, difficult to comprehend for database designers, and even more difficult to communicate to other stake-holders of the target database. Since SQL database design is a challenging and essential task, academic and commercial database design tools, e.g. ERWin (CA Technologies 2011), promote the use of good sample data to visualize, validate and consolidate the database designs they produce. Unfortunately, advice on what constitutes good sample data, or support to create good sample data are hard to come by. Armstrong databases provide a right notion of perfect sample data (Beeri et al. 1984, Fagin 1982, Hartmann, Kirchberg & Link 2012, Mannila & Räihä 1986). They constitute single database instances that satisfy the SQL constraints currently perceived semantically meaningful by the team of database designers, and, for a given class of SQL constraints under consideration, violate all those constraints currently perceived meaningless. Hence, Armstrong databases are visualizations of abstract sets of SQL constraints.

2 Motivating Example

We will now examine an example that illustrates how Armstrong databases can be used to transfer a relational approximation of a target database schema into an SQL table definition. The example showcases the benefit in using good sample data to complement current database design methodologies. For this purpose we revisit a classical example, originally used to show that there are Boyce-Codd normal form decompositions that cannot preserve all functional dependencies (FDs) (Beeri & Bernstein 1979). Suppose the design team has obtained the relation schema CONTACT with columns *Address*, *City*, and *ZIP*, and FD set Σ with *Address*, *City* $\rightarrow$ *ZIP* and *ZIP* $\rightarrow$ *City*. This schema is in Third normal form, but not in Boyce-Codd normal form (Beeri & Bernstein 1979). Normalization algorithms stop here, and cannot provide any further guidance on how to implement the relation schema within an SQL table definition. An inspection of an Armstrong relation for Σ such as

Armstrong relation

Address	*City*	*ZIP*
03 Hudson St	Manhattan	10001
70 King St	Manhattan	10001
70 King St	San Francisco	94107
15 Maxwell St	San Francisco	94129

does also not help, as SQL features like duplicate rows and null markers are not featured in relations. We may therefore ask for an Armstrong *table* (Hartmann,

Kirchberg & Link 2012) for the given set Σ of FDs and NOT NULL constraints, say on *Address* and *ZIP*, e.g.,

Armstrong table t_1

Address	*City*	*ZIP*
03 Hudson St	Manhattan	10001
03 Hudson St	Manhattan	10001
70 King St	Manhattan	10001
70 King St	San Francisco	94107
15 Maxwell St	San Francisco	94129
46 State St	ni	60609

An inspection of the table t shows that a specification of FDs does not exclude occurrences of duplicate rows in SQL tables. In fact, Σ does not imply any uniqueness constraints (UCs) over SQL tables (Hartmann, Kirchberg & Link 2012). At this stage, the design team decides that the FD *Address, City* $\rightarrow$ *ZIP* should be replaced by the stronger UC $u(Address, City)$, meaning that there cannot be any different rows with matching total values on both *Address* and *City*. Furthermore, the interpretation of the null marker ni is *no information*, i.e., a value may not exist, or it may exist but is currently unknown. This is the interpretation that SQL uses (Zaniolo 1984). The occurrence of ni in the table above indicates that the column *City* is nullable. An Armstrong table t' for the revised constraint set is

Armstrong table t_2

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

Looking at the last two rows of the table t', the design team notices that the UC $u(Address, ZIP)$ 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. Inspections of further sample data does not reveal any additional meaningful constraints. Thus, the design team finally arrives at the following SQL table implementation

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

where the state assertion Q enforces the FD *ZIP* $\rightarrow$ *City* by

```
SELECT COUNT(*)
FROM Contact c1
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)));
```

on a data or middle tier.

3 Contribution

In this article we showcase a new tool, called *SQL-Sampler*, to generate Armstrong tables for different classes of SQL constraints including

- NOT NULL constraints,
- uniqueness constraints, and
- functional dependencies,

and the following two interpretations of null marker occurrences:

- ni, that is, no information, and
- unk, that is, value unknown at present.

The tool complements existing database design methodologies by creating sample data with provably good properties. The creation of such sample data is promoted by leading database design tools, but has not enjoyed any support yet. Our tool enables the effective visualization, consolidation and communication of database designs produced by currently available design methodologies. In particular, it can be used to transfer relational approximations of a target schema into an SQL implementation.

Organization. We discuss related systems and the novelty of our tool in Section 4. In Section 5 we give the necessary definitions of the SQL table model and the classes of constraints under investigation. An overview of the functionality of *SQL-Sampler* is given in Section 6. The use of the graphical user interface is illustrated by a simple example in Section 7. Details on the implementation of *SQL-Sampler* are given in Section 8. Finally, we conclude in Section 9.

4 Related Systems and Novelty

SQL-Sampler helps design teams identify those SQL constraints that encode the semantics of the given application domain. The help comes in form of Armstrong tables that provide an exact, sample-based representation of the SQL constraints currently perceived meaningful. Design teams can consolidate their current perceptions about the application domain's semantics by inspecting Armstrong tables together with domain experts.

Armstrong databases are user-friendly representation formats of abstract constraints (Beeri et al. 1984, Fagin 1982, Hartmann, Kirchberg & Link 2012, Mannila & Räihä 1986). Several prototypes were developed that compute Armstrong databases for a given set of FDs, and the paradigm *design-by-example* was established (De Marchi et al. 2003, Mannila & Räihä 1986, Silva & Melkanoff 1979). However, all these tools produce relations. These are just the idealized special case of SQL tables where neither duplicate rows nor null values occur. Thus, relations never show what delicate interactions between NOT NULL constraints, UCs, and FDs are possible over SQL tables. Hence, previous tools do not help with SQL table design - and were not intended for this task.

