Database dependencies

Non Peer-reviewed author version

GYSSENS, Marc (2009) Database dependencies. In: Liu, Ling & Özsu, M. Tamer (Ed.) Encyclopedia of Database Systems, p. 704-708..

DOI: 10.1007/978-0-387-39940-9_1236
Handle: http://hdl.handle.net/1942/10732

DATABASE DEPENDENCIES

Marc Gyssens, Hasselt University and Transnational University of Limburg, marc.gyssens@uhasselt.be

SYNONYMS

Database constraints

DEFINITION

For a relational database to be valid, it is not sufficient that the various tables of which it is composed conform to the database schema. In addition, the instance must also conform to the intended meaning of the database. [15] While many aspects of this intended meaning are inherently informal, it will in general induce certain formalizable relationships between the data in the database, in the following sense: whenever a certain pattern is present among the data, this pattern can either be extended or certain data values must be equal. Such a relationship is called a database dependency. The vast majority of database dependencies in the literature are of the following form [5]:

$$(\forall x_1)\dots(\forall x_n)\varphi(x_1,\dots,x_n) \Rightarrow (\exists z_1)\dots(\exists z_k)\psi(y_1,\dots,y_m,z_1,\dots,z_k).$$

Here, $\{y_1,\dots,y_m\} \subseteq \{x_1,\dots,x_n\}$, φ is a (possibly empty) conjunction of relation atoms using all the variables $x_1,\dots,x_n$, and ψ is either a single equality atom involving universally quantified variables only (in which case the dependency is called *equality-generating*); or ψ is a non-empty conjunction of relation atoms involving all the variables $y_1,\dots,y_m,z_1,\dots,z_k$ (in which case the dependency is called *tuple-generating*. A tuple-generating dependency is called *full* if it has no existential quantifiers; in the other case, it is called *embedded*.

HISTORICAL BACKGROUND

The theory of database dependencies started with the introduction of *functional dependencies* by Codd in his seminal paper. [8] They are a generalization of (super)keys. A relation satisfies a functional dependency $X \rightarrow Y$ (where X and Y are sets of attributes) if, whenever to tuples agree on X they also agree on Y. For example, if in a employee relation of a company database with schema

$$\Omega = \{\textit{EMP-NR}, \textit{EMP-NAME}, \textit{DEPT}, \textit{JOB}, \textit{SALARY}\},$$

the functional dependencies

$$\begin{array}{lcl} \{\textit{EMP-NR}\} & \rightarrow & \{\textit{EMP-NAME}, \textit{DEPT}, \textit{JOB}, \textit{SALARY}\}; \\ \{\textit{DEPT}, \textit{JOB}\} & \rightarrow & \{\textit{SALARY}\} \end{array}$$

hold, this means that *EMP-NR* is a key of this relation, i.e., uniquely determines the values of the other attributes, and that *JOB* in combination with *DEPT* uniquely determines *SALARY*.

Codd also noticed that the presence of a functional dependency $X \rightarrow Y$ also allowed a lossless decomposition of the relation into its projections onto $X \cup Y$ and $X \cup \overline{Y}$ ($\overline{Y}$ denoting the complement of Y). In the example above, the presence of $\{\textit{DEPT}, \textit{JOB}\} \rightarrow \{\textit{SALARY}\}$ allows for the decomposition of the original relation into its projections onto $\{\textit{DEPT}, \textit{JOB}, \textit{SALARY}\}$ and $\{\textit{EMP-NR}, \textit{EMP-NAME}, \textit{DEPT}\}$.

Hence, the identification of constraints was not only useful for integrity checking but also for more efficient representation of the data and avoiding update anomalies through redundancy removal.

