Proceedings of the Eight Annual Workshop on Information Technologies and Systems WITS'98, Helsinki (FIN), December 12-13, 1998, pp 257-266. Best paper nominee.

A KDD Framework for Database Audit [1]

Jean-François Boulicaut

Institut National des Sciences Appliquées de Lyon

LISI Bâtiment 501

F-69621 Villeurbanne cedex, France

Jean-Francois.Boulicaut@insa-lyon.fr

Abstract

Understanding data semantics from real-life databases is considered following an *audit* perspective: it must help experts to analyse what properties actually hold in the data and support the comparison with desired properties. This is a typical problem of knowledge discovery in databases (KDD) and it is specified within the framework of Mannila and Toivonen where data mining consists in querying theories e.g., the theories of approximate inclusion dependencies. This formalization enables us to identify an important subtask to support database audit as well as a generic algorithm. Next, we consider the DREAM relational database reverse engineering method and DREAM heuristics are revisited within this new setting.

[1] To appear in Proceedings of the 8th Workshop on Information Technologies and Systems, December 12-13, 1998, Helsinki (Finland). This work has been done while the author was on sabbatical in the Department of Computer Science at the University of Helsinki (Finland). It is partly supported by AFFRST, Association Franco-Finlandaise pour la Recherche Scientifique et Technique.

1 Introduction

We are interested in understanding data semantics from real-life databases. This process is considered following an *audit* perspective in the following sense: it must help experts to analyse what properties actually hold in the data and support the comparison with desired properties. This condensed research paper takes examples from relational database audit, assuming that inclusion and functional dependencies that (almost) hold in the data capture the so-called data semantics. This will be called hereafter *the basic problem*. However, our framework can be applied to other kinds of databases and/or properties.

Auditing databases is an important topic. Integrity constraints that have been more or less explicited at the design time of a database may not always hold in a given instance. Understanding actual data semantics in databases is of a crucial interest to support their maintenance and evolution. The fact that some property holds or not in an instance can be used by experts to fix some integrity violation in the data or give rise to an explicit definition of an integrity constraint (e.g., for further use of built-in checking mechanisms). Improving our knowledge of encoded data semantics is also useful for semantic query optimization (see e.g., [1]). Last but not least, audit is an important preliminary step for a database reverse engineering process [3] or the design of federated databases. Indeed, solving the basic problem provides the raw knowledge that is needed to start a restructuring phase on a denormalized relational schema.

Auditing as querying multiple theories. Auditing databases is a typical problem of Knowledge Discovery in Databases (KDD). Discovering knowledge from databases can be seen as a process containing several steps: understanding the domain, preparing the data set, discovering patterns (e.g., dependencies), postprocessing of discovered patterns (e.g., selecting dependencies that should become integrity constraints), and putting the results into use [4]. This is a semi-automatic and iterative process that is often described using ad-hoc formalisms and/or notations. However, a general KDD framework has been proposed by Mannila and Toivonen [8]. Data mining consists in querying the so-called *theory* of the database w.r.t. a class of patterns and a selection predicate that defines their interestingness. Audit can then be supported by queries over relevant theories.

Contribution. First, we specify auditing tasks within this general KDD framework. The basic problem is formalized as mining theories of approximate inclusion and functional dependencies (see Section 2). This enables us to identify an important subtask to support database audit i.e., querying an intensionally defined collection of dependencies. A generic algorithm, the "guess-and-correct" scheme introduced in [8], is a good starting point for the evaluation of such queries (Section 3). Finally, in Section 4, we revisit the heuristics that constitute the core of the DREAM reverse engineering method [9, 10]. In this project, equi-joins that are performed in the application programs are used to support the discovery of "relevant" dependencies.

