# Chasing after secrets in relational databases

Joachim Biskup[1], Sven Hartmann[2], Sebastian Link[3], and Jan-Hendrik Lochner[1]

[1] Fakultät für Informatik, Technische Universität Dortmund, Germany
[2] Institut für Informatik, Technische Universität Clausthal, Germany
[3] School of Information Management, Victoria University of Wellington, New Zealand

**Abstract.** Inference control can guarantee confidentiality but is costly to implement. Access control can be implemented efficiently but cannot guarantee confidentiality. Hence, it is a natural question to ask when exactly inference control becomes necessary. We characterize the situation in which it becomes possible to infer secrets without any violation of a given access control policy. For this purpose, we establish the Chase as a tool that infers secrets from previous query answers by applying a class of equality- and tuple-generating data dependencies declared over the underlying schema. Our characterization aims to exploit new opportunities for maximizing the availability of data while confidentiality is preserved dynamically and efficiently.

## 1 Introduction

Information is a fundamental asset in today's society. The owners of information may want to discretionarily share some pieces of their private information while hiding other pieces. Inference control is a security mechanism aiming to keep information confidential, according to a confidentiality policy declared by the information owner [1, 2]. Unfortunately, however, inference control is known to be costly: we need to control the users' access to data items that represent crucial information, but additionally we also have to take into account the users' abilities to draw conclusions from the data accessed by a usage history, their application-specific a priori knowledge, and further potential background knowledge. Indeed, the overall quality of inference control depends crucially on the assumptions about the users' capabilities.

**Controlled Query Evaluation.** As a generally applicable countermeasure for preventing the gain of forbidden information, controlled query evaluation has been developed and analyzed as a policy-based, dynamic inference control mechanism for enforcing confidentiality in information systems [3–8]. The mechanism is based on maintaining a log file of previously returned query answers and a priori knowledge, and on evaluating a censor function that has to solve implications between logical sentences constructed from the log file and the actual query on the one side, and logical sentences formed from the potential secrets as declared by the confidentiality policy on the other side. Whenever the control system detects that a query answer would lead to a violation of confidentiality, the query answer is suitably distorted by refusing the answer or by lying.

In the following we only consider the refusal approach [5, 6]. For this approach, the non-refused answers are logged, and the censor checks whether the a priori knowledge

and the previous non-refused answers together with either 1) the correct query result or 2) the negated query result would logically imply a potential secret. The first check ensures that the log file never logically implies any potential secret. The second check is necessary to prevent so-called meta-inferences by the user who might reason about the actual causes of a refusal. This necessity is a well-known feature of dynamic inference control (i.e. inference control at run-time), e.g., for controlling logic-based queries [3] or statistical aggregate queries [9].

**Drawbacks and Optimizations.** Controlled query evaluation by refusals can be employed for relational databases, subject to some restrictions that ensure the decidability of all implication problems that occur during the inference control procedures [8]. Even then, the generally high computational costs at run-time remain a major drawback of inference control. However, for specific cases highly efficient optimizations are possible [10–12]. In these cases costly inference control can be reduced to an efficient form of access control, called "natural" in the following. *Natural access control* simply checks whether every constant that appears in a potential secret (an existential-*R*-sentence, i.e. a closed select-project query) also appears in a query (another existential-*R*-sentence) at the same position. Hence, natural access control does not require the maintenance of a log file, and deciding logical implications is reduced to a simple check whether the pattern of a potential secret matches that of a query.

So far, two cases have been identified in which natural access control provides an efficient mechanism that guarantees confidentiality effectively [10–12]. These cases are described in terms of the data definition language (DDL) that defines the relational schema, the confidentiality policy language (CPL) in which potential secrets can be declared, and the query language (QL) in which queries can be specified.

In both cases, regarding the DDL, only schemas with functional dependencies are considered, which are then taken as the sole a priori knowledge; and regarding the CPL and the QL, only existential-*R*-sentences are permitted. Under these restrictions alone, however, confidentiality is not guaranteed.

For the first case [10] it is necessary to confine the DDL to schemas in Boyce-Codd normal form having a unique key [15] and the CPL to potential secrets in which constants only occur for attributes of either the unique key or the unique key and one additional attribute, but for the QL any existential-*R*-sentence is permitted. In the second case [11] the DDL accepts schemas with any set of functional dependencies, and the CPL permits any set of existential-*R*-sentences, but the QL prohibits any existential quantification in queries, i.e., permits *R*-sentences (select queries) only. Moreover, in both cases appropriate examples indicate that the restrictions can essentially not be relaxed without a violation of the confidentiality requirement. This is simply the price that needs to be paid to guarantee confidentiality efficiently.

The following example exhibits a simple type of violations. Consider a relation schema *EMP(Id,Name,Salary)* where the attribute *ID* forms a key, i.e., both attributes *Name* and *Salary* are functionally dependent on *ID*. Suppose that the security officer declares the subtuple (*Steve Jobs, $1,000K*) over the attribute set {*Name,Salary*} as a potential secret. Suppose further that the database administrator permits users to submit any sequence of existential-*EMP*-sentences (closed select-project queries), in particular queries that retrieve subtuples over the attribute sets {*ID,Name*} and {*ID,Salary*},

respectively. This combination is prohibited in both cases sketched above, for good reasons as shown now. If the user queries first whether the subtuple (*0001,Steve Jobs*) over the attribute set {*ID,Name*} occurs in the database and then whether the subtuple (*0001,$1,000K*) over the attribute set {*ID,Salary*} occurs in the database, then neither query answer contains the potential secret (*Steve Jobs,$1,000K*). Therefore, natural access control would return the correct answers. Provided the subtuples do occur in the database, the user could then apply the key property to infer that the tuple (*0001,Steve Jobs,$1,000K*) must also occur in the database, and thus discover the potential secret. Hence, natural access control would fail, in contrast to inference control by controlled query evaluation, which logs the returned answers and censors the crucial inference.