Subseqent researchers (e.g., [18]) noticed independently that the presence of the functional dependency $X \to Y$ is a sufficient condition for decomposability of the relation into its projection onto $X \cup Y$ and $X \cup \overline{Y}$, but not a necessary one. For example,

DRINKER	*BEER*	*BAR*
Jones	*Tuborg*	*Tivoli*
Smith	*Tuborg*	*Far West*
Jones	*Tuborg*	*Tivoli*
Smith	*Tuborg*	*Tivoli*

can be decomposed losslessly into its projections onto $\{DRINKER, BEER\}$ and $\{BEER, BAR\}$, but neither $\{BEER\} \to \{DRINKER\}$ nor $\{BEER\} \to \{BAR\}$ holds. This led to the introduction of the *multivalued dependency*: a relation satisfies the multivalued dependency $X \twoheadrightarrow Y$ exactly when this relation can be decomposed losslessly into its projections onto $X \cup Y$ and $X \cup \overline{Y}$. Fagin [10] also introduced *embedded multivalued dependencies*: A relation satisfies the embedded multivalued dependency $X \twoheadrightarrow Y|Z$ if its projection onto $X \cup Y \cup Z$ can be decomposed losslessly into its projections onto $X \cup Y$ and $X \cup Z$. Sometimes, however, a relation be decomposed losslessly into three or more of its projections but not in two. This led Rissanen [17] to introduce a more general notion: a relation satisfies a *join dependency* $X_1 \bowtie \cdots \bowtie X_k$ if it can be decomposed losslessly into its projections onto $X_1, \ldots, X_k$.

Quite different considerations led to the introduction of *inclusion dependencies* [6], which are based on the concept of referential integrity, already known to the broader database community in the 1970s. As an example, consider a company database in which one relation, *MANAGERS*, contains information on department managers, in particular, *MAN-NAME*, and another, *EMPLOYEES*, contains general information on employees, in particular, *EMP-NAME*. As each manager is also an employee, every value *MAN-NAME* in *MANAGERS* must also occur as a value of *EMP-NAME* in *EMPLOYEES*. This is written as the inclusion dependency $MANAGERS[MAN\text{-}NAME] \subseteq EMPLOYEES[EMP\text{-}NAME]$. More generally, a database satisfies the inclusion dependency $R[A_1, \ldots, A_n] \subseteq S[B_1, \ldots, B_m]$ if the projection of the relation R onto the sequence of attributes $A_1, \ldots, A_n$ is contained in the the projection of the relation S onto the sequence of attributes $B_1, \ldots, B_n$.

The proliferation of dependency types motivated researchers to propose subsequent generalizations, eventually leading to the tuple- and equality-generating dependencies of Beeri and Vardi [5] defined higher. For a complete overview, the reader isd referred to [14] or the bibliographic sections in [2]. For the sake of completeness, it should also be mentioned that dependency types have been considered that are not captured by the formalism of Beeri and Vardi. An example is the *afunctional dependency* of De Bra and Paredaens (see, e.g., Chapter 5 of [15]).

SCIENTIFIC FUNDAMENTALS

The development of database dependency theory has been driven mainly by two concerns. One of them is solving the inference problem, and, when decidable, developing tools for deciding it. The other is, as pointed out in the historical background, the use of database dependencies to achieve decompositions of the database contributing to more efficient data representation, redundancy removal, and avoiding update anomalies. Each of these concerns is discussed in some more detail below.

Inference

The inference problem is discussed here in the context of tuple- and equality-generating dependencies. The question that must be answered is the following: given a subtype of the tuple- and equality generating dependencies, given as input a set of constraints $\mathcal{C}$ and a single constraint c, both of the given type, is it decidable whether $\mathcal{C}$ logically implies c In other words, is it decidable if each database instance satisfying $\mathcal{C}$ also satisfies c? Given that database dependencies have been defined as first-order sentences, one might be inclined to think that the inference problem is just an instance of the implication problem in mathematical logic. However, for logical implication, one must consider all models of the given database scheme, also those containing infinite relations, while database relations are by definition finite. (In other words, the study of the inference of database dependencies lies within finite model theory.) To separate both notions of inference, a distinction is made between *unrestricted implication* (denoted $\mathcal{C} \models c$) and *finite implication* (denoted $\mathcal{C} \models_f c$) [5]. Since unrestricted implication is recursively enumerable and finite implication is co-recursively enumerable, their coincidence yields that the finite implication problem is decidable. The opposite, however is not true, as is shown by the following counterexample. Consider a database consisting of a single relation R with scheme $\{A, B\}$. Let

$\mathcal{C} = \{B \to A, R[B] \subseteq R[A]\}$ and let c be the inclusion dependency $R[A] \subseteq R[B]$. One can show that $\mathcal{C} \models_f c$, but $\mathcal{C} \not\models c$, as illustrated by the following, necessarily infinite, counterexample:

A	B
0	1
1	2
2	3
3	4
⋮	⋮

As will be pointed out later, the finite implication problem for functional dependencies and so-called *unary* inclusion dependencies (i.e., involving only one attribute in each side) is decidable.

