Modelling Entity Integrity for Semi-Structured Big Data

Ilya Litvinenko, Ziheng Wei, and Sebastian Link[0000−0002−1816−2863]

The University of Auckland, Auckland, New Zealand
{ilit874,z.wei,s.link}@auckland.ac.nz

Abstract. We propose a data model for investigating constraints that enforce the entity integrity of semi-structured big data. Particular support is given for the volume, variety, and veracity dimensions of big data.

Keywords: Big data· Functional Dependency· JSON· Key· SQL.

1 Introduction

Database management systems model some domain of the real-world within a database system. For that purpose, SQL governs data by the rigid structure of relations [5]. Big data must handle potentially large volumes of data that may originate from heterogeneous sources (variety) with different degrees of uncertainty (veracity) [1]. Given the mature and popular technology that SQL provides many organizations use SQL to manage big data, at least when it is semi-structured such as in JSON format. While unstructured data, such as text or images, is not our focus, there is a rich landscape of techniques and tools for converting unstructured into semi-structured or even structured data [1].

We introduce the class of keys and functional dependencies (FDs) over possibilistic SQL data, with the aim to efficiently reason about the entity integrity of semi-structured big data that accommodates the volume, variety and veracity dimensions. Codd stipulated entity integrity as one of the three major integrity principles in databases [5]. Entity integrity refers to the principle of representing each entity of the application domain uniquely within the database. Violations of this principle are common in database practice, resulting in their own fields of research including entity resolution [4] and data cleaning [9].

While keys and FDs have standard definitions in the relational model, simple extensions introduce opportunities to define these concepts differently [18]. SQL, for example, permits occurrences of a so-called *null marker*, denoted by $\perp$, to say that there is no information about the value of this row on this column [10, 21]. Moreover, columns can be defined as `NOT NULL` to prevent null marker occurrences. The interpretation of $\perp$ is deliberately kept simple to uniformly accommodate many types of missing information, including values that do not exist or values that exist but are currently unknown [21]. While such distinction is possible, it would lead to an application logic that is too complex for database practice [6]. In modern applications, for example data integration, null

Fig. 1: JSON data from different information sources

markers are used frequently by SQL to fit data of heterogeneous structure within a uniform table. This is SQL's answer to the variety dimension of big data. SQL also permits the duplication of rows in support of a multiset semantics, where FDs can no longer express keys [12]. Hence, for SQL we need to study the combined class of keys and FDs. The veracity dimension abandons the view that all data are equal to improve the outcomes of data-driven decision making. Probabilistic and possibilistic databases offer complementary approaches to uncertain data. Essentially, there is a trade-off as probabilistic databases offer continuous degrees of uncertainty and real probability distributions are hard to come by and maintain, while possibilistic databases offer discrete degrees of uncertainty and are simpler to come by and maintain [8, 17].

Contributions and Organization. We introduce our running example in Section 2. We propose a framework of data structures capable of handling all combinations of the volume, variety, and veracity dimension of semi-structured big data within possibilistic SQL. Section 3 reviews isolated previous work under this framework. We define our possibilistic SQL model in Section 4, and possibilistic SQL constraints in Section 5. We conclude in Section 6.

2 The Running Example

As a simple running example consider Figure 1 that shows some JSON data. JSON is the defacto standard for managing and exchanging semi-structured data, due to its capability to accommodate different information structures [16].

Table 1: A University Employment Table

row	*emp*	*dpt*	*mng*	p-degree	interpretation	origin
1	$\perp$	Math	Simon	α_1	fully possible	payroll
2	$\perp$	Stats	Simon	α_1	fully possible	payroll
3	Mike	CS	Shaun	α_1	fully possible	payroll
4	Tom	CS	Shaun	α_1	fully possible	payroll
5	Derek	Physics	$\perp$	α_1	fully possible	payroll
6	John	Music	Scott	α_1	fully possible	payroll
7	John	Music	Scott	α_2	quite possible	website
8	Andy	$\perp$	Sofia	α_2	quite possible	website
9	Andy	$\perp$	Sam	α_2	quite possible	website
10	Bob	Biology	Susan	α_3	somewhat possible	blog
11	Bob	Arts	Susan	α_3	somewhat possible	blog

In our example, the information origins from three sources: payroll data, web data, and blog data. As the data stewards associate different levels of trust with these sources, they would like to attribute these levels of trust to the data from the sources. We are using an SQL-based DBMS, and the data stewards have transformed the JSON data into SQL-compliant format as shown in Table 1.