**Availability.** The restrictions specified for the two cases above prevent a violation of confidentiality in a uniform way. That is, either by prohibiting the pertinent potential secret under the considered schema (first case), or by prohibiting the queries regarding subtuples (second case). These prohibitions are uniform in the sense that the exhibited violation type cannot occur for any permitted instantiation of the considered languages. However, uniformity might result in unnecessary restrictions at run time, thus disabling maximal availability of information. Consequently, important resources can potentially not be shared effectively and successful cooperations become impossible. Limited availability appears to be a general trade-off in static approaches to inference control. For example, in recent works the goal of availability is demoted by defining a query to be secret with respect to a set of views, if i) the query and the view answers are independent statistical events [14], or ii) the query can be rewritten using the views only [13].

In contrast, dynamic approaches to inference control, such as controlled query evaluation, can be tailored to maximize the availability of information. For this purpose, the confidentiality policy is treated as a declaration of exceptions to generally accessible data elements, and, at run-time, query answers are only refused when necessary, i.e., when the answer together with previous query answers would result in a violation of the confidentiality policy. In our example above, any user may either learn that the subtuple (*0001,Steve Jobs*) occurs in the database or learn that the subtuple (*0001,$1,000K*) occurs in the database, but no user must learn that both subtuples occur in the database.

**Contribution.** Accordingly, we are interested in exploring necessary and sufficient conditions for a potential violation of confidentiality in terms of a particular schema or even of a particular relation instance of a schema. Clearly, for any particular schema or instance, respectively, the exhibited violation constitutes a "forbidden structure": the user must not successfully query the connections of one key value *0001* with both the property value *Steve Jobs* and the property value *$1,000K* whenever the property value combination (*Steve Jobs,$1,000K*) is protected. For detecting the occurrence of this "forbidden structure" it would suffice to exploit the "forbidden structure" as an inference (intrusion) signature. As the main contribution of this paper we demonstrate that inference control becomes necessary precisely when there is a nontrivial template dependency *TD* [16] such that i) *TD* is implied by the data dependencies declared over the underlying schema (and assumed to be functional or full join dependencies here), ii) previous query answers result in an instantiation of all rows of the hypothesis of *TD* and iii) the potential secret is covered by the instantiated conclusion of *TD*. For our

example above, a "forbidden structure" is encoded in the template dependency

$$\frac{\begin{array}{c} a_{ID}\ a_N\ b_S \\ a_{ID}\ b_N\ a_S \end{array}}{a_{ID}\ a_N\ a_S} \quad .$$

In fact, this template dependency is implied by the two FDs *ID* $\rightarrow$ *Name* and *ID* $\rightarrow$ *Salary*, when we map $a_{ID}$ to *0001*, $a_N$ to *Steve Jobs* and $a_S$ to *$1,000K* the two query answers result in an instantiation of all rows of the hypothesis and the instantiated conclusion covers the potential secret (*Steve Jobs,$1,000K*). Hence, a violation of confidentiality occurs precisely when there is a potential secret that results from chasing [17–19], as pioneered also by A. Mendelzon, previous query answers and a non-refused answer by the declared data dependencies.

**Organization.** After introducing a formal framework (Section 2), we establish an *application scenario*, describing which actors employ which languages (Section 3). We then establish the *violation condition* and prove it to be sufficient and necessary under reasonable circumstances (Section 4). We illustrate our results by a medical example (Section 5), and comment on a new approach to inference control that is based on our results (Section 6). Finally, we conclude and comment on future work (Section 7).

## 2 Formal framework

A *relation schema* is denoted by $RS = \langle R, \mathcal{U}, \Sigma \rangle$ where $R$ is a *relation symbol*, $\mathcal{U}$ is a finite set of *attributes*, and $\Sigma$ is a finite set of dependencies (semantic constraints). $\Sigma$ consists of functional dependencies, assumed to be a minimal cover [20]; or it consists of full join dependencies; or it comprises both kinds of dependencies. An *instance* $r$ of a relation schema is a finite dependency-satisfying Herbrand interpretation of the schema, considering the relation symbol as a predicate. A *tuple* is denoted by $\mu = R(a_1, \ldots, a_n)$ where $n = |\mathcal{U}|$ and $a_i \in Const$, an infinite set of constants. As further discussed in [8], the infinity assumption avoids inferences based on the combinatorial effects of a fixed finite domain. If $\mu$ is an element of $r$, we write $r \models_M \mu$. More generally, $\models_M$ denotes the *satisfaction* relation between an interpretation and a sentence. The corresponding notion of logical *implication* (entailment) between sentences is denoted by $\models$.