An important tool for deciding (unrestricted) implication is the *chase*. In the chase, a table is created for each relation in the database. For each relation atom in the left-hand side of the dependency c to be inferred, its tuple of variables is inserted in the corresponding table. This set of tables is then *chased* with the dependencies of $\mathcal{C}$: in the case of a tuple-generating dependency, new tuples are added in a minimal way until the dependency is satisfied (in each application, new variables are substituted for existential variables); in the case of an equality-generating dependency, variables are equated until the dependency is satisfied. The result, $chase(\mathcal{C})$, which may be infinite, can be seen as a model for $\mathcal{C}$. It is the case that $\mathcal{C} \models c$ if and only if the right-hand side of c is subsumed by some tuple of $chase(\mathcal{C})$ (in the case of a tuple-generating dependency) or the required equality has been aplied during the chase procedure.

In the case where only *full* tuple-generating dependencies and equality-generating dependencies are involved, the chase procedure is bound to end, as no existential variables occur in the dependencies, whence no new values are introduced. In particular, the unrestricted implication problems coincides with the finite implication problem, and is therefore decidable. Deciding this inference problem is EXPTIME-complete, however.

The inference problem for all tuple- and equality-generating dependencies is undecidable, however (whence unrestricted and finite implication do not coincide). In 1992, Herrmann [13] solved a longstanding open problem by showing that the finite implication problem is already undecidable for embedded multivalued dependencies.

Another approach towards deciding inference of dependency types is trying to find an *axiomatization*: a finite set of inference rules that is both sound and complete. The existence of such an axiomatization is also a sufficient condition for the decidability of inference. Historically, Armstrong [1] was the first to propose such an axiomatization for functional dependencies. This system of inference rules was eventually extended to a sound and complete axiomatization for functional and multivalued dependencies together: [3]

(F1)	$\emptyset \models X \to Y$ if $Y \subseteq X$	(reflexivity)
(F2)	$\{X \to Y\} \models XZ \to YZ$	(augmentation)
(F3)	$\{X \to Y, Y \to Z\} \models X \to Z$	(transitivity)
(M1)	$\{X \twoheadrightarrow Y\} \models X \twoheadrightarrow \overline{Y}$	(complementation)
(M2)	$\emptyset \models X \twoheadrightarrow Y$ if $Y \subseteq X$	(reflexivity)
(M3)	$\{X \twoheadrightarrow Y\} \models XZ \twoheadrightarrow YZ$	(augmentation)
(M4)	$\{X \twoheadrightarrow Y, Y \twoheadrightarrow Z\} \models X \twoheadrightarrow Z - Y$	(pseudo-transitivity)
(FM1)	$\{X \to Y\} \models X \twoheadrightarrow Y$	(conversion)
(FM2)	$\{X \twoheadrightarrow Y, Y \to Z\} \models X \to Z - Y$	(interaction)

Moreover, (F1)–(F3) are sound and complete for the inference of functional dependencies alone, and (M1)–(M4) are sound and complete form the inference of multivalued dependencies alone. The above axiomatization is at the basis of an algorithm to decide inference of functional and multivalued dependencies in low polynomial time. Of course, the inference problem for join dependencies is also decidable, as they are full tuple-generating dependencies. However, there does not exist a sound and complete axiomatization for the inference of join dependencies [16], even though there does exist an axiomatization for a larger class of database dependencies.