The null marker $\perp$ indicates that no information is available for a data element of a given attribute, such as information on employees working in Maths. The column *p-degree* represents the levels of trust for the data elements. The highest p-degree α_1 is assigned to payroll data, α_2 to web data, and α_3 to blog data. This application scenario will be used to illustrate concepts.

3 Related Work

As a first main contribution, we introduce a systematic framework for handling the volume, variety, and veracity dimension of big data. We apply the framework to manage and reason about entity integrity in those dimensions. Figure 2 shows all combinations of the three dimensions, ordered as a lattice. A directed edge means that the target node covers additional dimensions over the source node. For each combination of dimensions, we indicate a data structure for the combination. At the bottom are relations (sets of rows). Arguably, these may already accommodate the volume dimension. The nodes on top of relations are bags, partial relations, and p-relations. Bags accommodate the volume dimension by permitting duplicate rows, partial relations accommodate the variety dimension by permitting null markers (as explained earlier), and p-relations accommodate the veracity dimension. Above these single dimensions we then have any combinations of two dimensions, and the top node combines all three dimensions.

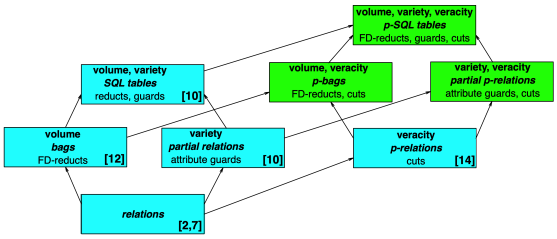

Fig. 2: Framework for Semi-structured Big Data, Related work, and New Contributions

We now use this framework to discuss previous work on entity integrity rules, based on the three dimensions of semi-structured big data. These are marked in cyan in Figure 2. Well-known are Armstrong's axioms for FDs in the relational model [2] and the corresponding algorithms for deciding implication [7]. For bags, the interaction of keys and FDs was characterized in [12], using the technique of FD-reducts. Over partial relations, the implication problem for keys and FDs in the presence of `NOT NULL` constraints was solved in [10], using so-called *attribute guards*. The implication problem over SQL tables, which combine bags and partial relations, was solved by [10]. Finally, the implication problem of FDs over p-relations was solved in [14], using so-called β-cuts.

We can view these isolated results under our big data framework. This view motivates us to extend the previous techniques to new combinations of these dimensions. These include the combination of i) volume and veracity, ii) variety and veracity, and iii) volume, variety, and veracity (marked green in Figure 2).

4 Possibilistic SQL Tables

An *SQL table schema*, denoted by T, is a finite non-empty set of attributes. The domain of each attribute contains the null marker, $\perp$, as a distinguished element. As running example, we use the table schema WORK=$\{emp, dpt, mng\}$ with information about employees that work in departments under managers.

A *row* (or *tuple*) over table schema T is a function $r : T \to \cup_{A \in T} dom(A)$ assigning to every attribute a value from the attribute's domain. The image $r(A)$ of a row r on attribute A is the *value* of r on A. For $X \subseteq T$, a row r over T is X-*total*, if $r(A) \neq \perp$ for all $A \in X$. A row r is *total*, if it is T-total.

We adopt Zaniolo's interpretation of $\perp$ as "no information" [21]. That is, $r(A) = \perp$ means no information about the value of row r on attribute A is available. It may mean there is no value at all, or that there is a value which is currently unknown. SQL uses this interpretation [10, 21].

Our data model handles duplicates as tables are multisets of rows. An *SQL table* over table schema T is a finite multiset t of rows over T. For $X \subseteq T$, the table t is X-total, if every row $r \in t$ is X-total. Table t is total, if t is T-total. A total table is called a bag. Table 2 shows an example of a table over WORK. The third row has value *Derek* on *emp*, value *Physics* on *dpt*, and marker $\perp$ on *mng*. The first and second row are total, and the third and fourth row, as well as the table itself are $\{emp, dpt\}$-total. The first and second row, as well as the third and fourth row, respectively, are duplicate tuples, since they have matching values on all attributes.