Let $\mathcal{A}, \mathcal{B} \subseteq \mathcal{U}$ be attribute sets. A relation $r$ over $\mathcal{U}$ satisfies the *functional dependency (FD)* $\mathcal{A} \rightarrow \mathcal{B}$ if any two tuples that agree on the values of attributes in $\mathcal{A}$ also agree on the values of the attributes in $\mathcal{B}$. $\mathcal{A} \rightarrow \mathcal{B}$ is called *trivial* if $\mathcal{B} \subseteq \mathcal{A}$. $\mathcal{A} \subseteq \mathcal{U}$ is a *super key* of $RS$ if $\Sigma \models \mathcal{A} \rightarrow \mathcal{U}$. A *key* is a minimal super key. $RS$ is in *Boyce-Codd normal form (BCNF)* if for every nontrivial FD $\mathcal{A} \rightarrow \mathcal{B}$, logically implied by $\Sigma$, $\mathcal{A}$ is a super key of $RS$.

Let $\mathcal{C}_1, \ldots, \mathcal{C}_l \subseteq \mathcal{U}$ be attribute sets (used as hypothesis) such that $\mathcal{C}_1 \cup \ldots \cup \mathcal{C}_l = \mathcal{U}$ and $\mathcal{C}_{l+1}$ an attribute set (used as conclusion) with $\mathcal{C}_{l+1} \subseteq \mathcal{U}$. A relation $r$ over $\mathcal{U}$ satisfies the *embedded join dependency (EJD)* $\bowtie[\mathcal{C}_1, \ldots, \mathcal{C}_l | \mathcal{C}_{l+1}]$ if whenever there are tuples $\mu_1, \ldots, \mu_l$ (not necessarily different) in $r$ with $\mu_i[\mathcal{C}_i \cap \mathcal{C}_j] = \mu_j[\mathcal{C}_i \cap \mathcal{C}_j]$ for $1 \leq i, j \leq l$, there is also a tuple $\mu_{l+1}$ in $r$ with $\mu_{l+1}[\mathcal{C}_i \cap \mathcal{C}_{l+1}] = \mu_i[\mathcal{C}_i \cap \mathcal{C}_{l+1}]$ for $1 \leq i \leq l$ (cf. [20]). If the conclusion covers all declared attributes, i.e., $\mathcal{C}_{l+1} = \mathcal{U}$, then we have a *full join dependency*, denoted by $\bowtie[\mathcal{C}_1, \ldots, \mathcal{C}_l]$.

We express *queries* in a fragment of the relational calculus. Let $Var$ be a set of variables. The *query language* $\mathscr{L}_Q^c$ is the set of all *closed* formulas (sentences) of the form $(\exists X_1)\dots(\exists X_l)R(v_1,\dots,v_n)$ with $0 \le l \le n$, $X_i \in Var$, $v_i \in Const \cup Var$, $\{X_1,\dots,X_l\} \subseteq \{v_1,\dots,v_n\}$, and $v_i \neq v_j$ if $v_i, v_j \in Var$ and $i \neq j$; these properties and the closedness imply that $\{X_1,\dots,X_l\} = \{v_1,\dots,v_n\} \cap Var$. We refer to such formulas as *existential-R-sentences*, or *closed select-project queries*. For $\Phi \in \mathscr{L}_Q^c$, we define the *scheme* of $\Phi$ as the set of attributes for which a constant appears; the set of the remaining attributes, i.e., $\mathcal{U} \setminus scheme$, is called the *bound* part.

Let $\Phi \in \mathscr{L}_Q^c$ be a query and $r$ an instance. The *ordinary evaluation* of $\Phi$ on $r$ is defined by

$$eval^*(\Phi)(r) := \texttt{if } r \models_M \Phi \texttt{ then } \Phi \texttt{ else } \neg\Phi. \tag{1}$$

For a sentence in $\mathscr{L}_Q^c$ let its corresponding "generalized tuple" denote the sentence without its prefix of existential quantifiers. In this case we think of the variables in the generalized tuple as the null value "exists but unknown".

We employ *controlled query evaluation (CQE)*, developed in [3–8], briefly outlined as follows. A *potential secret* $\Psi$ is a sentence in the language $\mathscr{L}_Q^c$. If $r \not\models_M \Psi$ for a database instance $r$, the database user may learn that $\Psi$ is false in the instance; however, if $r \models_M \Psi$, it has to be kept secret that $\Psi$ is actually true. The set $pot\_sec \subseteq \mathscr{L}_Q^c$ denotes a *confidentiality policy* being known to the database user. The *a priori user knowledge* $log_0$, with $\Sigma \subseteq log_0 \subseteq \mathscr{L}_Q^c \cup \Sigma$, $r \models_M log_0$, and $log_0 \not\models \Psi$ for every $\Psi \in pot\_sec$, has the following properties: it consists of the declared constraints; it is true in the actual instance; and none of the actual truth values of the potential secrets is known to the user in advance. For the purpose of this paper we assume that $log_0 = \Sigma$. A query sequence is given by $Q = \langle \Phi_1, \Phi_2, \dots \rangle$ with $\Phi_i \in \mathscr{L}_Q^c$. The CQE for known potential secrets enforced by *refusal* is defined by $cqe^R(Q, log_0)(r, pot\_sec) := \langle (ans_1, log_1), (ans_2, log_2), \dots \rangle$. The values of the returned answers $ans_i$ and the representation of the current user knowledge $log_i$ are determined by a *censor function*:

