Discovery of “Interesting” Data Dependencies from a Workload of SQL Statements

S. Lopes J-M. Petit F. Toumani

Université Blaise Pascal
Laboratoire LIMOS
Campus Universitaire des Cézeaux
24 avenue des Landais
63 177 Aubière cedex, France
Tel. : 04-73-40-74-92 Fax. : 04-73-40-74-44
{slopes,jmpetit,ftoumani}@libd2.univ-bpclermont.fr

Abstract. Discovering data dependencies consists in producing the whole set of a given class of data dependencies holding in a database, the task of selecting the interesting ones being usually left to an expert user. In this paper we take another look at the problems of discovering inclusion and functional dependencies in relational databases. We define rigourously the so-called *logical navigation* from a workload of SQL statements. This assumption leads us to devise tractable algorithms for discovering “interesting” inclusion and functional dependencies.

1 Introduction and related works

The problem of discovering data dependencies can be formulated as follows: *given a database instance, find all non-trivial data dependencies satisfied by that particular instance* [9].

These problems are known to be hard, at least in the worst case [9]. Existing KDD approaches consist in producing the whole set of dependencies holding in a database [4, 1, 8, 3].

Nevertheless, some of the discovered dependencies can be accidental or erroneous: The task of selecting the “interesting” dependencies is left to an expert user. More generally, the issue of interestingness of discovered knowledge is known to be quite hard in data mining [6, 10].

Example 1. Consider the relations Instructor and Department of Table 1. An inclusion dependency holds from Instructor[rank] to Department[dnumber]. However, the attributes rank and dnumber do not represent the same information: rank gives the level of qualification of instructors whereas dnumber is an integer denoting the chronology of creation of departments. This kind of inclusion dependency is erroneous and dangerous to use in database applications: for instance, it does not make sense to enforce a referential integrity constraint from Instructor[rank] to Department[dnumber].

Instructor

ssn	status	rank
1	Assoc. Prof	1
2	Prof.	2
3	Assist.	1
4	Assist.	2
5	Prof.	2
6	Prof.	1
7	Assoc. Prof.	1

TeachesIn

ssn	dnum	year	depname
1	1	85	Biochemistry
1	5	94	Admission
2	2	92	C.S
3	2	98	C.S
4	3	98	Geophysics
5	1	75	Biochemistry
6	5	88	Admission

Department

dnumber	dname	mgr
1	Biochemistry	5
2	C.S.	2
3	Geophysics	2
4	Medical center	10
5	Admission	12
6	Genetic	6

Table 1. A relational database

In this paper, we consider the problem of finding "interesting" dependencies from the set of all functional and inclusion dependencies holding in a database.

We argue that "interesting" dependencies concern *duplicate attribute sequences*[1] that are used to link together the relation schemas in a database schema. Such attribute sequences form the so-called *logical navigation* in a database schema.

Duplicate attribute sequences are usually used as access paths to navigate in a relational database schema. When querying a relational database, users have to explicitly specify logical access paths between relation schemas [7].

Given a *workload* of SQL statements, the intuition is to say that we have to capture *communicating attributes*, i.e., attribute sequences involved in SQL join statements. The obtainment of such a workload is rather simple in recent RDBMS (cf. next Section).

Example 2. The inclusion dependency Instructor[rank] $\subseteq$ Department[dnumber] given in Example 1 cannot be deduced using the logical navigation. Indeed, it is reasonable to assume that the attributes rank and dnumber will probably *never* be used together in a join condition to relate instructors to departments. Conversely, the pairs of attributes (Instructor[ssn], TeachesIn[ssn]) and (Department[dnumber], TeachesIn[dnum]) will certainly occur in a join condition. Therefore, the inclusion dependencies TeachesIn[ssn] $\subseteq$ Instructor[ssn] and TeachesIn[dnum] $\subseteq$ Department[dnumber] could be found out.

Therefore, we formally define the logical navigation inherently available in relational databases thanks to an SQL workload. By this way, two *hard problems* are solved in the same time:

- the number of candidate dependencies is reduced drastically,
- no expert user has to be involved to distinguish interesting dependencies.