There also exists a sound and complete axiomatization for inclusion dependencies: [6]

(I1)	$\emptyset \models R[X] \subseteq R[X]$	(reflexivity)
(I2)	$\{R[A_1, \ldots, A_m] \subseteq S[B_1, \ldots, B_m]\} \models R[A_{i_1}, \ldots, A_{i_k}] \subseteq S[B_{i_1}, \ldots, B_{i_k}]$ if $i_1, \ldots, i_k$ is a sequence of integers in $\{1, \ldots, m\}$	(projection)
(I3)	$\{R[X] \subseteq S[Y], S[Y] \subseteq T[Z]\} \models R[X] \subseteq T[Z]$	(transitivity)

Above, X, Y, and Z represent sequences rather than sets of attributes. Consequently, the implication problem

for inclusion dependencies is decidable, even though inclusion dependencies are embedded tuple-generating dependencies. However, deciding implication of inclusion dependencies is PSPACE-complete.
It has already been observed above that the unrestricted and finite implication problems for functional dependencies and unary inclusion dependencies taken together do no coincide. Nevertheless, the finite implication problem for this class of dependencies is decidable. Unfortunately, the finite implication problem for functional dependencies and general inclusion dependencies taken together is undecidable (e.g., [7]).

Decompositions

As researchers realized that the presence of functional dependencies yields the possibility to decompose the database, the question arose as to how far this decomposition process ought to be taken. This led Codd in follow-up papers to [8] to introduce several *normal forms*, the most ambitious of which is *Boyce-Codd Normal Form (BCNF)*. A datatabase is in BCNF if, whenever one of its relations satisfies a nontrivial functional dependency $X \rightarrow Y$ (i.e., where Y is not a subset of X), X must be a superkey of the relation (i.e., the functional dependency $X \rightarrow U$ holds, where U is the set of all attributes of that relation). There exist algorithms that construct a lossless BCNF decomposition for a given relation. Unfortunately, it is not guaranteed that such a decomposition is also dependency-preserving, in the following sense: the set of functional dependencies that hold in the relations of the decomposition and that can be inferred from the given functional dependencies is in general not equivalent with the set of the given functional dependencies. Even worse, a dependency-preserving BCNF decomposition of a given relation does not always exist. For that reason, *Third Normal Form (3NF)*, historically a precursor to BCNF, is also still considered. A datatabase is in 3NF if, whenever one of its relations satisfies a nontrivial functional dependency $X \rightarrow \{A\}$ (A being a single attribute), the relation must have a minimal key containing A. Every database in BCNF is also in 3NF, but not the other way around. However, there exists an algorithm that, given a relation, produces a dependency-preservering lossless decomposition in 3NF. Several other normal forms have also been considered, taking into account multivalued dependencies or join dependencies besides functional dependencies.
However, one can argue that by giving a join dependency, one actually already specifies how one wants to decompose a database. If one stores this decomposed database rather than the original one, the focus shifts from integrity checking to *consistency checking*: can the various relations of the decompositions be interpreted as the projections of a universal relation? Unfortunately, consitency checking is in general exponential in the number of relations. Therefore, a lot of attention has been given to so-called *acyclic join dependencies* [4]. There are many equivalent definitions of this notion, one of which is that an acyclic join dependency is equivalent to a set of multivalued dependencies. Also, global consistency of a decomposition is already implied by pairwise consistency if and only if the join dependency defining the decomposition is acyclic, which explains in part the desirability of acyclicity. Gyssens [12] generalized the notion of acyclicity to k-cyclicity, where acyclicity corresponds with the case $k = 2$. A join dependency is k-cyclic if it is equivalent to a set of join dependencies each of which has at most k components. Also, global consistency of a decomposition is already implied by k-wise consistency if and only if the join dependency defining the decomposition is k-cyclic.

KEY APPLICATIONS*

Despite the explosion of dependency types during the latter half of the 1970s, one must realize that the dependency types most used in practice are still functional dependencies (in particular, key dependencies) and inclusion dependencies. It is therefore unfortunate that the inference problem for functional and inclusion dependencies combined is undecidable.
At a more theoretical level, the success of studying database constraints from a logical point view and the awareness that is important to distinguish beteen unrestricted and finite implication, certainly contributed to the interest in and study and further development of finite model theory by theoretical computer scientists.
Finally, decompositions of join dependencies led to a theory of decompositions for underlying hypergraphs, which found applications in other areas as well, notably in artificial intelligence (e.g., [11, 9]).

CROSS REFERENCE*

2nd Normal Form (2NF), 3rd Normal Form (3NF), 4th Normal Form (4NF), Boyce-Codd Normal Form (BCNF), Chase, Equality-generating dependencies, Functional Dependency, Inconsistent databases, Join dependency,

Multivalued dependency, Normal forms and normalization, Relational Model, Tuple-generating dependencies.