$censor^R(pot\_sec, log, \Phi) := (exists\Psi)[\Psi \in pot\_sec$ and
$(log \cup \{\Phi\} \models \Psi$ or $log \cup \{\neg\Phi\} \models \Psi)]$

$$\begin{aligned} ans_i := {} & \texttt{if } log_{i-1} \models eval^*(\Phi_i)(r) \texttt{ then } eval^*(\Phi_i)(r) \\ & \texttt{else if } censor^R(pot\_sec, log_{i-1}, \Phi_i) \\ & \texttt{then mum else } eval^*(\Phi_i)(r) \end{aligned}$$

$$\begin{aligned} log_i := {} & \texttt{if } censor^R(pot\_sec, log_{i-1}, \Phi_i) \texttt{ then } log_{i-1} \\ & \texttt{else } log_{i-1} \cup \{ans_i\} \end{aligned}$$

The CQE by refusal $cqe^R$ is secure for all possible query sequences and confidentiality policies in the sense of the following definition (see [5, 7]).

**Definition 1.** *A CQE $cqe$ is* secure for $Q$ and $pot\_sec$ *if for every finite prefix $Q'$ of $Q$ the following holds: For every $\Psi \in pot\_sec$, for every instance $r_1$, and for every $log_0$ with $r_1 \models_M log_0$ and $log_0 \not\models \chi$ for every $\chi \in pot\_sec$, there exists an appropriate instance $r_2$ with $r_2 \models_M log_0$ such that:*
*1) $cqe(Q', log_0)(r_1, pot\_sec) = cqe(Q', log_0)(r_2, pot\_sec)$;*
*2) $eval^*(\Psi)(r_2) = \neg\Psi$.*

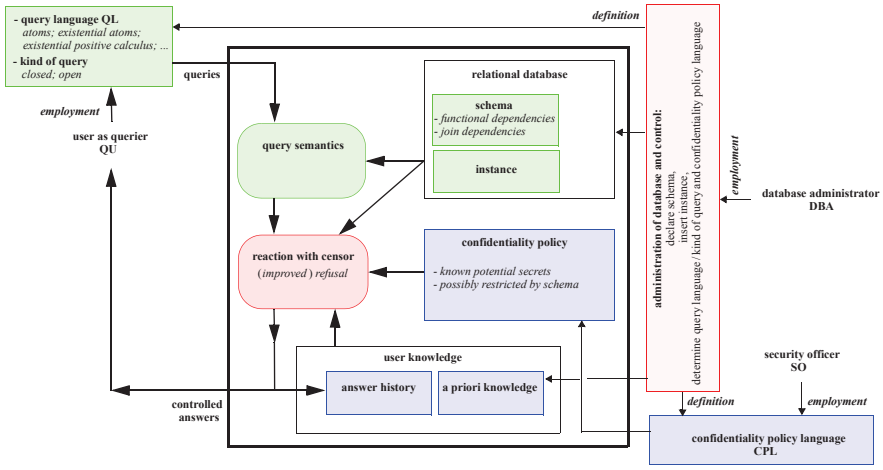


**Fig. 1.** Application scenario with roles of agents and system architecture

## 3 Application scenario

Roughly sketched, in any specific application, firstly, a database administrator configures and initializes the system. Secondly, a security officer deals with permissions and prohibitions. Thirdly and finally, users actually employ the system by issuing queries. In this paper, we assume that the database is not subject to any updates. Thus, there are three (roles of) agents DBA, SO, and QU, who exploit some formal languages DDL, CPL, and QL, respectively, as outlined in the following (and exemplified by formal definitions in the preceding section).

The *database administrator*, DBA, acts as follows. (1) The DDL (data definition language) is employed for declaring a (relation) schema $\langle R, \mathcal{U}, \Sigma \rangle$ which includes semantic constraints as functional dependencies and full join dependencies. (2) An instance $r$ is generated, which complies with the declared schema, by inserting appropriate data (assumed to be fixed since later updates are not considered). (3) An access control method is installed. (4) The CPL (confidentiality policy language) is defined, which provides the formal means to express prohibitions in terms of sentences to be kept confidential (while we assume permissions by default). (5) The QL (query language) is defined, which provides the formal means to express queries in terms of the schema referring to the instance.

The *security officer*, SO, employs the CPL, as previously defined by the DBA, for declaring a confidentiality policy $pot\_sec$, which is specific for a particular user (or group of collaborating users).

The user, acting as a *querier*, QU, employs the QL, as previously defined by the DBA, for stepwise submitting a query sequence $Q = \langle \Phi_1, \Phi_2, \ldots, \Phi_i, \ldots \rangle$.

The controlled relational database management system consists of the traditional functional part and the control part. The functional part maintains an instance $r$ according to the schema $\langle R, \mathcal{U}, \Sigma \rangle$ and evaluates the submitted queries $\Phi_i$. The control part

applies a censor function to the query answers and reacts appropriately based on the declared confidentiality policy $pot\_sec$ and the current user knowledge $log$. The overall situation is illustrated by Figure 1.

## 4 The violation condition

In the context of controlled query evaluation a violation of the confidentiality policy occurs whenever instance data, returned as part of query answers, logically imply a potential secret under the a priori knowledge of the declared data dependencies. If we completely understand the circumstances under which such implications occur, then we might be able to maximize the availability of data while confidentiality is still guaranteed. Consequently, it is our goal to characterize, in terms of the declared data dependencies, when natural access control fails to observe violations of the confidentiality policy, i.e., when inference control becomes necessary. This characterization will be established in Theorems 1 and 2.