Table 2: Table over WORK

emp	*dpt*	*mng*
John	Music	Scott
John	Music	Scott
Derek	Physics	$\perp$
Derek	Physics	$\perp$

SQL does not accommodate uncertainty. For example, one cannot say tuple (Derek, Physics, $\perp$) is less likely to occur than tuple (John, Music, Scott). We extend our data model by assigning degrees of possibilities (p-degrees) to tuples, thereby also extending the model of [14, 15] where no duplicate nor partial information was used. In our example, p-degrees result from the source the tuples originate from. The tuples in Table 1 originate from payroll, website or blog data, respectively. Payroll data is 'fully possible', website data 'quite possible', and blog data 'somewhat possible', while other tuples are 'impossible' to occur in the current table. Since p-degrees can have different mean-

Table 3: Possible Worlds of p-SQL table from Table 1

t_1

emp	*dpt*	*mng*
$\perp$	Math	Simon
$\perp$	Stats	Simon
Mike	CS	Shaun
Tom	CS	Shaun
Derek	Physics	$\perp$
John	Music	Scott

t_2

emp	*dpt*	*mng*
$\perp$	Math	Simon
$\perp$	Stats	Simon
Mike	CS	Shaun
Tom	CS	Shaun
Derek	Physics	$\perp$
John	Music	Scott
John	Music	Scott
Andy	$\perp$	Sofia
Andy	$\perp$	Sam

t_3

emp	*dpt*	*mng*
$\perp$	Math	Simon
$\perp$	Stats	Simon
Mike	CS	Shaun
Tom	CS	Shaun
Derek	Physics	$\perp$
John	Music	Scott
John	Music	Scott
Andy	$\perp$	Sofia
Andy	$\perp$	Sam
Bob	Biology	Susan
Bob	Arts	Susan

ings, we denote them by abstract symbols $\alpha_1, \ldots, \alpha_k, \alpha_{k+1}$. Table 1 shows an instance with p-degrees assigned to tuples. The table has meta-data columns: 'row' assigns an identifier to each tuple, while 'interpretation' and 'origin' show the interpretation of p-degrees and the source of tuples, respectively.

A *possibility scale* is a strict finite linear order $\mathcal{S}_p = (S_p, >_p)$, denoted by $\alpha_1 >_p \cdots > \alpha_k >_p \alpha_{k+1}$, where k is at least one. The elements α_i are *possibility degrees* (p-degrees). In Table 1 we have $k = 3$ for the *possibility scale*. Fully possible rows have p-degree α_1, while the least possible rows have p-degree α_3. The bottom p-degree $\alpha_{k+1} = \alpha_4$ captures rows 'impossible' for the current table. Non-possibilistic tables are a special case of possibilistic ones where $k = 1$.

A *possibilistic SQL table schema* (or p-SQL table schema) is a pair $(T, \mathcal{S}_p)$, where T is a table schema and $\mathcal{S}_p$ is a possibility scale. A *possibilistic SQL table* (or p-SQL table) over $(T, \mathcal{S}_p)$ consists of a table t over T, and a function $Poss_t$ that maps each row $r \in t$ to a p-degree $Poss_t(r) \neq \alpha_{k+1}$ in the p-scale $\mathcal{S}_p$. The p-SQL table of our example is shown in Table 1. It consists of an SQL table over WORK in which every row is assigned a p-degree from α_1, α_2 or α_3. P-SQL tables enjoy a well-founded possible world semantics. The possible worlds form a linear chain of k SQL tables in which the i-th possible world contains tuples with p-degree α_i or higher. Given a p-SQL table t over $(T, \mathcal{S}_p)$, the possible world t_i associated with t is defined by $t_i = \{r \in t \mid Poss_t(r) \geq \alpha_i\}$, that is, t_i is an SQL table of those rows in t that have p-degree α_i or higher. Since t_{k+1} would contain impossible tuples it is not considered a possible world. Table 3 shows the possible worlds of the p-SQL table from Table 1. The possible worlds of t form a linear chain $t_1 \subseteq t_2 \subseteq t_3$.