2 A Formal Framework for Database Audit

Notations The reader is supposed to be familiar with relational database concepts. Suppose $\mathbf{r}$ is a relational database instance over the schema $\mathbf{R}$. A *relation* $R_i(X_i)$ belongs to $\mathbf{R}$ and is defined by a relation name R_i and a set of *attributes* X_i. Each relation $R_i(X_i)$ is associated with a *table* r_i which is a set of *tuples*. The database extension $\mathbf{r}$ represents the set of tables r_i. $r_i[Y]$ is the *projection* of the table r_i on $Y \subseteq X_i$ and $t[Y]$ is the projection of the tuple t following Y. Let Y and Z be two subsets of X_i, a *functional dependency* denoted by $R_i : Y \rightarrow Z$ on $R_i(X_i)$ is true in r_i iff $\forall t, t' \in r_i \quad t[Y] = t'[Y] \Rightarrow t[Z] = t'[Z]$. It can be written $\mathbf{r} \models Y \rightarrow Z$. Let $R_i(X_i)$ and $R_j(X_j)$ be two relations associated with tables r_i and r_j respectively. Let Y (resp. Z) be a subset of attributes of X_i (resp. X_j). An *inclusion dependency* denoted by $R_i[Y] \subseteq R_j[Z]$ is true in r_i and r_j iff $r_i[Y] \subseteq r_j[Z]$. It can be written $\mathbf{r} \models R_i[Y] \subseteq R_j[Z]$.

2.1 Querying theories

Given a database instance $\mathbf{r}$, assume the definition of a language $\mathcal{L}$ for expressing properties of the data and a *selection predicate* q. The predicate q is used for evaluating whether a sentence $\varphi \in \mathcal{L}$ defines a potentially interesting property of $\mathbf{r}$. Therefore, a mining task is to compute the *theory* of $\mathbf{r}$ with respect to $\mathcal{L}$ and q, i.e., the set $\mathcal{T}h(\mathcal{L}, \mathbf{r}, q) = \{\varphi \in \mathcal{L} \mid q(\mathbf{r}, \varphi) \text{ is true}\}$. A reasonnable collection of data mining tasks have already been carried out using this approach (see [7] for a survey).

Example 1 *Consider the discovery of dependencies that hold in a database. Assume* $\mathcal{L}_1$ *is the language of inclusion dependencies and consider* q_1 *as the satisfaction predicate:*

let r and s be the relations corresponding to R and S and $\delta = R[X] \subseteq S[Y] \in \mathcal{L}_1$, $q_1(\mathbf{r}, \delta)$ is true iff $\mathbf{r} \models \delta$. Now, assume $\mathcal{L}_2$ is the language of functional dependencies and that here again, the predicate q_2 is the satisfaction predicate. □

Looking for a generic data mining technique, we can define a specialization relation on $\mathcal{L}$ and then use a simple levelwise algorithm to compute $\mathcal{T}h(\mathcal{L}, \mathbf{r}, q)$ [8]. A *specialization relation* $\preceq$ is a partial order: φ is *more general* than θ, if $\varphi \preceq \theta$ (θ is *more specific* than φ). It is a *monotone specialization relation* w.r.t. q if for all $\mathbf{r}$ and φ, if $q(\mathbf{r}, \varphi)$ and $\gamma \preceq \varphi$ then $q(\mathbf{r}, \gamma)$. In other words, if a sentence φ satisfies q, then also all more general sentences γ satisfy q. A simple but powerful "generate-and-test" algorithm can now be derived: start from the most general sentences and then try to generate and to evaluate more and more specific sentences, but do *not* evaluate those sentences that are not interesting given the available information. Such an algorithm moves gradually to the interesting more specific sentences.

Example 2 *Continuing Example 1, a monotone specialization relation w.r.t. inclusion dependencies is defined as follows: for $\varphi = R[X] \subseteq S[Y]$ and $\theta = R'[X'] \subseteq S'[Y']$, we have $\varphi \preceq_1 \theta$ only if $R = R'$, $S = S'$, and furthermore $X' = \langle A_1, \ldots, A_k \rangle, Y' = \langle B_1, \ldots, B_k \rangle$, and for some disjoint $i_1, \ldots, i_h \in \{1, \ldots, k\}$ with $h < k$ we have $X = \langle A_{i_1}, \ldots, A_{i_h} \rangle$, $Y = \langle B_{i_1}, \ldots, B_{i_h} \rangle$. For instance, given $R = \{A, B, C, D\}$ and $S = \{E, F, G\}$, $R[\langle A \rangle] \subseteq S[\langle E \rangle] \preceq_1 R[\langle A, B \rangle] \subseteq S[\langle E, F \rangle]$. The most general sentences are all the unary inclusion dependencies. Now, assuming the restriction to functional dependencies with a fixed right-hand side denoted as B, a monotone specialization relation w.r.t. functional dependencies is the reverse of set inclusion: for $X, Y \subseteq R$ and $B \in R$ we have $X \to B \preceq_2 Y \to B$ iff $Y \subseteq X$. For instance, $AB \to D \preceq_2 A \to D$. The most general sentences have the whole set R as the left-hand side. The selection predicate q_1 (resp. q_2) is monotone w.r.t. $\preceq_1$ (resp. $\preceq_2$). For instance, if $R[\langle A, B \rangle] \subseteq S[\langle E, F \rangle]$ holds then $R[\langle A \rangle] \subseteq S[\langle E \rangle]$ holds and, if $A \to D$ holds then $AB \to D$ holds.* □