As an expressive class of equality- and tuple-generating data dependencies that are frequently declared on relational schemas in practice we consider functional and arbitrary full join dependencies. The following example points us to our anticipated characterization. It illustrates how potential secrets can be chased by applying the declared data dependencies to previous query answers.

*Example 1.* Consider a relation schema $RS = \langle R, \mathcal{U}, \Sigma \rangle$ with attribute set $\mathcal{U} = NLK_1K_2AB$ and constraints

$$\Sigma = \{N \rightarrow K_1, L \rightarrow K_2, \bowtie[K_1K_2A, K_1K_2NLB], \bowtie[K_1K_2B, K_1K_2NLA]\}$$

consisting of functional and full join dependencies. Furthermore, let the instance $r$ contain just one tuple denoted by $R(n, l, k_1, k_2, a, b)$ and

$$\Psi = (\exists X_N)(\exists X_L)(\exists X_{K_1})(\exists X_{K_2})R(X_N, X_L, X_{K_1}, X_{K_2}, a, b)$$

the sole potential secret. Suppose that the user issues the following queries:
$\Phi_1 = (\exists X_L)(\exists X_{K_2})(\exists X_A)R(n, X_L, k_1, X_{K_2}, X_A, b)$,
$\Phi_2 = (\exists X_L)(\exists X_{K_1})(\exists X_B)R(n, X_L, X_{K_1}, k_2, a, X_B)$,
$\Phi_3 = (\exists X_N)(\exists X_{K_1})(\exists X_B)R(X_N, l, X_{K_1}, k_2, a, X_B)$,
$\Phi_4 = (\exists X_N)(\exists X_{K_2})(\exists X_A)R(X_N, l, k_1, X_{K_2}, X_A, b)$.

Under natural access control all queries can be answered correctly as none of the answers contains the potential secret. Therefore, the user knowledge is $log = \Sigma \cup \{\Phi_1, \Phi_2, \Phi_3, \Phi_4\}$. Then, the user could infer the following equalities and sentences by chasing and substitutions, respectively:
apply $N \rightarrow K_1$ to $\Phi_1$ and $\Phi_2$: $k_1 = X_{K_1}$;
apply $k_1 = X_{K_1}$ to $\Phi_2$: $\Phi_5 = (\exists X_L)(\exists X_B)R(n, X_L, k_1, k_2, a, X_B)$;
apply $L \rightarrow K_2$ to $\Phi_3$ and $\Phi_4$: $k_2 = X_{K_2}$;
apply $k_2 = X_{K_2}$ to $\Phi_4$: $\Phi_6 = (\exists X_N)(\exists X_A)R(X_N, l, k_1, k_2, X_A, b)$;
apply $\bowtie[K_1K_2A, K_1K_2LNB]$ to $\Phi_5$ and $\Phi_6$: $\Phi_7 = (\exists X_N)R(X_N, l, k_1, k_2, a, b)$.

As a result we have $log \models \Psi$, i.e. the user can infer $\Psi$, resulting in a violation of confidentiality. Using JD-chasing, we observe that $\Sigma$ implies the embedded join dependency $\varphi = \bowtie[NK_1B, NK_2A, LK_2A, LK_1B|LK_1K_2AB]$, where the hypothesis captures the respective schemes of the four queries $\Phi_1$, $\Phi_2$, $\Phi_3$, and $\Phi_4$, and the conclusion corresponds to the scheme of the inferred sentence $\Phi_7$ and thereby covers the scheme of the potential secret.

Note that the hypothesis of the embedded join dependency $\varphi$ of the previous example must never be fully instantiated by constants that occur in query answers. The reason is that such an instantiation would cover the potential secret in the conclusion of $\varphi$. Since $\varphi$ is implied by $\Sigma$, this would mean that the potential secret can be inferred from the query answers and would therefore result in a violation of the confidentiality policy. Vice versa, an inference of a potential secret from a set of query answers also entails that there is some dependency that is implied by the constraint set $\Sigma$ and that encodes the instantiations that can lead to this inference. As it turns out, the class of template dependencies [16, 20, 21] can be successfully utilized to encode the forbidden structures, and characterize the situations under which natural access control can be bypassed and, therefore, inference control becomes a necessity.

A *template dependency* over $\mathcal{U}$ is represented by one or more rows $h_1, \ldots, h_l$, called *hypothesis rows*, or *hypotheses*, and a *conclusion row* $c$, or *conclusion*, below a line. Each row consists of abstract symbols, one symbol per attribute in $\mathcal{U}$. A symbol may appear more than once, but not in columns that correspond to different attributes, and thus a symbol is typed. We usually denote a template dependency by $TD[h_1, \ldots, h_l|c]$.

Let $t$ and $t'$ denote tuples (or rows) over $\mathcal{U}$. We define the *agree set* $ag(t, t')$ as the set of attributes where the tuples (rows) $t$ and $t'$ agree; that is, $ag(t, t') = \{A \mid A \in \mathcal{U} \text{ and } t(A) = t'(A)\}$. A relation $r$ over $\mathcal{U}$ *satisfies* the template dependency $TD[h_1, \ldots, h_l|c]$ if whenever $l$ tuples $t_1, \ldots, t_l$ can be found in $r$ such that for all $i, j \in \{1, \ldots, l\}$ we have $ag(h_i, h_j) \subseteq ag(t_i, t_j)$, then $r$ has a tuple $t$ such that for all $i = 1, \ldots, l$ we have $ag(c, h_i) \subseteq ag(t, t_i)$.