SQL-Sampler is designed to create Armstrong databases for different classes of SQL constraints, and for different interpretations of null marker occurrences within SQL tables. The classes considered include NOT NULL constraints, UCs, and FDs. The interpretations of null marker occurrences include the *no information* (Zaniolo 1984) and the *unk* interpretation (Codd 1979). The choice of a class and an interpretation determines the semantics of SQL tables,

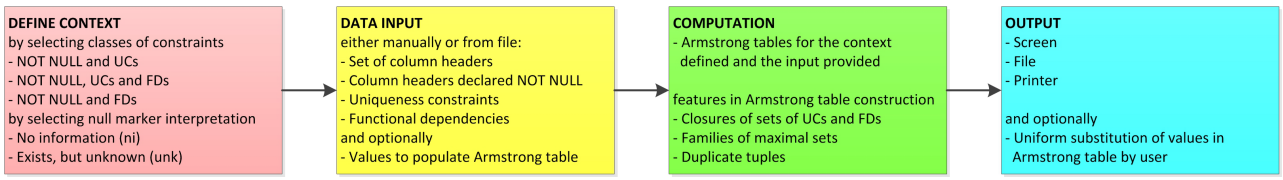

Figure 1: Workflow in *SQL-Sampler*

and thus also the Armstrong tables produced. *SQL-Sampler* is the result of implementing algorithms for different combinations of these SQL constraints, and different interpretations of null marker occurrences, published in our recent work (Ferrarotti et al. 2011, Hartmann, Kirchberg & Link 2012, Le et al. 2012*b*). Our tool is the first implementation of these algorithms. The tool has been exploited in comprehensive experiments that confirmed its usefulness in identifying semantically meaningful constraints that were incorrectly perceived as meaningless before its use (Le et al. 2013). This research has extended usability studies from the relational model of data (Langeveldt & Link 2010).

5 SQL tables and constraints

In this section we define the syntax and semantics for the different classes of constraints under different interpretations of null markers.

Let $\mathfrak{H} = \{H_1, H_2, \ldots\}$ be a countably infinite set of symbols, called *columns*. A *table schema* is a finite non-empty subset T of $\mathfrak{H}$. Each column 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, $\texttt{ni} \in dom(H)$.

For column 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 $r_2(H) = \texttt{ni}$. For example, the row

(03 Hudson St,Manhattan,10001)

subsumes the row

(03 Hudson St,ni,ni).

For a row r over T and a set $X \subseteq T$, r is said to be X-total if for all $H \in X$, $r(H) \neq \texttt{ni}$. 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 columns 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 $r(Y) = r'(Y)$ (Lien 1982). 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 t_1 and t_2 from the introduction satisfy $ZIP \to City$. While the table t_1 violates every SUC, the table t_2 satisfies $u(Address,City)$ but violates $u(Address,ZIP)$.

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 table t_2 from the introduction shows that the set Σ consisting of $ZIP \to City$, $u(Address,City)$, and the NFS $nfs(Address,ZIP)$ does not imply $u(Address,ZIP)$.

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 t_2 from the introduction is Armstrong for the set Σ containing $ZIP \to City$, $u(Address,City)$, and $nfs(Address,ZIP)$. By inspecting table t_2, we know that Σ does not imply $City \to Address$ nor $u(Address,ZIP)$, but does imply $Address,ZIP \to City$ and $u(Address,City)$.

Constraints 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

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

A *Codd functional dependency* (CFD) over table schema T is an expression $\diamond(X \to Y)$ where $X, Y \subseteq T$. A Codd table t over T satisfies $\diamond(X \to Y)$ 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. The use case in Section 7 discusses CUCs and CFDs on the same example from the introduction, thereby illustrating the differences between both semantics.

Algorithms to compute $\mathcal{C}$-Armstrong tables were recently developed for the classes $\mathcal{C}$ of `NOT NULL` constraints and i) SUCs in (Le et al. 2012*b*,*a*), ii) SFDs in (Hartmann, Kirchberg & Link 2012), iii) SUCs and SFDs in (Hartmann, Kirchberg & Link 2012), and iv) CUCs and CFDs in (Ferrarotti et al. 2011). Our tool implements all of these algorithms.