Such a simple approach has been already implemented for inclusion and functional dependency computation (see [8] for a complexity analysis and pointers to related work). It provides the best known algorithm for inclusion dependency discovery. For functional dependencies, better algorithms are available (e.g., [5]).

This provides a good starting point to support audit tasks. However, the computation of the most interesting sentences in a theory can be quite slow if there are interesting statements that are far from the most general sentences. Furthermore, a framework designed to support (basic) audit task might consider some important specificities of this kind of application. First, one should not consider only exact dependencies: we want to study dependencies even in the case where some tuples violate these constraints. Next, the expert user is quite often interested in tightly specified subset of the dependencies that hold. Furthermore, quite often, dependencies do not have to be computed from scratch i.e., either the expert user has already a good knowledge of constraints that should hold and/or the computation of constraints has been already done on a previous state of the database. These observations motivate the definition of the theories of approximate dependencies, a querying approach over intensionally defined theories, and the use of an interesting variation of the levelwise algorithm, the so-called "guess-and-correct" scheme.

2.2 Solving the basic problem

Computing approximate dependencies Inconsistencies in the database can be allowed by defining $q'(\mathbf{r}, \delta)$ to be true iff some error measure of the dependency δ is lower or equal to a user-defined threshold. Let us define an error measure g for the inclusion dependency $\delta = R[X] \subseteq S[Y]$ in $\mathbf{r}$: $g(\delta, \mathbf{r}) = 1$ - max $\{|r'| \mid r' \subseteq r \wedge (r' \cup (\mathbf{r} \setminus r)) \models \delta\}/|r|$ where r is the instance of R. Using g enables to consider dependencies that almost hold since it gives the proportion of tuples that must be removed from r to get a true dependency. Among the several ways of defining approximate functional dependencies in an instance r of R, one can also consider the minimum number of rows that need to be removed from r for the dependency $\gamma = R : X \rightarrow B$ to hold (the so-called g_3 error measure in [5]): $g_3(\gamma, \mathbf{r}) = 1 - max\{|r'| \mid r' \subseteq r \wedge r' \models \gamma\}/|r|$.

For instance, assuming the following instances of $R = \{A, B, C, D\}$ and $S = \{E, F, G\}$, a few approximate inclusion and functional dependencies are given.

A	B	C	D
1	2	4	5
2	2	2	3
3	1	1	2
4	2	2	3

E	F	G
1	2	3
2	3	4
3	2	2

Inclusion dep.	Error	Functional dep.	Error
$R[\langle B \rangle] \subseteq S[\langle E \rangle]$	0	$A \rightarrow B$	0
$R[\langle D \rangle] \subseteq S[\langle E \rangle]$	0.25	$AB \rightarrow C$	0
$S[\langle E \rangle] \subseteq R[\langle B \rangle]$	0.33	$BC \rightarrow A$	0.25
$R[\langle C, D \rangle] \subseteq S[\langle E, F \rangle]$	0.25	$BCD \rightarrow A$	0.25

The selection predicate q_1 (resp. q_2) is now modified to denote that all the approximate inclusion (resp. functional) dependencies whose error is lower or equal to a user-supplied threshold are desired. These predicates remain monotone w.r.t. their respective specialization relations.

Now, one naive approach to solve the basic problem might be to compute theories of approximate dependencies for some error thresholds, store them in a "SQL3" table and then query such tables using available query languages. This is a typical approach in many KDD applications where interestingness of patterns is considered in a postprocessing phase while pattern discovery is mainly guided by simple criteria like statistical significance or error thresholds. This gives rise to several problems. The size of such theories can be huge while the expert user is interested in only a few dependencies. Not only it might be untractable to compute the whole theories but also it gives rise to tedious postprocessing phases (e.g., a posteriori elimination of redundancies). It motivates a flexible querying framework that support the analysis of tightly specified theories, merging the traditional data mining step with some typical postprocessing steps.