Let $c'$ be a row over $\mathcal{U}$ and $\{t_1, \ldots, t_n\}$ a set of rows over $\mathcal{U}$. Let $p$ be a mapping that maps a symbol in $c'$ either to itself or to a symbol that does not appear in $t_1, \ldots, t_n$. We extend $p$ to map rows in the usual way. We say $c = p(c')$ is a *weakening of* $c'$ *in the context of* $\{t_1, \ldots, t_n\}$.

A template dependency $TD[h_1, \ldots, h_l|c]$ is *trivial* if the conclusion $c$ can be obtained by weakening of one of its hypotheses $h_1, \ldots, h_l$ in the context of $\{h_1, \ldots, h_l\}$.

The following theorems (we omit proofs due to space restrictions) characterize when inference control becomes necessary in order to guarantee confidentiality. Informally summarized, the characterization expresses that any violating implication with instance data that cannot be detected by natural access control has a formal proof that we can obtain by applying a single template dependency which is implied by the declared dependencies. Thus, for any specific situation, a full analysis of the implications regarding the constraints declared over the schema will provide us with a full understanding of the violating implications regarding the actual instance.

Note that for template dependencies a sound and complete set of inference rules exists and *chasing* can be utilized as a proof procedure to recognize implications (see Theorem 1 and Lemma 3 of [16], respectively, and [21]). To establish our characteri-

zation we apply this method to the case of functional and full join dependencies. We conjecture that, under the provision of taking care of potentially nonterminating constructions, we can generalize our results to functional and template dependencies. We note, however, that in practice it is unlikely that a database user is sufficiently knowledgeable to apply inferences involving template dependencies.

**Theorem 1 (forbidden structures, sufficient for a violation).** *Let $RS = \langle R, \mathcal{U}, \Sigma \rangle$ be a relation schema where the dependency set $\Sigma$ consists of functional and full join dependencies. Let $TD[h_1, \ldots, h_l|c]$ be a nontrivial template dependency implied by $\Sigma$.*

*Then there are an instance $r$ of $RS$, queries $\Phi_1, \ldots, \Phi_l \in \mathscr{L}_Q^c$ with schemes $\mathcal{F}_i$ where $\cup_{j \in \{1,\ldots,l\} \setminus \{i\}} ag(h_i, h_j) \subseteq \mathcal{F}_i$, and a potential secret $\Psi \in \mathscr{L}_Q^c$ with scheme $\mathcal{P} = \cup_{j=1}^{l} ag(h_j, c)$ such that the following properties hold:*

1. *$\Phi_i \not\models \Psi$, for all $i = 1, \ldots, l$, i.e., all queries are permitted under natural access control (there is a constant in $\Psi$ that does not appear at the same position in $\Phi_i$);*
2. *$r \models_{\mathrm{M}} \Phi_i$, for all $i = 1, \ldots, l$, i.e., all queries are true in the instance $r$;*
3. *$\Sigma \cup \{\Phi_1, \ldots, \Phi_l\} \models \Psi$, i.e., the answers violate the confidentiality policy.*

**Theorem 2 (forbidden structures, necessary for a violation).** *Let $RS = \langle R, \mathcal{U}, \Sigma \rangle$ be a relation schema where the dependency set $\Sigma$ consists of functional and full join dependencies, and r an instance of $RS$. Let $\Psi \in \mathscr{L}_Q^c$ be a potential secret with scheme $\mathcal{P} \subseteq \mathcal{U}$, and $\Phi_1, \ldots, \Phi_l \in \mathscr{L}_Q^c$ queries with schemes $\mathcal{F}_1, \ldots, \mathcal{F}_l$ such that the following properties hold:*

1. *$\Phi_i \not\models \Psi$, for all $i = 1, \ldots, l$, i.e., all queries are permitted under natural access control (there is a constant in $\Psi$ that does not appear at the same position in $\Phi_i$);*
2. *$r \models_{\mathrm{M}} \Phi_i$, for all $i = 1, \ldots, l$, i.e., all queries are true in the instance $r$;*
3. *$\Sigma \cup \{\Phi_1, \ldots, \Phi_l\} \models \Psi$, i.e., the answers violate the confidentiality policy.*

*Then there exists a nontrivial template dependency $TD[h_1, \ldots, h_l|c]$ implied by $\Sigma$ such that $\mathcal{P} = \cup_{j=1}^{l} ag(h_j, c)$ and $\cup_{j \in \{1,\ldots,l\} \setminus \{i\}} ag(h_i, h_j) \subseteq \mathcal{F}_i$ holds for all $i = 1, \ldots, l$.*