Paper organization The logical navigation is formally defined in Section 2. We point out how to discover inclusion and approximate dependencies in Section 3, and functional dependencies in Section 4. We conclude in Section 5.

Due to lack of space, we refer the reader to text books such as [9] for relational database concepts.

[1] ordered set of attributes

2 The logical navigation

We define the logical navigation w.r.t. a set of join queries on a given database. Informally, a logical navigation is a binary relation which associates an attribute sequence $R_i.\overline{X}$ with another attribute sequence $R_j.\overline{Y}$ such that $\overline{X}$ and $\overline{Y}$ appear together in a join condition.

Details of discovering a logical navigation from an operational database go beyond the scope of this paper. In this section, we only give some clues to cope with this task.

To find out the logical navigation, a simple solution is to have a workload of SQL statements which is representative of the system. In modern DBMS, such representative workload can be generated by logging activity on the server and filtering the events we want to monitor [2]. For instance, this task can be achieved using the `profiler` under MS SQL SERVER 7.0 or the `trace utilities` under ORACLE 8.

We represent uniformly the pairs of attribute sequences used in a join condition in a set called $\mathcal{Q}$. An element of $\mathcal{Q}$ is denoted by $R_i[\overline{A_1..A_k}] \bowtie R_j[\overline{B_1..B_k}]$ and indicates that each attribute $R_i[A_l]$, for $l \in [1,k]$, is compared with an attribute $R_j[B_l]$ in a join condition.

We give now a formal definition of the logical navigation.

Definition 1. *Let* R *be a relational database schema and* U *be the set of its attributes. Let* $\mathcal{Q}$ *be the set of pairs of attribute sequences extracted from a representative* SQL *workload on a database* r *over* R.
The logical navigation *of* R, *denoted by nav, is a binary relation over* $2^U \times 2^U$ *defined by:*

$$nav(R_i.\overline{X}, R_j.\overline{Y}) \overset{def}{=} \exists q \in \mathcal{Q} \ s.t. \ q = R_i[\overline{X}] \bowtie R_j[\overline{Y}]$$

nav is symmetric but neither reflexive nor transitive. Let nav^* be the reflexive transitive closure of *nav*. nav^* becomes an equivalence relation and let π_{nav^*} be the set of equivalence classes of nav^*. By this way, each equivalence class captures *duplicated attribute sequences*.

Note that computing the reflexive transitive closure of *nav* can be achieved in $O(n^3)$ (e.g. Warshall algorithm) where n is the number of attribute sequences implied in *nav*.

Example 3. Consider the database given in Table 1. From a representative workload of this database, the set $\mathcal{Q}$ should be:

$$\mathcal{Q} = \left\{ \begin{array}{c} \mathsf{Instructor[ssn]} \bowtie \mathsf{TeachesIn[ssn]}, \\ \mathsf{Department[dnumber]} \bowtie \mathsf{TeachesIn[dnum]}, \\ \mathsf{Instructor[ssn]} \bowtie \mathsf{Department[mgr]} \end{array} \right\}$$

Thus, the set of equivalence classes is:

$$\pi_{nav^*} = \left\{ \begin{array}{c} \{\ \mathsf{Instructor[ssn]},\ \mathsf{Department[mgr]},\ \mathsf{TeachesIn[ssn]}\}, \\ \{\mathsf{Department[dnumber]},\ \mathsf{TeachesIn[dnum]}\} \end{array} \right\}$$

We sketch in the two following sections how the logical navigation will give us valuable clues to discover interesting inclusion dependencies (Section 3) and interesting functional dependencies (Section 4).
In the sequel, the input parameters of these discovery tasks are a database r over a database schema R and the logical navigation π_{nav^*}.

3 Application to inclusion and approximate dependencies discovery

Binary combinations of attribute sequences of each equivalence class of π_{nav^*} will deliver approximate inclusion dependencies.