The linear order of the p-degrees $\alpha_1 > \cdots > \alpha_k$ results in a reversed linear order of possible worlds associated with a p-SQL table t: $t_1 \subseteq \cdots \subseteq t_k$. We point out the distinguished role of the top p-degree α_1. Every row that is fully possible belongs to every possible world. Therefore, every fully possible row is also fully certain. This explains why p-SQL tables subsume SQL tables as a special case.

5 Possibilistic SQL Constraints

We recall the definitions of SQL FDs and `NOT NULL` constraints [10]. Keys are essential to entity integrity and cannot be expressed by FDs in this context.

Intuitively, a key is an attribute collection that can separate different rows by their values on the key attributes. We adopt the semantics for the SQL constraint `UNIQUE` by separating different rows whenever they are total on the key attributes. A *key* over an SQL table schema T is an expression $u(X)$ where $X \subseteq T$. An SQL table t over T *satisfies* $u(X)$ over T, denoted by $\models_t u(X)$, if for all $r_1, r_2 \in t$ we have: if $r_1(X) = r_2(X)$ and r_1, r_2 are X-total, then $r_1 = r_2$. The possible world t_1 of Table 3 satisfies $u(emp)$, while t_2 and t_3 violate this key.

The following semantics of FDs goes back to Lien [13]. A *functional dependency* (FD) over an SQL table schema T is an expression $X \to Y$ where $XY \subseteq T$. An SQL table t over T *satisfies* $X \to Y$ over T, denoted by $\models_t X \to Y$, if for all $r_1, r_2 \in t$ we have: if $r_1(X) = r_2(X)$ and r_1, r_2 are X-total, then $r_1(Y) = r_2(Y)$.

The possible world t_2 of Table 3 satisfies *emp* $\rightarrow$ *dpt* and *dpt* $\rightarrow$ *mng*, while t_3 satisfies *dpt* $\rightarrow$ *mng*, but not *emp* $\rightarrow$ *dpt*.

SQL `NOT NULL` constraints control occurrences of the null marker. They have been studied in combination with FDs and multivalued dependencies [10]. A `NOT NULL` constraint over an SQL table schema T is an expression $n(X)$ where $X \subseteq T$. An SQL table t over T *satisfies* the `NOT NULL` constraint $n(X)$ over T, denoted by $\models_t n(X)$, if t is X-total. For a given set Σ of constraints over T we call $T_s = \{A \in T \mid \exists n(X) \in \Sigma \wedge A \in X\}$ the *null-free subschema* (NFS) over T. If $T_s = T$, we call T a *bag schema*, as instances over T are bags. For example, $n(dpt)$ is satisfied by the possible world t_1 in Table 3, but not by t_2 or t_3.

Possibilistic SQL Constraints. We extend our semantics of SQL constraints to possibilistic SQL tables. Following [14], we use the p-degrees of rows to specify with which certainty an SQL constraint holds. Similar to how α_i denotes p-degrees of rows, β_i denotes c-degrees by which constraints hold. Let us inspect some SQL constraints on the possible worlds t_1, t_2, t_3 in Table 3. The constraint *dpt* $\rightarrow$ *mng* is satisfied by t_3, and therefore by t_2 and t_1. Since the constraint is satisfied by every possible world, it is 'fully certain' to hold, denoted by β_1. The constraint *emp* $\rightarrow$ *dpt* is satisfied by t_2 and therefore by t_1, but it is not satisfied by t_3. Since the constraint is only violated by the 'somewhat possible' world t_3, it is 'quite certain' to hold, denoted by β_2. The constraint *u(emp)* is satisfied by t_1, but it is not satisfied by t_2 and therefore not by t_3. Since the smallest possible world that violates the constraint is 'quite possible', it is 'somewhat certain' to hold, denoted by β_3. The constraint *n(emp)* is not even satisfied in the 'fully possible' world t_1. It is 'not certain at all' to hold, denoted by β_4.

The examples illustrate how the p-degrees of rows motivate degrees of certainty (c-degrees) with which constraints hold on p-SQL tables. If the smallest world that violates a constraint has p-degree α_i (this world is impossible only when all possible worlds satisfy the constraint), then the constraint holds with c-degree β_{k+2-i}. For example, the p-key *u(emp)* holds with c-degree β_3 in the p-SQL table t of Table 1, meaning the smallest possible world that violates *u(emp)* is t_2, which is 'quite possible', that is *u(emp)* is 'somewhat certain' to hold in t. We introduce the certainty scale derived from a given possibility scale.