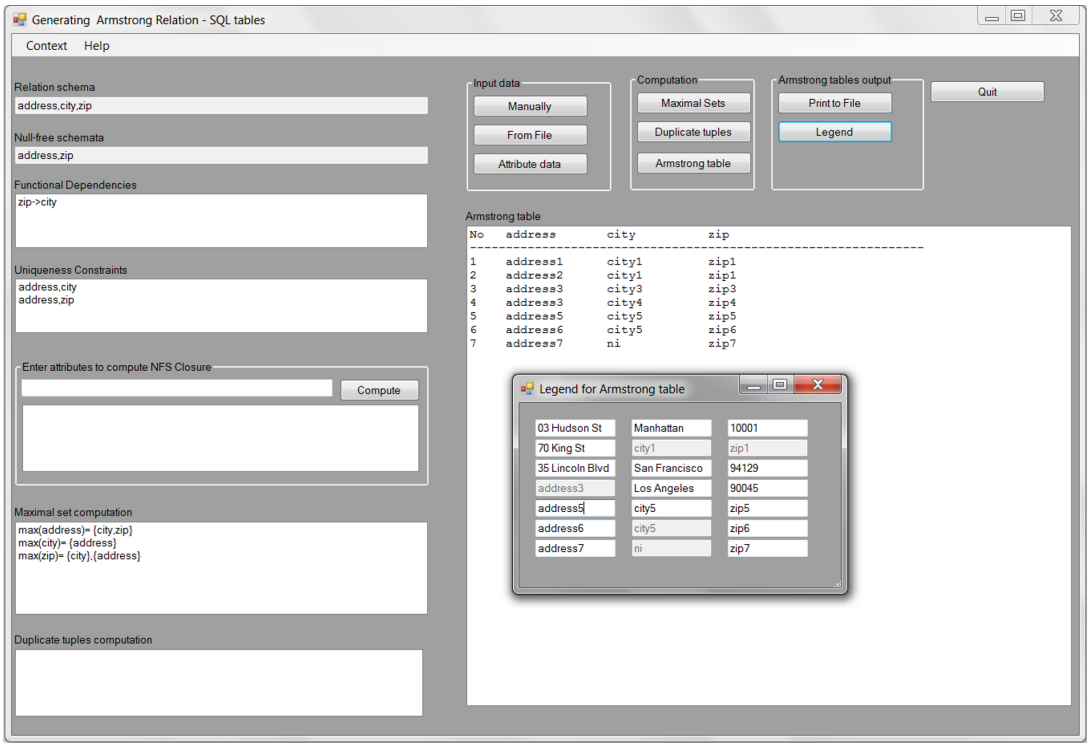

Figure 2: Main Interface of *SQL-Sampler*

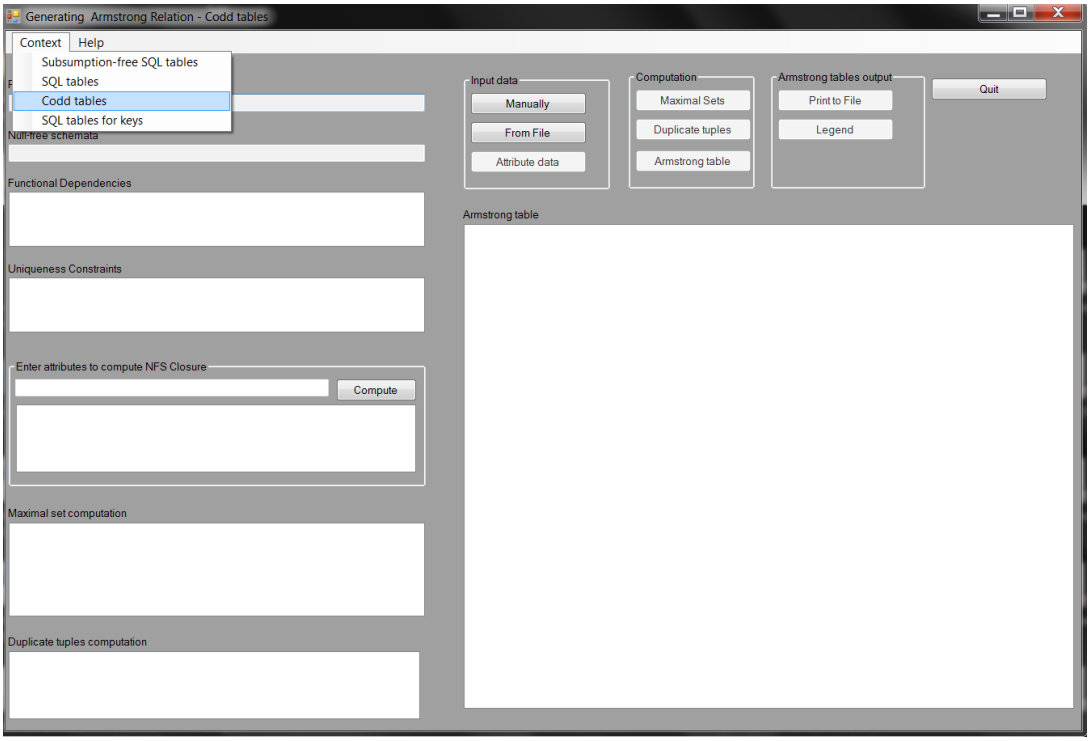

Figure 3: Screenshot of Selecting the Context

6 System Overview

SQL-Sampler was developed in *C#*. The desktop version runs in Windows 7 (64 bit) and can be downloaded at

`armstrongtable.sim.vuw.ac.nz/ArmstrongData.zip`

and the web-based tool is available at

`armstrongtable.sim.vuw.ac.nz`.

Its general workflow is depicted in Figure 1. The graphical user interface (GUI) of *SQL-Sampler* consists of four main modules, as shown in Figure 2.

6.1 Context Module

In the context module, users select the class of constraints they consider for their application domain. This choice also determines the interpretation of null marker occurrences within the Armstrong tables produced by the tool. Possible selections include the context of i) subsumption-free SQL tables where `NOT NULL` constraints and SFDs are considered, ii) SQL tables where `NOT NULL` constraints, SUCs, and SFDs are considered, iii) Codd tables where `NOT NULL` constraints, CUCs, and CFDs are considered, and iv) SQL tables with keys where `NOT NULL` constraints and SUCs are considered. Note that in subsumption-free SQL tables, the class of SFDs subsumes the class of SUCs, but in arbitrary SQL tables the class of SFDs does not subsume the class of SUCs (Hartmann, Kirchberg & Link 2012). For contexts i), ii), and iv), the corresponding interpretation of the null marker is fixed to `ni` for *no information*. That is, a value may not exist or it exists, but is currently unknown. For context iii), the interpretation is fixed to `unk` for value exists, but is currently unknown.

6.2 Input Module

In the input module the user defines a table schema, a set of columns declared `NOT NULL` and a set Σ of UCs and/or FDs. The constraints are specified by a simple selection of the columns involved. Figure 4 shows part of the input module. Users can also open saved inputs from a file, and values for the domains of columns can be defined. These values are then used in the computation module to populate Armstrong tables. If no values are provided by users, generic values will be chosen. Users always have the choice of suitably replacing values in the Armstrong tables by new values. The system guarantees that the replacements always result in Armstrong tables.