Querying tightly specified theories It happens that, a priori, only small subsets of $\mathcal{L}_1$ or $\mathcal{L}_2$ are interesting. Restrictions of practical interest concern non trivial inclusion or non trivial functional dependencies, unary inclusion dependencies but also various selection criteria over attributes. In fact, only expert users can define them and, technically, it is possible to define such criteria as context-sensitive restrictions to the definition of $\mathcal{L}_1$ and $\mathcal{L}_2$ or to integrate them in the definition of the selection predicate. These different views influence the computation process and its efficiency: it is more or less a "generate-and-test" scheme when the generation of candidate dependencies can make an active use of given restrictions.

A typical audit process requires the computation of many related theories. Not only, several theories for the same dependency class are needed, depending of the dynamically evolving user's interest, but also different theories for different kind of dependencies might be useful. An obvious example is the audit of *referential integrity constraints* for which one must consider inclusion dependencies whose right-hand side set of attributes is a key i.e., a special case of functional dependency.

The conceptual framework of inductive databases has recently emerged, [2]. It sug-

gests an elegant approach to support audit or more generally data mining over multiple theories. An *inductive database*, is a database that contains inductive generalizations about the data, in addition to the usual data. Formally, the *schema of an inductive database* is a pair $\mathcal{R} = (\mathbf{R}, (\mathcal{Q}_{\mathbf{R}}, e, \mathcal{V}))$, where $\mathbf{R}$ is a database schema, $\mathcal{Q}_{\mathbf{R}}$ is a collection of patterns, $\mathcal{V}$ is a set of *result values*, and e is the *evaluation function* that defines how patterns occur in the data. The function e maps each pair $(\mathbf{r}, \theta_i)$ to an element of $\mathcal{V}$, where $\mathbf{r}$ is a database over $\mathbf{R}$ and θ_i is a pattern from $\mathcal{Q}_{\mathbf{R}}$. An *instance* $(\mathbf{r}, s)$ of an inductive database over the schema $\mathcal{R}$ consists of a database $\mathbf{r}$ over the schema $\mathbf{R}$ and a subset $s \subseteq \mathcal{Q}_{\mathbf{R}}$. For our basic problem, we need two inductive databases that associate to the database all the inclusion dependencies and functional dependencies that can be built from its schema. The evaluation functions respectively return the g and g_3 error measures as previously defined. At each stage of manipulating an inductive database $(\mathbf{r}, s)$, the user can think that the value of $e(\mathbf{r}, \theta)$ is available for each pattern θ which is present in the set s i.e., each dependency that is actually in s. He/she send queries over intensionally defined collections of all dependencies and selects only dependencies fulfilling some constraints. For example, a user might be interested in selecting only inclusion dependencies that involve a set A in their left-hand side, do not involve a set B in their right-hand side and finally have a g error value lower than 0.01. Obviously, an implementation will not compute all the values of the evaluation function beforehand; rather, only those values $e(\mathbf{r}, \theta)$ that user's queries require to be computed should be computed. The design and the implementation of such a query language is still an open problem for that kind of patterns though many ideas can be reused from association rule mining [2].

3 The "guess-and-correct" generic algorithm

[8] provides a generic algorithm that start the process of finding $\mathcal{T}h(\mathcal{L}, \mathbf{r}, q)$ from an initial guess $\mathcal{S} \subseteq \mathcal{L}$. It appears as an interesting basis for query evaluation. Consider a set $\mathcal{S} \subseteq \mathcal{L}$ closed downwards under $\preceq$, i.e., if $\varphi \in \mathcal{S}$ and $\gamma \preceq \varphi$, then $\gamma \in \mathcal{S}$ (by definition, this is true for a theory $\mathcal{T}h(\mathcal{L}, \mathbf{r}, q)$ w.r.t. a monotone specialization relation). The *border* $\mathcal{B}d(\mathcal{S})$ of $\mathcal{S}$ consists of those sentences φ such that all generalizations of φ are in $\mathcal{S}$, the so-called *positive border* $\mathcal{B}d^+(\mathcal{S})$, and none of the specializations of φ is in $\mathcal{S}$, the so called *negative*

border $\mathcal{B}d^-(\mathcal{S})$). $\mathcal{B}d^+(\mathcal{S})$ consists of the most specific sentences in $\mathcal{S}$, and $\mathcal{B}d^-(\mathcal{S})$ consists of the most general sentences that are not in $\mathcal{S}$.