Let $(T, \mathcal{S}_p)$ denote a p-SQL table schema where the bottom p-degree of $\mathcal{S}_p$ is $k+1$. The certainty scale $\mathcal{S}_p^T$ for $(T, \mathcal{S}_p)$ is the strict finite linear order $\beta_1 >_p \cdots >_p \beta_k >_p \beta_{k+1}$. The top c-degree β_1 is for constraints that are 'fully certain', while the bottom c-degree β_{k+1} is for constraints that are 'not certain at all'.

We define by which c-degree an SQL constraint holds on a p-SQL table. Similar to marginal probabilities in probability theory, we call this c-degree the *marginal certainty*. In SQL tables an SQL constraint either holds or does not hold. In a p-SQL table, an SQL constraint always holds with some c-degree.

Definition 1 (Marginal certainty). *Let σ denote an SQL key, FD or `NOT NULL` constraint over table schema T. The* marginal certainty $c_t(\sigma)$ *by which σ holds in the p-SQL table t over $(T, \mathcal{S}_p)$ is the c-degree β_{k+2-i} that corresponds to the p-degree α_i of the smallest possible world t_i of t in which σ is violated, that is, $c_t(\sigma) = \beta_1$ if $\models_{t_k} \sigma$, and $c_t(\sigma) = \min\{\beta_{k+2-i} \mid \not\models_{t_i} \sigma\}$ otherwise.*

For example, when t denotes the p-SQL table of Table 1, then $c_t(dpt \rightarrow mng) = \beta_1$, $c_t(emp \rightarrow dpt) = \beta_2$, $c_t(u(emp)) = \beta_3$, and $c_t(n(emp)) = \beta_4$.

Constraints specify the semantics of an application domain. They govern which databases are regarded as meaningful for the application. We classify a p-SQL table as meaningful whenever it satisfies a given set of possibilistic constraints (σ, β) (key, FD, `NOT NULL` constraint), which allow us to stipulate the minimum marginal c-degree β by which the constraint σ must hold in every p-SQL table that is considered to be meaningful in the application domain.

Definition 2 (Possibilistic constraints). *Let $(T, \mathcal{S}_P)$ denote a p-SQL table schema. A* possibilistic SQL key*,* possibilistic SQL FD*, or* possibilistic `NOT NULL` constraint *is a pair (σ, β) where σ denotes an SQL key, FD or `NOT NULL` constraint over T, respectively, and β denotes a c-degree from $\mathcal{S}_P^T$. The p-constraint (σ, β_i) is* satisfied *by a p-SQL table t over $(T, \mathcal{S}_P)$ iff $c_t(\sigma) \geq \beta_i$.*

For example, when t denotes the p-SQL table of Table 1, then the following examples of p-constraints are satisfied by t: $(dpt \rightarrow mng, \beta_3)$ since $c_t(dpt \rightarrow mng) = \beta_1 \geq \beta_3$, $(emp \rightarrow dpt, \beta_2)$ since $c_t(emp \rightarrow dpt) = \beta_2 \geq \beta_2$, and $(u(emp), \beta_4)$ since $c_t(u(emp)) = \beta_3 \geq \beta_4$. In other words, t satisfies these three constraints. On the other hand, t violates (i.e. does not satisfy) any of the following p-constraints: $(emp \rightarrow dpt, \beta_1)$ since $c_t(emp \rightarrow dpt) = \beta_2 < \beta_1$, $(u(emp), \beta_2)$ since $c_t(u(emp)) = \beta_3 < \beta_2$, and $(n(emp), \beta_3)$ since $c_t(n(emp)) = \beta_4 < \beta_3$.

6 Conclusion and Future Work

We aim at a comprehensive toolbox for reasoning about the integrity of real-world entities in semi-structured big data. As underlying data model we chose a possibilistic extension of SQL. We showed how previous work captures some of the big data dimensions as special cases. Our definition of possibilistic keys, FDs, and `NOT NULL` constraints lays the foundation for investigating fundamental reasoning tasks for them in the future.