Whatever the database instance, we know exactly the number of *pair* $(R.\overline{X}, S.\overline{Y})$ from which approximate inclusion dependencies will be inferred. Indeed, each equivalence class of n elements will deliver $C_n^2 = n(n-1)/2$ candidates. The number of database accesses is bounded by the following property.

Property 1. Let l be the number of equivalence classes of π_{nav^*}. Let C_i be the ith equivalence class of π_{nav^*}, $i \in [1, l]$ and $k_i = |C_i|$. The number of pair implied by π_{nav^*} is equal to $\sum_{i=1}^{l} k_i(k_i - 1)/2$.

Database accesses Let us introduce approximate inclusion dependencies based on the error measure g_3 [5]. The idea is to count the minimal number of tuples we have to remove (from the relation of the left-hand side) to obtain a relation that satisfies the dependency. Then, the error g_3 of an approximate inclusion dependency is defined as:

$$g3(R_i[\overline{Y}] \subseteq R_j[\overline{Z}]) = 1 - \frac{max\{|s| | s \subseteq r_i \text{ and } R_i[\overline{Y}\,] \subseteq R_j[\overline{Z}\,] \text{ holds in } s \text{ and } r_j\,\}}{|r_i|}$$

In the sequel, approximate inclusion dependencies will be parameterized by their error g_3 i.e. $R_i[\overline{Y}\,] \subseteq_{g_3} R_j[\overline{Z}\,]$.

From a pair of attribute sequences, it remains to find out the direction of the associated inclusion dependency and to compute its error.
We decide to determine the direction of an approximate inclusion dependency between two attribute sequences induced by π_{nav^*} by comparing the number of their distinct values.

Let C be an equivalence class of π_{nav^*} and let $R_i.\overline{X}$ and $R_j.\overline{Y}$ be two attribute sequences of C. If $|\pi_X(r_i)| \leq |\pi_Y(r_j)|$ then the direction of the approximate inclusion dependency is from R_i to R_j.

It remains to compute the error measure g_3. In fact, the maximal subset s of the relation r_i for which the inclusion dependency $R_i[\overline{X}\,] \subseteq R_j[\overline{Y}\,]$ holds is given by the semi-join (denoted by the symbol $\ltimes$) between $\overline{X}$ and $\overline{Y}$, i.e. $g_3(R_i[\overline{X}\,] \subseteq R_j[\overline{Y}\,]) = 1 - |r_i \ltimes_{(\overline{X}=\overline{Y})} r_j\,|/|r_i\,|$.

Given a pair of attribute sequences, we can effectively determine the existence of approximate inclusion dependency by performing SQL queries against the database.

Example 4. Assume that we want to compute the error associated with the inclusion dependency $\mathsf{Department[mgr]} \subseteq_{g_3} \mathsf{Instructor[ssn]}$. From Table 1, we have: $|\mathsf{Department} \ltimes_{(mgr=ssn)} \mathsf{Instructor}| = 4$ and $|\mathsf{Department}| = 6$. Therefore, the error g_3 is equal to $1/3$.

Within this framework, an algorithm has been devised to discover approximate inclusion dependencies from π_{nav^*} in [11].

Example 5. For instance, the following approximate inclusion dependencies can be discovered:
$I = \{\mathsf{TeachesIn[ssn]} \subseteq_0 \mathsf{Instructor[ssn]}, \mathsf{TeachesIn[dnum]} \subseteq_0 \mathsf{Department[dnumber]}, \mathsf{Department[mgr]} \subseteq_{1/3} \mathsf{Instructor[ssn]}, \mathsf{Department[mgr]} \subseteq_{1/3} \mathsf{TeachesIn[ssn]}\}$

4 Application to functional dependencies discovery

Each attribute sequences of each equivalence class of π_{nav^*} will possibly deliver left-hand sides of interesting functional dependencies. As for inclusion dependencies, the following property bounds the number of candidates.

Property 2. Let l be the number of equivalence classes of π_{nav^*}. Let C_i be the ith equivalence class of π_{nav^*}, $i \in [1, l]$ and $k_i = |C_i|$. The number of left-hand sides implied by π_{nav^*} is equal to $\sum_{i=1}^{l} k_i$.