Basically, we obtain the (omitted) proofs by appropriately combining two known equivalences from logic, one observation on join dependencies and the known completeness result on chasing mentioned above. We briefly outline this in the following. We have to inspect implications of the form $\Sigma \cup \{\Phi_1, \ldots, \Phi_l\} \models \Psi$, where the $\Phi_i$'s are answers to queries in $\mathscr{L}_Q^c$, and $\Psi$ is a potential secret in $\mathscr{L}_Q^c$. By the Deduction Theorem (see, e.g., [22]), such an implication is equivalent to the implication $\Sigma \models \Phi_1 \wedge \ldots \wedge \Phi_l \Rightarrow \Psi$. A full join dependency, and more generally each template dependency, can be seen as a first-order sentence where all terms (variables) are typed in the sense that a term occurs in only one attribute position. In particular, any single equality requirement expressed by that sentence deals with the values of tuples for one attribute. Now suppose that in the sentences on the right hand side of the latter implication, $\Phi_1 \wedge \ldots \wedge \Phi_l \Rightarrow \Psi$, some constant $c$ occurs in two or more attribute positions. Then the equality requirements expressed by these occurrences do not affect the status of being implied by $\Sigma$. Accordingly, we can think of $c$ being split into suitably many

variants $c_A$, one for each attribute $A$ where $c$ occurs. And thus we can treat all constants as if they were typed. The sentence on the right hand side of the implication considered is composed by elements of $\mathscr{L}_Q^c$. Accordingly, each of these elements is obtained from an atomic formula that is built from constants, treated as if they were typed, and variables, such that all variables are existentially quantified, assumed to be pairwise different throughout the whole sentence (and thus to be typed as well). We might then replace all occurrences of some subset of the constants, say the constants $c_1, \ldots, c_d$ by new and pairwise different variables $x_1, \ldots, x_d$, and then take the universal closure. The transformed sentence has the form $(\forall x_1) \ldots (\forall x_d)[\Phi_1[c_i/x_i] \wedge \ldots \wedge \Phi_l[c_i/x_i] \Rightarrow \Psi[c_i/x_i]]$ Then, by the Theorem on Constants (see, e.g., [22]), the validity of the original sentence is equivalent to the validity of the transformed sentence. The transformed sentence can be seen as a template dependency, where the required typed equalities are expressed by multiple occurrences of the new variables. Then, by Lemma 3 of [16] (Completeness of Chasing) $\Sigma \models (\forall x_1) \ldots (\forall x_d)[\Phi_1[c_i/x_i] \wedge \ldots \wedge \Phi_l[c_i/x_i] \Rightarrow \Psi[c_i/x_i]]$ means that $\Sigma$-chasing of the antecedent produces a sentence $\Psi'$ such that $\Psi$ is a weakening of $\Psi'$.

## 5 An Example

Consider the relation schemas $\langle R_1, \mathcal{U}_1, \Sigma_1 \rangle$ and $\langle R_2, \mathcal{U}_2, \Sigma_2 \rangle$ over

$$\mathcal{U}_1 = \{S(ymptom), M(ethod\_of\_Examination)\},$$
$$\mathcal{U}_2 = \{S, D(iagnosis), P(atient)\}.$$

General practitioners (GPs) will get to see the view $V$ which is defined by the SQL query `SELECT * FROM R1,R2 WHERE R1.S = R2.S`. On the view $V$, the JDs $\bowtie [SM, SDP]$ and $\bowtie [MD, MSP]$ hold. Due to the patients' privacy, GPs are only allowed to see the diagnosis for their own patients. For this example we assume that the SO declares $\Psi = (\exists X_s)(\exists X_m)V(X_s, X_m, Cancer, Smith)$ as a potential secret for some GPs. However, GPs may ask any other queries in our language, in particular

$$\Phi_1 = (\exists X_m)(\exists X_p)V(Fever, X_m, Cancer, X_p),$$
$$\Phi_2 = (\exists X_d)V(Fever, Xray, X_d, Smith)$$

with schemes $F_1 = \{S, D\}$ and $F_2 = \{S, M, P\}$, respectively.

Both queries can be asked individually, without revealing the potential secret $\Psi$. However, $\Psi$ can still be inferred: chasing the abstract tuples $V(Fever, X_m, Cancer, X_p)$ and $V(Fever, Xray, X_d, Smith)$ using $\bowtie [SM, SDP]$ leads to the tuples $V(Fever, X_m, X_d, Smith)$ and $V(Fever, Xray, Cancer, X_p)$. Chasing the tuples $V(Fever, Xray, X_d, Smith)$ and $V(Fever, Xray, Cancer, X_p)$ with $\bowtie [MD, MSP]$ leads to the tuple $V(Fever, Xray, Cancer, Smith)$ that reveals $\Psi$.

Our crucial observation is that such a data-dependent derivation of prohibited information can already be anticipated from the view declaration. In this example, on the schema level, the two JDs imply the nontrivial template dependency

$$\begin{array}{cccc} a_S & b_1 & a_D & b_2 \\ \underline{a_S} & \underline{a_E} & \underline{b_3} & \underline{a_P} \\ a_S & a_E & a_D & a_P \end{array} \quad ,$$

and on the instance level, when this template dependency is instantiated with the constants appearing in the query answers to $\Phi_1$ and $\Phi_2$, then the potential secret $(Cancer, Smith)$ appears in the conclusion.

## 6 An Application

For detecting the occurrence of a "forbidden structure" we do not need the full mechanism of a general log file and a general censor. Rather it suffices to exploit the "forbidden structure" as an inference (intrusion) signature, similarly to the well-known approach to intrusion detection.

At *declaration time*, the schema under consideration, possibly together with the particular instance, and the particular potential secrets, are examined to identify all "forbidden structures" and to compile them into inference signatures.