6.3 Computation Module

For the computation module several algorithms established in the recent literature (Ferrarotti et al. 2011, Hartmann, Kirchberg & Link 2012, Le et al. 2012*b*) have been implemented to compute Armstrong tables from the context and input specified. Figure 2 shows an Armstrong table produced for our example from the introduction. For users interested in the composition and structure of the Armstrong table, other computational features can be selected. These include the computation of closures of sets of columns, the computation of maximal set families, and the computation of duplicate rows (Hartmann, Kirchberg & Link 2012). In Section 7 we illustrate the definition of maximal and duplicate sets by a detailed example, and explain their instrumental role in computing Armstrong tables.

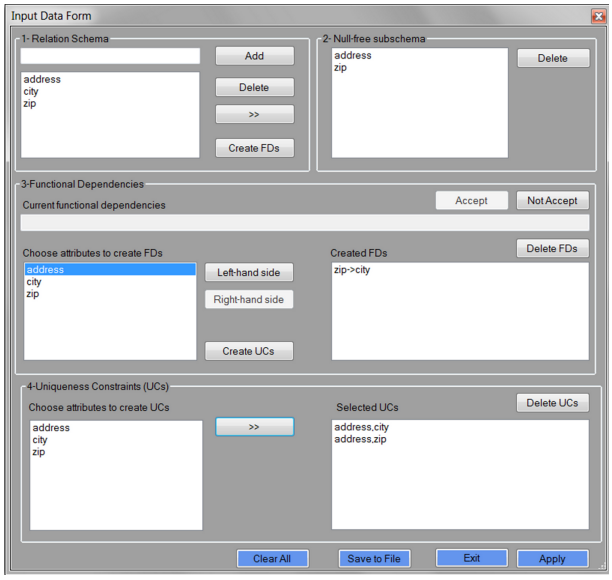

Figure 4: The Input Module of *SQL-Sampler*

6.4 Output Module

The output module allows the user of *SQL-Sampler* to modify, present and save the Armstrong table produced. Figure 2 shows the legend interface where values from the table produced can be replaced manually by the user. The interface guarantees that the tables resulting from these replacements are always Armstrong tables for the context and input specified earlier. Finally, the user has the possibility to save the Armstrong table in a file.

7 Use Case

In this section we briefly illustrate the use of *SQL-Sampler* on a simple example.

As use case we select the relation schema CONTACT which consists of the columns *Address*, *City*, and *ZIP*. The null-free subschema $nfs(\text{CONTACT}_s)$ is defined by $\text{CONTACT}_s = \{ZIP\}$, and as the input set of constraints we select the set

$$\Sigma = \{\diamond u(Address, City), \diamond(ZIP \rightarrow City)\}$$

that consists of a Codd uniqueness constraint and a Codd functional dependency. We illustrate how SQL-Sampler can be used to compute an Armstrong table for Σ and $nfs(\text{CONTACT}_s)$ with respect to Codd uniqueness constraints, Codd functional dependencies and `NOT NULL` constraints.

7.1 Context

The use case description above tells us which context needs to be defined: We select the context *Codd Table*, which means that the interpretation of all null marker occurrences `unk` in the Codd table are fixed to "value unknown at present". The Armstrong table is computed with respect to the combined class of CUCs, CFDs with `NOT NULL` constraints. Figure 3 shows how the context can easily be selected in *SQL-Sampler*.

7.2 Input Data

Figure 5 shows a screenshot of the Input Data module of *SQL-Sampler* after the data from the use case was filled in.