Indeed, different approaches should be applied to the big data dimensions, such as probabilistic approaches to the veracity dimension [3], different approaches of handling missing information to the variety dimension such as embedded keys and FDs [20], and different approaches to entity integrity such as key sets [19]. In a different direction, we may want to add further big data dimensions. For example, temporal extensions [11] may support the velocity dimension.

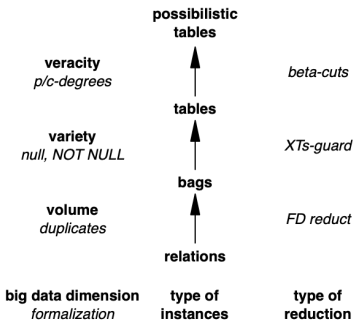

Fig. 3: Summary

References

1. Amalina, F., Hashem, I.A.T., Azizul, Z.H., Ang, T.F., Firdaus, A., Imran, M., Anuar, N.B.: Blending big data analytics: Review on challenges and a recent study. IEEE Access 8, 3629–3645 (2020)
2. Armstrong, W.W.: Dependency Structures of Data Base Relationships. In: Proc. of IFIP World Computer Congress. pp. 580–583 (1974)
3. Brown, P., Link, S.: Probabilistic keys. IEEE Trans. Knowl. Data Eng. 29(3), 670–682 (2017)
4. Christophides, V., Efthymiou, V., Stefanidis, K.: Entity Resolution in the Web of Data. Synthesis Lectures on the Semantic Web, Morgan & Claypool Publishers (2015)
5. Codd, E.F.: A relational model of data for large shared data banks. Commun. ACM 13(6), 377–387 (1970)
6. Date, C.J.: A critique of the SQL database language. SIGMOD Record 14(3), 8–54 (1984)
7. Diederich, J., Milton, J.: New methods and fast algorithms for database normalization. ACM Trans. Database Syst. 13(3), 339–365 (1988)
8. Dubois, D., Prade, H., Schockaert, S.: Generalized possibilistic logic: Foundations and applications to qualitative reasoning about uncertainty. Artif. Intell. 252, 139–174 (2017)
9. Ganti, V., Sarma, A.D.: Data Cleaning: A Practical Perspective. Synthesis Lectures on Data Management, Morgan & Claypool Publishers (2013)
10. Hartmann, S., Link, S.: The implication problem of data dependencies over SQL table definitions: Axiomatic, algorithmic and logical characterizations. ACM Trans. Database Syst. 37(2), 13:1–13:40 (2012)
11. Jensen, C.S., Snodgrass, R.T., Soo, M.D.: Extending existing dependency theory to temporal databases. IEEE Trans. Knowl. Data Eng. 8(4), 563–582 (1996)
12. Köhler, H., Link, S.: Armstrong axioms and Boyce-Codd-Heath normal form under bag semantics. Inf. Process. Lett. 110(16), 717–724 (2010)
13. Lien, Y.E.: On the equivalence of database models. J. ACM 29(2), 333–362 (1982)
14. Link, S., Prade, H.: Possibilistic functional dependencies and their relationship to possibility theory. IEEE Trans. Fuzzy Systems 24(3), 757–763 (2016)
15. Link, S., Prade, H.: Relational database schema design for uncertain data. Inf. Syst. 84, 88–110 (2019)
16. Liu, Z.H., Hammerschmidt, B.C., McMahon, D.: JSON data management: supporting schema-less development in RDBMS. In: International Conference on Management of Data, SIGMOD 2014, Snowbird, UT, USA, June 22-27, 2014. pp. 1247–1258 (2014)
17. Suciu, D., Olteanu, D., Ré, C., Koch, C.: Probabilistic Databases. Synthesis Lectures on Data Management, Morgan & Claypool Publishers (2011)
18. Thalheim, B.: Dependencies in relational databases. Teubner (1991)
19. Thalheim, B.: On semantic issues connected with keys in relational databases permitting null values. Elektronische Informationsverarbeitung und Kybernetik 25(1/2), 11–20 (1989)
20. Wei, Z., Link, S.: Embedded functional dependencies and data-completeness tailored database design. PVLDB 12(11), 1458–1470 (2019)
21. Zaniolo, C.: Database relations with null values. J. Comput. Syst. Sci. 28(1), 142–166 (1984)