At *run time*, the user behavior is monitored whether a compiled "forbidden structure" is arising; and at the latest step, just before a violating condition is met, the system refuses the answer, possibly complemented by further alarm actions.

To analyze the proposed non-uniform case-specific inference detection further we can utilize our violation condition which is necessary and sufficient for an occurrence of a forbidden structure. In fact, we could either compile its occurrences into signatures, or recognize that no violations can ever occur. In particular, we will design the compilation process and the resulting inference detection procedures in future work.

## 7 Conclusion and Future Work

In principle, dynamic inference control ensures high flexibility regarding query expressiveness and maximal theoretical availability of queried information, utilizing a "last-minute" distortion strategy.

In practice, however, this may result in (too) high computational costs and thus endanger the practical availability of information. Consequently, the requirement to preserve confidentiality results in a trade-off between theoretical availability and efficient query answering, leading to practical availability.

If we favor efficiency and practical availability, then we must impose further *uniform* restrictions on the languages to express queries and confidentiality policies. If we favor flexibility and theoretical availability, we must understand when inference control becomes necessary. We have presented a characterization of this situation that may help us to establish a framework in the future that guarantees inference-proof query answers with affordable computational costs and maximal availability of data.

Recently, tuple generating dependencies and chasing have been used for studying schema mappings in data exchange, see e.g. [23, 24]. For those applications, however, the main interest is to obtain an assurance for all possible instances of the schemas involved, whereas our primary goal was to find a characterization for an individual inference situation. Despite this difference, a common topic is composability of mappings and inferences, respectively, an issue already considered in Section 8 of [21].

## References


1. Denning, D.E.: Cryptography and Data Security. Addison-Wesley (1983)
2. Biskup, J.: Security in Computing Systems – Challenges, Approaches and Solutions. Springer (2009)
3. Sicherman, G.L., de Jonge, W., van de Riet, R.P.: Answering queries without revealing secrets. ACM Trans. Database Syst. **8**(1) (1983) 41–59
4. Bonatti, P.A., Kraus, S., Subrahmanian, V.S.: Foundations of secure deductive databases. IEEE Trans. Knowl. Data Eng. **7**(3) (1995) 406–422
5. Biskup, J., Bonatti, P.A.: Lying versus refusal for known potential secrets. Data Knowl. Eng. **38** (2001) 199–222
6. Biskup, J., Bonatti, P.A.: Controlled query evaluation for enforcing confidentiality in complete information systems. Int. J. Inf. Sec. **3**(1) (2004) 14–27
7. Biskup, J., Bonatti, P.A.: Controlled query evaluation for known policies by combining lying and refusal. Ann. Math. Artif. Intell. **40** (2004) 37–62
8. Biskup, J., Bonatti, P.A.: Controlled query evaluation with open queries for a decidable relational submodel. Ann. Math. Artif. Intell. **50** (2007) 39–77
9. Kenthapadi, K., Mishra, N., Nissim, K.: Simulatable auditing. In: Proc. PODS 2005, ACM (2005) 118–127
10. Biskup, J., Embley, D.W., Lochner, J.H.: Reducing inference control to access control for normalized database schemas. Inf. Process. Lett. **106**(1) (2008) 8–12
11. Biskup, J., Lochner, J.H.: Enforcing confidentiality in relational databases by reducing inference control to access control. In: Proc. ISC 2007. Volume 4779 of LNCS., Springer (2007) 407–422
12. Biskup, J., Lochner, J.H., Sonntag, S.: Optimization of the controlled evaluation of closed relational queries. In: Proc. IFIP/SEC 2009. Volume 297 of IFIP Series., Springer (2009) 214–225
13. Zhang, Z., Mendelzon, A.O.: Authorization views and conditional query containment. In: Proc. ICDT 2005. Volume 3363 of LNCS. (2005) 259–273
14. Miklau, G., Suciu, D.: A formal analysis of information disclosure in data exchange. J. Computer and System Sciences **73** (2007) 507–534
15. Biskup, J.: Boyce-Codd normal form and object normal forms. Inf. Process. Lett. **32**(1) (1989) 29–33
16. Sadri, F., Ullman, J.: Template dependencies: a large class of dependencies in relational databases and its complete axiomatization. J. ACM **29**(2) (1982) 363–372
17. Aho, A.V., Beeri, C., Ullman, J.D.: The theory of joins in relational databases. ACM Trans. Database Syst. **4**(3) (1979) 297–314
18. Deutsch, A., Nash, A., Remmel, J.B.: The Chase revisited. In: PODS 2008. (2008) 149–158
19. Maier, D., Mendelzon, A., Sagiv, Y.: Testing implications of data dependencies. ACM Trans. Database Syst. **4**(4) (1979) 455–469
20. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley (1995)
21. Fagin, R., Maier, D., Ullman, J.D., Yannakakis, M.: Tools for template dependencies. SIAM J. on Computing **12**(1) (1983) 36–58
22. Nerode, A., Shore, R.: Logic for Applications. 2nd edn. Springer (1997)
23. Kolaitis, P.G.: Schema mappings, data exchange, and metadata management. In: Proc. PODS 2005, ACM (2005) 61–75
24. Fagin, R., Kolaitis, P.G., Popa, L., Tan, W.C.: Composing schema mappings: second-order dependencies to the rescue. ACM Trans. on Database Systems **30**(4) (2005) 994–1055