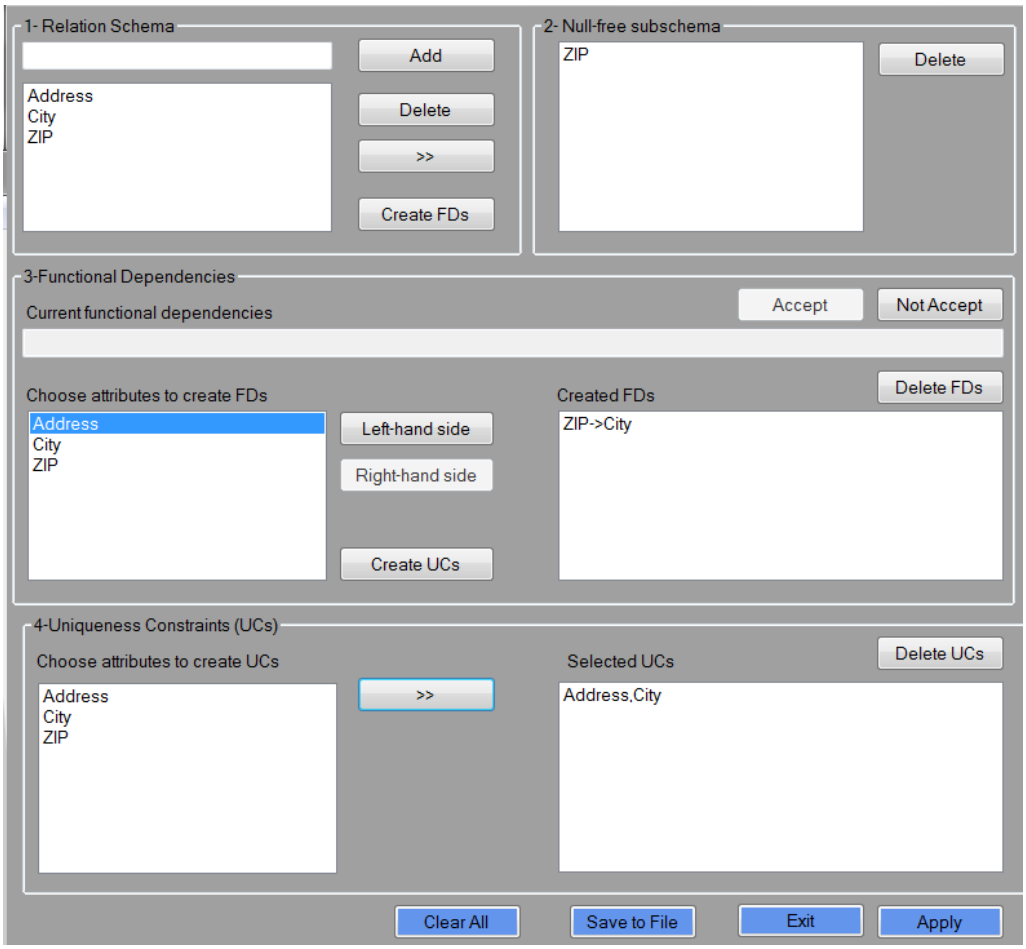

Figure 5: Screenshot of Putting in Data

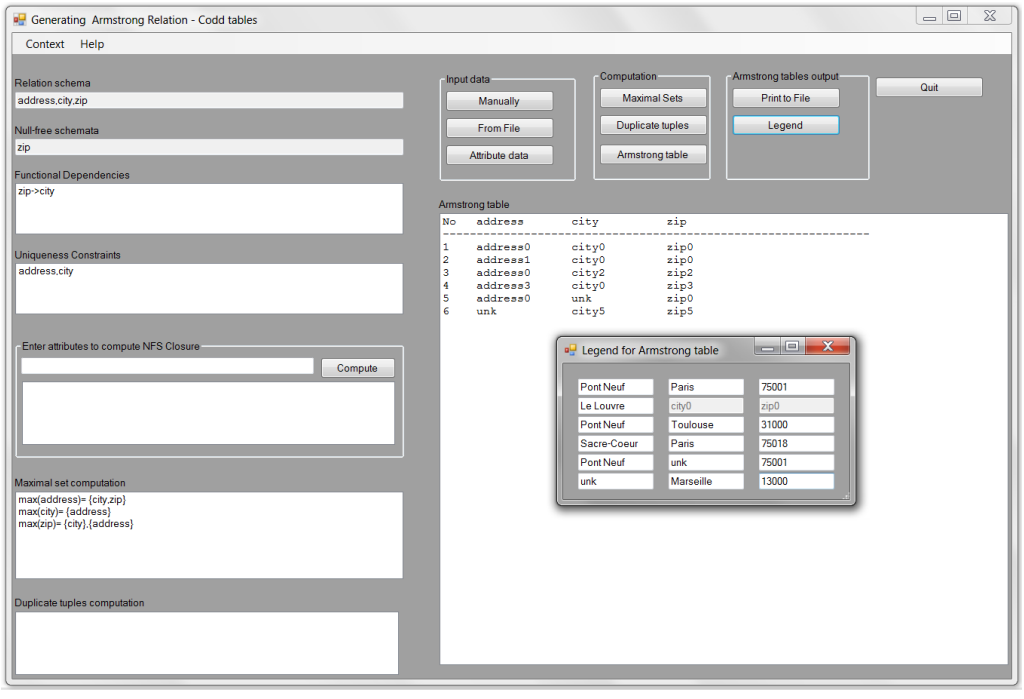

Figure 6: Screenshot of Output Data

7.3 Computing Armstrong Table

Figure 6 contains a screenshot of the Armstrong table computed by *SQL-Sampler* on the basis of the input data. Since no domain values had been supplied by the user, the Armstrong table was populated with generic data values. The screenshot also shows the maximal and duplicate sets computed by *SQL-Sampler*. The definition of maximal and duplicate sets was given in (Ferrarotti et al. 2011, Hartmann, Kirchberg & Link 2012) and is instrumental to the computation of Armstrong tables in all contexts. Here, we illustrate their instrumental role on the use case.

An Armstrong table for a given set Σ of uniqueness constraints and functional dependencies, and a null-free subschema $nfs(T_s)$ must violate all functional dependencies not implied by Σ and $nfs(T_s)$. For every column A, however, it suffices to violate those FDs $X \to A$ where the left-hand side X is maximal, under set inclusion, with the property that $X \to A$ is not implied. Hence, for a column A the set $max_{\Sigma,T_s}(A)$ contains all those sets X that are maximal with the property that $X \to A$ is not implied by Σ and $nfs(T_s)$. The computation of maximal set families from Σ and $nfs(T_s)$ is detailed in (Ferrarotti et al. 2011, Hartmann, Kirchberg & Link 2012).

In our use case where $\text{CONTACT}_s = \{ZIP\}$ and

$$\Sigma = \{\diamond u(Address, City), \diamond(ZIP \to City)\}$$

the set $max_{\Sigma,\text{CONTACT}_s}(ZIP)$ of maximal sets for *ZIP* is $\{\{Address\}, \{City\}\}$, as shown in Figure 6. Indeed, Σ and $nfs(\text{CONTACT}_s)$ do not imply $\diamond(Address \to ZIP)$ and $\diamond(City \to ZIP)$, but they do imply $\diamond(Address, City \to ZIP)$.

When computing an Armstrong table for Σ and $nfs(T_s)$ it is ensured that for each set X that is maximal for some A, there are two different rows in the table that have matching non-null values on all the columns in X and different values on A. This ensures that the Armstrong table violates the FD $X \to A$, and thereby also every FD $X' \to A$ where $X' \subseteq X$ holds. The exact construction of the Armstrong table depends on the context and is detailed in (Ferrarotti et al. 2011, Hartmann, Kirchberg & Link 2012).