Database accesses From a given left-hand side X of a relation schema R, the candidate right-hand sides are in $R \setminus X$. Let $A \in R \setminus X$, the functional dependencies $R : X \rightarrow A$ holds in r iff $|\pi_{X \cup A}(r)| = |\pi_X(r)|$ [3].

Such tests can be easily performed with SQL queries.

Example 6. From the set π_{nav^*} given in Example 3, the following exact functional dependencies can be carried out by querying the database given in Table 1:
$\mathsf{Instructor : ssn} \longrightarrow \mathsf{status, rank}$
$\mathsf{TeachesIn : dnum} \longrightarrow \mathsf{depname}$
$\mathsf{Department : dnumber} \longrightarrow \mathsf{dname, mgr}$

5 Conclusion

The main contribution of this paper is to define rigourously the logical navigation inherently available in relational databases. From it, tractable algorithms for discovering "interesting" functional and inclusion dependencies can be devised.

In [12], we showed that such dependencies are useful to reverse engineer first normal form relational databases.

It must be clear that *only a subset* of all possible functional and inclusion dependencies is discovered from the logical navigation. In our opinion, the missing inclusion dependencies (those which cannot be deduced from the logical navigation) seem to be of little interest in database applications. However, some missing functional dependencies can be interesting: For example, the dependency TeachesIn : ssn,dnum $\longrightarrow$ year, which holds in the relation TeachesIn given in Table 1, is not revealed by the logical navigation. We are currently working on the design of efficient algorithms to discover a small cover of the functional dependencies holding in a relation .

Currently, we are working on an implementation of a tool to manage the logical navigation for aiding the database administrator both for re-organizing its database schema thanks to inclusion and functional dependencies and for inquiring data consistency from such dependencies.

References

1. S. Bell and P. Brockhausen. Discovery of Data Dependencies in Relational Databases. Technical report, LS-8 Report 14, University of Dortmund, 18p, April 1995.
2. S. Chaudhuri and V. Narasayya. Autoadmin "what-if" Index Analysis Utility. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 367–378, Seattle, June 1998.
3. Y. Huhtala, J. Karkkainen, P. Porkka, and H. Toivonen. Efficient Discovery of Functional and Approximate Dependencies Using Partitions. In *Proceedings of the* 14^{th} *IEEE International Conference on Data Engineering*, Orlando, USA, 1998.
4. M. Kantola, H. Mannila, K-J. Räihä, and H. Siirtola. Discovering Functional and Inclusion Dependencies in Relational Databases. *International Journal of Intelligent Systems*, 7(1):591–607, 1992.
5. J. Kivinen and H. Mannila. Approximate Inference of Functional Dependencies from Relations. *Theoretical Computer Science*, 149(1):129–149, 1995.
6. M. Klemettinen, H. Mannila, P. Ronkainen, H. Toivonen, and A.I. Verkamo. Finding Interesting Rules from Large Sets of Discovered Association Rules. In *Proceedings of the* 3^{td} *International Conference on Information and Knowledge Management*, pages 401–407, December 1994.
7. D. Maier, J.D. Ullman, and M. Y. Vardi. On the Foundations of the Universal Relation Model. *ACM Transaction on Database Systems*, 9(2):283–308, June 1984.
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. Heikki Mannila and Kari-Jouko Räihä. *The Design of Relational Databases.* Addison-Wesley, second edition, 1994.
10. N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal. Efficient mining of association rules using closed itemset lattices. *Information System*, 24(1):25–46, 1999.
11. J-M. Petit and F. Toumani. Discovery of inclusion and approximate dependencies in databases. In *To appear in BDA99 (french conference on DB), 20 pages*, October 1999.
12. J-M. Petit, F. Toumani, J-F. Boulicaut, and J. Kouloumdjian. Towards the Reverse Engineering of Denormalized Relational Databases. In S. Su, editor, *Proceedings of the* 12^{th} *IEEE International Conference on Data Engineering*, pages 218–227. IEEE Computer Society, New Orleans, February 1996.