Example 3 *Consider inclusion dependencies between* $R = \{A, B, C, D\}$ *and* $S = \{E, F, G\}$. *Assume that the collection of maximal nontrivial inclusion dependencies, i.e., the positive border of the theory is* $\{R[\langle A, B, D\rangle] \subseteq S[\langle G, F, E\rangle], R[\langle C\rangle] \subseteq S[\langle E\rangle], S[\langle E\rangle] \subseteq R[\langle C\rangle], S[\langle E\rangle] \subseteq R[\langle D\rangle]\}$. *Its negative border contains many (non-)dependencies like* $R[\langle A\rangle] \subseteq S[\langle E\rangle]$, *or* $R[\langle B\rangle] \subseteq S[\langle E\rangle]$. □

Algorithm 4 *The* "guess-and-correct" algorithm *[8]. Given, a database* $\mathbf{r}$, *a language* $\mathcal{L}$ *with specialization relation* $\preceq$, *a selection predicate* q, *and an initial guess* $\mathcal{S}$ *closed under generalizations, this algorithm outputs* $\mathcal{T}h(\mathcal{L}, \mathbf{r}, q)$.

1. $\mathcal{E} := \emptyset$;
2. $\mathcal{C} := \mathcal{B}d^+(\mathcal{S})$; *// correct* $\mathcal{S}$ *downward*
3. **while** $\mathcal{C} \neq \emptyset$ **do**
4. $\quad \mathcal{E} := \mathcal{E} \cup \mathcal{C}$;
5. $\quad \mathcal{S} := \mathcal{S} \setminus \{\varphi \in \mathcal{C} \mid q(\mathbf{r}, \varphi)$ *is false*$\}$;
6. $\quad \mathcal{C} := \mathcal{B}d^+(\mathcal{S}) \setminus \mathcal{E}$; **od**;
7. $\mathcal{C} := \mathcal{B}d^-(\mathcal{S}) \setminus \mathcal{E}$; *//* $\mathcal{S} \subseteq \mathcal{T}h(\mathcal{L}, \mathbf{r}, q)$; *expand* $\mathcal{S}$ *upwards*
8. **while** $\mathcal{C} \neq \emptyset$ **do**
9. $\quad \mathcal{S} := \mathcal{S} \cup \{\varphi \in \mathcal{C} \mid q(\mathbf{r}, \varphi)$ *is true*$\}$;
8. $\quad \mathcal{C} := \mathcal{B}d^-(\mathcal{S}); \setminus \mathcal{E}$; **od**;
9. *output* $\mathcal{S}$;

The algorithm first evaluates the sentences in the positive border $\mathcal{B}d^+(\mathcal{S})$ and removes from $\mathcal{S}$ those that are not interesting. These steps are repeated until the positive border only contains sentences satisfying q, and thus $\mathcal{S} \subseteq \mathcal{T}h(\mathcal{L}, \mathbf{r}, q)$. $\mathcal{E}$ is used to avoid evaluating sentences twice. Then the algorithm expands $\mathcal{S}$ upwards, it evaluates such sentences in the negative border $\mathcal{B}d^-(\mathcal{S})$ that have not been evaluated yet, and adds those that satisfy q to $\mathcal{S}$. Again, these steps are repeated until there are no sentences to evaluate. Finally, the output is $\mathcal{S} = \mathcal{T}h(\mathcal{L}, \mathbf{r}, q)$.

Example 5 *The discovery of (approximate) functional and inclusion dependencies in a database can be solved by algorithm 4 given the specialization relations we introduced.* □

Important results about the complexity of such a scheme can be found in [8] and can not be discussed here due to the lack of space. How to obtain good original guesses $\mathcal{S}$? One fairly widely applicable method is sampling: take a small sample $\mathbf{s}$ from $\mathbf{r}$, compute $\mathcal{T}h(\mathcal{L}, \mathbf{s}, q)$ and use it as $\mathcal{S}$. Another obvious situation where a guess is available is when a new audit is performed on the same database: most of the dependencies should have been preserved. Application programs can also be used to produce a guess.