In our use case, for example, the maximal set $\{Address\}$ for the column *ZIP* is represented by the first and third row in the Armstrong table, while the maximal set $\{City\}$ for the column *ZIP* is represented by the first and fourth row in the Armstrong table, as shown in Figure 6.

An Armstrong table for a given set Σ of uniqueness constraints and functional dependencies, and a null-free subschema $nfs(T_s)$ must also violate all uniqueness constraints $u(X)$ not implied by Σ and $nfs(T_s)$.

If for some column A, the FD $X \to A$ is not implied by Σ and $nfs(T_s)$, then X is the subset of some set that is maximal for A. Hence, the construction of the Armstrong table - as described above - ensures that the uniqueness constraint $u(X)$ is violated by the Armstrong table. In our use case, for example, the uniqueness constraint $\diamond u(City, ZIP)$ is not implied by Σ and $nfs(\text{CONTACT}_s)$, but also not the CFD $\diamond(City,ZIP \to Address)$. Indeed, the set $\{City,ZIP\}$ is maximal for *Address*, and the FD is violated by the first and second row in the Armstrong table, as shown in Figure 6.

Otherwise, for every column $A \in T$ the FD $X \to A$ is implied by Σ and $nfs(T_s)$, that is $X \to T$ is implied. In that case, which cannot occur over pure relations with no duplicate rows, we still need to violate the uniqueness constraints $u(X)$ that is not implied by Σ and $nfs(T_s)$. Therefore the set $dup_{\Sigma,T_s}(T)$ of duplicate rows contains those column subsets $X \subseteq T$ which are maximal with the property that $u(X)$ is not implied by Σ and $nfs(T_s)$ but $X \to T$ is implied by Σ and $nfs(T_s)$. In our use case, for example, the CUC $\diamond u(Address, ZIP)$ is not implied by Σ and $nfs(\text{CONTACT}_s)$, but the CFD $\diamond(Address, ZIP \to City)$ is implied by Σ and $nfs(\text{CONTACT}_s)$. Consequently, $\{Address,ZIP\}$ is a duplicate set.

When computing an Armstrong table for Σ and $nfs(T_s)$ it is ensured that for each duplicate set X, there are two different rows in the table that have matching non-null values on all the columns in X, and either one (for Codd constraints) or both rows (for SQL constraints) carry null marker occurrences on every other column outside of X. Note that this is well-defined as every duplicate set X contains all the columns of the null-free subschema $nfs(T_s)$, and therefore, every column outside of X can carry null marker occurrences. The construction ensures that the Armstrong table violates the UC $u(X)$ and satisfies the FD $X \to T$. The exact construction of the Armstrong table depends on the context and is detailed in (Ferrarotti et al. 2011, Hartmann, Kirchberg & Link 2012). In our use case, for example, the duplicate set $\{Address,ZIP\}$ is represented by the first and fifth row, where the fifth row is `unk` on column *City*.

7.4 Output Data

Figure 6 shows a screenshot of how the generic values in an Armstrong table can be replaced by real data values. This can be done by activating the "Legend" button in the "Armstrong table output" menu. The user can then manually enter the real data values that replace the generic ones.

8 Some Implementation Details

Apart from computing Armstrong tables for different classes of constraints and different interpretations of null markers, *SQL-Sampler* should allow users to re-use and modify previous data. This ensures the effective use of the tool in the requirements acquisition process. For that reason we implemented *SQL-Sampler* in C# which is a powerful programming language to build a graphical user interface application. We have stored the data in an SQL Server database, consisting of nine main tables, and provided also a web application that database designers can access from everywhere without any installation concerns. To develop the web-based application we have re-implemented the algorithms in ASP.NET. We chose ASP.NET due to its properties which allow multiple users to share the same requested data for resources concurrently. ASP.NET utilizes the C# syntax, which enabled us to re-use the code already developed for the desktop version. The web-based application is hosted under the domain of `.sim.vuw.ac.nz`.

Users can utilize *SQL-Sampler* as a desktop application or as a web application. For the desktop version, the 32-bit Windows operation system and the .NET framework 3.5 are necessary for *SQL-Sampler* to operate. For the web application, a system with a common Internet browser such as Firefox, Internet Explorer, or Google Chrome are required at the client side. At the server-side, *SQL-Sampler* requires Internet Information Services (IIS) and the ASP.NET platform 4.0 and SQL Server 2005/2008 to operate *SQL-Sampler*.

The algorithms have been implemented as a library of C# objects to handle four different types of Armstrong tables. For each type, relevant components have been coded to handle the sets of columns for the subsequent table, and `NOT NULL` columns, UCs, and FDs as entries in an SQL database. This enables the Web-based system to efficiently and securely perform operations on the different sets of data. It is stressed that the implementation includes authentication and authorization mechanisms, as well as access abilities for multiple users. The conceptual diagram of the SQL database is shown in Figure 7.