RECOMMENDED READING

Between 5 and 15 citations to important literature, e.g., in journals, conference proceedings, and websites.

[1] W.W. Armstrong. Dependency Structures of Data Base Relationships. In Proceedings *IFIP Congress 74*, Stockholm, Sweden, August 5–10, 1974, J.L. Rosenfeld (Ed.), pp. 580–583. North-Holland, 1974.

[2] S. Abiteboul, R. Hull, V. Vianu. *Foundations of Databases*. Addison-Wesley, Reading, Mass., 1995. (Part C.)

[3] C. Beeri, R. Fagin, J.H. Howard. A Complete Axiomatization for Functional and Multivalued Dependencies. In Proceedings *ACM SIGMOD International Conference on Management of Data*, Toronto, Ontario, August 3–5, 1977, D.C.P.. Smith (Ed.), pp. 47–61. ACM Press, New York, 1978.

[4] C. Beeri, R. Fagin, D. Maier, M. Yannakakis. On the Desirability of Acyclic Database Schemes. *Journal of the ACM*, 30(3):479–513, 1983.

[5] C. Beeri, M.Y. Vardi. The Implication Problem for Data Dependencies. In Proceedings *International Conference on Algorithms, Languages, and Programming*, Acre, Israel, July 13–17, 1981, S. Even, O. Kariv (Eds.). Lecture Notes in Computer Science, Vol. 115, pp. 73–85. Springer-Verlag, Berlin/New York, 1981.

[6] M.A. Casanova, R. Fagin, C.H. Papadimitriou. Inclusion Dependencies and their Interaction with Functional Dependencies. *Journal of Computer and System Sciences*, 28(1):29–59, 1984.

[7] A.K. Chandra, M.Y. Vardi. The Implication Problem for Functional and Inclusion Dependencies is Undecidable. *SIAM Journal on Computing*, 14(3)-671–677, 1985.

[8] E.F. Codd. A Relational Model of Data for Large Shared Data Banks. *Communications of the ACM*, 13(6):377–387, 1970.

[9] D.A. Cohen, P. Jeavons, M. Gyssens. A Unified Theory of Structural Tractability for Constraint Satisfaction Problems. *Journal of Computer and System Sciences*, 2008, *in press*.

[10] R. Fagin. Multivalued Dependencies and a New Normal Form for Relational Databases. *ACM Transactions on Database Systems*, 2(3):262–278, 1977.

[11] G. Gottlob, Z. Miklós, T.Schwentick. Generalized Hypertree Decompositions: NP-Hardness and Tractable Variants. In Proceedings *Twenty-sixth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, Beijing, China, June 11–13, L. Libkin (Ed.), pp. 13–22. ACM Press, New York, 2007.

[12] M. Gyssens. On the Complexity of Join Dependencies. *Transaction on Database Systems*, 11(1):81–108, 1986.

[13] C. Herrmann. On the Undecidability of Implications between Embedded Multivalued Dependencies. *Information and Computation*, 122(2):221–235, 1995.

[14] P.C. Kanellakis. Elements of Relational Database Theory. In *Handbook of Theoretical Computer Science*, J. Van Leeuwen (Ed.), pp. 1074–1156. Elsevier, Amsterdam, 1991.

[15] J. Paredaens, P. De Bra, M. Gyssens, D. Van Gucht. *The Structure of the Relational Database Model*. EATCS Monographs on Theoretical Computer Science, W. Brauer, G. Rozenberg, A. Salomaa (Eds.), Vol. 17. Springer-Verlag, Berlin/New York, 1989.

[16] S.V. Petrov. Finite Axiomatization of Languages for Representation of System Properties. *Information Sciences*, 47(3):339–372, 1989.

[17] J. Rissanen. Independent Components of Relations. *ACM Transactions on Database Systems*, 2(4):317–325, 1977.

[18] C. Zaniolo. Analysis and Design opf Relational Schemata for Database Systems. Ph. D. thesis, University of California at Los Angeles, 1976. Technical Report UCLA-Eng-7669, Department of Computer Science.