4 Revisiting DREAM heuristic

An operational database is made of a dictionary that defines its schema, a dataset but also application programs. Research in database reverse engineering has shown that equi-joins performed in these application programs are an interesting source of information (see [3] for a survey). This is one of the key idea in the DREAM reverse engineering method [9, 10]. Informally, this method is based on the following heuristic: (a) equi-joins between sets of attributes that are embedded in application programs can be used to discover "relevant" inclusion dependencies, (b) non key attributes of inclusion dependencies are good candidates for the left-hand side of "relevant" functional dependencies. This heuristic obviously reduces the number of dependencies to be considered and "relevancy" refers to the interestingness of discovered dependencies for a reverse engineering process. The idea is that, in the context of database reverse engineering over denormalized schema, dependencies that are not only integrity constraints but that can also be used to support the identification of conceptual structures (e.g., the *hidden objects* of [6]) are desired.

Example 6 *Given* `emp={code,name,tel,add}, dept={dep,director,add}` *and the dependencies (1)* `dept[director]` $\subseteq$ `emp[code]` *(2)* `emp[add]` $\subseteq$ `dept[add]` *(3)* `code` $\rightarrow$ `name, tel, add` *and (4)* `tel` $\rightarrow$ `add`. *Dependencies (1) and (3) seem relevant for a restructuring phase while (2) and (4) are just integrity constraints. The DREAM heuristics rely on the assumption that* `dept.director` $\bowtie$ `emp.code` *is probably performed in application programs (pointing out the potentially interesting dependency (1) and that* `code` *is a candidate for a left-hand side of a potentially interesting functional dependencies) while* `dept.add` $\bowtie$ `emp.add` *probably do not occur.* □

These heuristics can be encoded in queries over the inductive databases of inclusion and functional dependencies, using selection criteria derived from equi-join occurrences

in the application programs. The collection of equi-joins can be considered as a theory whose computation requires non trivial compilation techniques. This might be a valuable source of information to support maintenance of application code as well as data semantics elicitation. Combining the different sources of information in order to produce guesses is an interesting research perspective.

5 Conclusions

We presented a framework for the audit of databases based on a KDD perspective. It provides a nice application domain for an ongoing research in generic data mining tools though it also gives concrete solutions to problems of practical interest. We must now study typical audit tasks. For instance, supporting the elicitation of referencial integrity constraints can be useful when moving from an old DBMS to a recent one. It needs not only to mine inclusion and functional dependencies but also the support of the search for erroneous data given a collection of almost true dependencies.

References

[1] S. Bell. Discovering rules in relational databases for semantic query optimisation. In *Proc. PADD'97, Practical Application Company*, pages 79 – 90, Apr. 1997.

[2] J.-F. Boulicaut, M. Klemettinen, and H. Mannila. Querying inductive databases: A case study on the MINE RULE operator. In *Proc. PKDD'98, Springer-Verlag LNAI 1510*, pages 194 – 202, Sept. 1998.

[3] R. H. L. Chiang, T. M. Barron, and V. C. Storey. A framework for the design and evaluation of reverse engineering methods for relational databases. *Data & Knowledge Engineering*, 21(1):57– 77, 1996.

[4] U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors. *Advances in Knowledge Discovery and Data Mining.* AAAI Press, 1996.

[5] Y. Huhtala, J. Kärkkäinen, P. Porkka, and H. Toivonen. Efficient discovery of functional and approximate dependencies using partitions. In *Proc. ICDE'98, IEEE Computer Society Press*, pages 392–401, Feb. 1998.

[6] P. Johannesson. A method for translating relational schemas into conceptual structures. In *Proc. ICDE'94, IEEE Computer Society Press*, pages 190 – 201, Feb. 1994.

[7] H. Mannila. Methods and problems in data mining. In *Proc. ICDT'97, Springer-Verlag LNCS 1186*, pages 41–55, 1997.

[8] H. Mannila and H. Toivonen. Levelwise search and borders of theories in knowledge discovery. *Data Mining and Knowledge Discovery*, 1(3):241 – 258, 1997.

[9] J.-M. Petit, J. Kouloumdjian, J.-F. Boulicaut, and F. Toumani. Usign queries to improve database reverse engineering. In *Proc. ER'94, Springer-Verlag LNCS 881*, pages 369 – 386, Oct. 1994.

[10] J.-M. Petit, F. Toumani, J.-F. Boulicaut, and J. Kouloumdjian. Towards the reverse engineering of denormalized relational databases. In *Proc. ICDE'96, IEEE Computer Society Press*, pages 218 – 227, Feb. 1996.