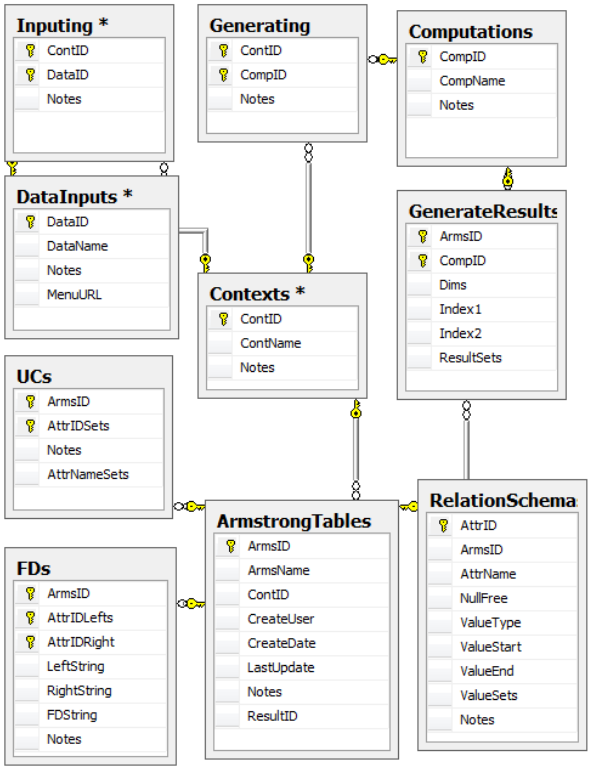

Figure 7: SQL Server database for *SQL-Sampler*

We explain the function of each table in the database next.

ArmstrongTables(ArmsID, ArmsName, ContID, CreateUser, CreateDate, LastUpdate, Notes): This table contains the name, ID, and user-related information for each Armstrong table. Most information in this table is automatically updated except for the name of the Armstrong table which is provided by the user.

RelationSchemata(AttrID, ArmsID, AttrName, NullFree, ValueType, ValueStart, ValueEnd, ValueSets, Notes): It stores properties of columns including their name, domain type, null-free value, start value and end value which define a range of automatic data values to populate an Armstrong table. If users specify their own domain values, the values will be stored in the *ValueSets* column.

FDs(ArmsID, AttrIDLefts, AttRight, LeftString, RightString, FDString, Notes): It contains the functional dependencies the user specifies during the input data stage.

UCs(ArmsID, AttrIDSets, AttrNameSets, Notes): It contains the uniqueness constraints the user specifies during the input data stage.

DataInputs(DataID, DataName, Notes, MenuURL): It contains the categories of input data which *SQL-Sampler* assembles to generate each type of Armstrong table. Currently, this table consists of four rows to encode names of columns, functional dependencies, uniqueness constraints, and domain values as input data categories. Data in this table cannot be updated by users.

Contexts(ContID, ContName, Notes): It stores the different contexts in which Armstrong tables can be computed by *SQL-Sampler*. Currently, this tables contains four rows to encode subsumption-free SQL tables, SQL tables, Codd tables, and SQL tables for keys, as previously described. Data in this table cannot be updated by users.

Inputting(ContID, DataID, Notes): This table specifies the inputs for each context in which Armstrong tables can be computed. For example, this table has four rows to specify the names of columns, null-free subschema, functional dependencies, and uniqueness constraints as input for the context *SQL table*; and it has three rows to specify the names of columns, null-free subschema, and functional dependencies with nulls as input for the context *Subsumption-free SQL table*. Data in this table cannot be updated by users.

Computations(CompID,CompName,Notes): It contains the possible types of outputs computed by *SQL-Sampler*. Currently, this tables consists of fives rows to encode Armstrong tables, closures of column subsets, maximal sets, duplicate sets, and anti-keys as possible output categories for *SQL-Sampler*. Data in this table cannot be updated by users.

Generating(ContID, CompID, Notes): This table specifies the types of outputs available in each context of *SQL-Sampler*. In the context *Codd table*, for example, output is available as Armstrong tables, column set closures, maximal, and duplicate sets. Data in this table cannot be updated by users.

9 Conclusion and Future Work

Humans learn a lot from good examples. *SQL-Sampler* creates Armstrong sample data that visualizes perfectly the delicate interactions of SQL constraints. It can thus be used by design teams to communicate and consolidate their perceptions of an application domain with different stake-holders of the target database. Leading database design tools (e.g. *ERWin*) advertise the use of sample data to validate database schemata produced by existing database design methodologies, for examples, ER modeling and relational normalization. *SQL-Sampler* produces sample data with provably perfect properties, complementing existing methodologies. The inspection of its Armstrong tables leads to the recognition of many meaningful uniqueness constraints and functional dependencies (Le et al. 2013). It thus helps design teams consolidate real-world SQL table designs, and not just relational approximations of application domains.

In future work *SQL-Sampler* can be enhanced to handle more classes of SQL constraints such as other types of key constraints (Hartmann et al. 2011, Thalheim 1989), cardinality constraints (Hartmann, Köhler, Link & Thalheim 2012, Liddle et al. 1993), referential constraints (Fagin & Vardi 1983), or multivalued dependencies (Fagin 1977, Hartmann & Link 2012, 2006, Link 2012, 2008). It is also desirable to study integrity constraints under different representations of incomplete information, or in data models such as XML (Buneman et al. 2002, Ferrarotti et al. 2013, Hartmann & Link 2009, Vincent et al. 2004), RDF (Lausen et al. 2008, Paredaens 2012), or probabilistic models (Demetrovics et al. 1998, Link 2013*b*,*a*, Suciu et al. 2011).

10 Acknowledgement

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

References

Beeri, C. & Bernstein, P. (1979), 'Computational problems related to the design of normal form relational schemas', *ACM Trans. Database Syst.* **4**(1), 30–59.

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

Buneman, P., Davidson, S., Fan, W., Hara, C. & Tan, W. (2002), 'Keys for XML', *Computer Networks* **39**(5), 473–487.

CA Technologies (2011), 'ERwin Data Modeler - methods guide', `https://support.ca.com/cadocs/0/e002961e.pdf`, page 86.

Codd, E. F. (1979), 'Extending the database relational model to capture more meaning', *ACM Trans. Database Syst.* **4**(4), 397–434.

De Marchi, F., Lopes, S., Petit, J.-M. & Toumani, F. (2003), 'Analysis of existing databases at the logical level: the DBA companion project', *SIGMOD Record* **32**(1), 47–52.

Demetrovics, J., Katona, G., Miklos, D., Seleznjev, O. & Thalheim, B. (1998), 'Asymptotic properties of keys and functional dependencies in random databases', *Theor. Comput. Sci.* **190**(2), 151–166.

Fagin, R. (1977), 'Multivalued dependencies and a new normal form for relational databases', *ACM Trans. Database Syst.* **2**(3), 262–278.

Fagin, R. (1982), Armstrong databases, Technical Report RJ3440(40926), IBM Research Laboratory, San Jose, California, USA.

Fagin, R. & Vardi, M. (1983), 'Armstrong databases for functional and inclusion dependencies', *Inf. Process. Lett.* **16**(1), 13–19.

Ferrarotti, F., Hartmann, S., Le, V. & Link, S. (2011), Codd table representations under weak possible world semantics, *in* 'DEXA (1)', Vol. 6860 of *Lecture Notes in Computer Science*, Springer, pp. 125–139.

Ferrarotti, F., Hartmann, S. & Link, S. (2013), 'Efficiency frontiers of XML cardinality constraints', *Data Knowl. Eng.* **87**, 297–319.

Hartmann, S., Kirchberg, M. & Link, S. (2012), 'Design by example for SQL table definitions with functional dependencies', *The VLDB Journal* **21**(1), 121–144.

Hartmann, S., Köhler, H., Link, S. & Thalheim, B. (2012), Armstrong databases and reasoning for functional dependencies and cardinality constraints over partial bags, *in* 'FoIKS', Vol. 7153 of *Lecture Notes in Computer Science*, Springer, pp. 164–183.

Hartmann, S., Leck, U. & Link, S. (2011), 'On Codd families of keys over incomplete relations', *Comput. J.* **54**(7), 1166–1180.

Hartmann, S. & Link, S. (2006), 'On a problem of Fagin concerning multivalued dependencies in relational databases', *Theor. Comput. Sci.* **353**(1-3), 53–62.

Hartmann, S. & Link, S. (2009), 'Efficient reasoning about a robust XML key fragment', *ACM Trans. Database Syst.* **34**(2).

Hartmann, S. & Link, S. (2012), 'The implication problem of data dependencies over sql table definitions: Axiomatic, algorithmic and logical characterizations', *ACM Trans. Database Syst.* **37**(2), 13.

Langeveldt, W. & Link, S. (2010), 'Empirical evidence for the usefulness of Armstrong relations on the acquisition of meaningful functional dependencies', *Inf. Syst.* **35**(3), 352–374.

Lausen, G., Meier, M. & Schmidt, M. (2008), SPARQLing constraints for RDF, *in* 'EDBT', Vol. 261 of *ACM International Conference Proceeding Series*, pp. 499–509.

Le, V. B. T., Link, S. & Memari, M. (2012*a*), 'Schema- and data-driven discovery of SQL keys', *JCSE* **6**(3), 193–206.

Le, V., Link, S. & Ferrarotti, F. (2013), Effective recognition and visualization of semantic requirements by perfect SQL samples, *in* 'ER', Vol. 8217 of *Lecture Notes in Computer Science*, Springer, pp. 227–240.

Le, V., Link, S. & Memari, M. (2012*b*), Discovery of keys from SQL tables, *in* 'DASFAA (1)', Vol. 7238 of *Lecture Notes in Computer Science*, Springer, pp. 48–62.

Liddle, S. W., Embley, D. W. & Woodfield, S. N. (1993), 'Cardinality constraints in semantic data models', *Data Knowl. Eng.* **11**(3), 235–270.

Lien, E. (1982), 'On the equivalence of database models', *J. ACM* **29**(2), 333–362.

Link, S. (2008), 'Charting the completeness frontier of inference systems for multivalued dependencies', *Acta Inf.* **45**(7-8), 565–591.

Link, S. (2012), 'Characterisations of multivalued dependency implication over undetermined universes', *J. Comput. Syst. Sci.* **78**(4), 1026–1044.

Link, S. (2013*a*), Reasoning about saturated conditional independence under uncertainty: Axioms, algorithms, and Levesque's situations to the rescue, *in* 'AAAI', AAAI Press.

Link, S. (2013*b*), 'Sound approximate reasoning about saturated conditional probabilistic independence under controlled uncertainty', *J. Applied Logic* **11**(3), 309–327.

Mannila, H. & Räihä, K.-J. (1986), 'Design by example: An application of Armstrong relations', *J. Comput. Syst. Sci.* **33**(2), 126–141.

Paredaens, J. (2012), What about constraints in RDF?, *in* 'Conceptual Modelling and Its Theoretical Foundations', Vol. 7260 of *Lecture Notes in Computer Science*, Springer, pp. 7–18.

Silva, A. & Melkanoff, M. (1979), A method for helping discover the dependencies of a relation, *in* 'Advances in Data Base Theory', pp. 115–133.

Suciu, D., Olteanu, D., Ré, C. & Koch, C. (2011), *Probabilistic Databases*, Synthesis Lectures on Data Management, Morgan & Claypool Publishers.

Thalheim, B. (1989), 'On semantic issues connected with keys in relational databases permitting null values', *Elektron. Informationsverarb. und Kybern.* **25**(1-2), 11–20.

Vincent, M., Liu, J. & Liu, C. (2004), 'Strong functional dependencies and their application to normal forms in XML', *ACM Trans. Database Syst.* **29**(3), 445–462.

Zaniolo, C. (1984), 'Database relations with null values', *J. Comput. Syst. Sci.* **28**(1), 142–166.