HAL
open science

Efficient Inference Control for Open Relational Queries

Joachim Biskup, Sven Hartmann, Sebastian Link, Jan-Hendrik Lochner

To cite this version:

Joachim Biskup, Sven Hartmann, Sebastian Link, Jan-Hendrik Lochner. Efficient Inference Control for Open Relational Queries. 24th Annual IFIP WG 11.3 Working Conference on Data and Applications Security and Privacy (DBSEC), Jun 2010, Rome, Italy. pp.162-176, 10.1007/978-3-642-13739-6_11 . hal-01056690

HAL Id: hal-01056690

https://inria.hal.science/hal-01056690v1

Submitted on 20 Aug 2014

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution 4.0 International License

Efficient Inference Control for Open Relational Queries

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

[1] Fakultät für Informatik, TU Dortmund, D-44221 Dortmund, Germany
{biskup|lochner}@ls6.cs.tu-dortmund.de
[2] Institut für Informatik, Technische Universität Clausthal, Germany
sven.hartmann@tu-clausthal.de
[3] School of Information Management, Victoria University of Wellington, New Zealand
sebastian.link@vuw.ac.nz

Abstract. We present a control mechanism for preserving confidentiality in relational databases under open queries. This mechanism is based on a reduction of costly inference control to efficient access control that has recently been developed for closed database queries. Our approach guarantees that secrets being declared in form of a confidentiality policy are not disclosed to database users even if they utilize their a priori knowledge to draw inferences. It turns out that there is no straightforward transition from the approach for closed queries to open queries. We show, however, that hiding the confidentiality policy from database users is sufficient to preserve confidentiality. Moreover, we propose an algorithmic implementation of the control mechanism.

1 Introduction

In our modern information society, individuals disseminate personal information over various channels. In the sense of informational self-determination, one should be able to freely decide which information to reveal, but in fact it is hardly possible to foresee all consequences of a revealed piece of information. Thus it seems more appropriate for an individual to declare which information should *not* be disclosed to other individuals.

In the context of relational databases (potentially carrying lots of personal information), the goal of *confidentiality preservation* can be enforced by suitable mechanisms based on confidentiality policies declaring the information that should not be disclosed to other database users. Besides confidentiality, however, availability of information is another important security goal: a database can only be productively employed by a user if it delivers all information needed to complete the user's task. This apparently leads to a tradeoff between confidentiality and availability.

Another aspect to be pointed out is the notion of information. Where *data* are merely uninterpreted (not application-oriented) constants, *information* is usually gained by adding semantics to data. For instance, data from a relational

database can be combined with semantic constraints in order to deduce information. Consider a relational database system maintaining the account data of a bank; with the semantic constraint that account numbers are unique within the bank, a database user learns the balance of an account holder when first asking for the account number of this client and then asking for the balance of the account with the number returned. Here, the combination of account holder and balance is deduced information from the two query results and the semantic constraint.

Controlled Query Evaluation (CQE) is an effective inference control mechanism for protecting information as declared by a suitable confidentiality policy in logic-oriented information systems. Such a policy consists of logical sentences, called secrets, which the user must not know if they are true in the actual database instance. For closed database queries, CQE checks whether the true answer (or, in some cases, also the negated answer) to a query together with the a priori knowledge of the querying user allows for the disclosure of information being protected by the policy; if so, the answer is modified, either by lying (i. e., returning the negated answer) or by refusing the answer, or by a combination of both. CQE primarily aims at preserving confidentiality of the declared secrets but also ensures availability of information when confidentiality is guaranteed.

Regarding efficiency, CQE suffers from two problems. First, it relies on the implication decision in first-order logic (being undecidable in general): each decision whether a (closed) query may be answered correctly corresponds to the decision whether a set of logical sentences implies a secret. Second, CQE has to maintain a growing log file of the assumed user knowledge. This leads to high time and space complexity, respectively. To overcome these drawbacks, a static form of CQE has been developed reducing the expensive implication decision to a pattern matching problem and abandoning the log file while keeping up confidentiality preservation.

Being originally developed for closed database queries, in this work we extend static CQE to open queries. A closed query does not contain free variables and can thus be answered by either *true* or *false*. In contrast, an open query contains free variables and the evaluation is the set of variable substitutions making the query true in the database instance. Considering open queries is an important step in enhancing our query language since most practical database queries are open ones. Being confined to closed queries usually requires that a database user already has certain knowledge about the content of the database whereas open queries provide a higher degree of freedom in terms of expressiveness: Consider a database that maintains the names of the account holders of a bank and their balances. A user being confined to closed queries can only determine the balance of an account holder by asking for different balances until the correct value has been guessed. With open queries, however, this balance information can be retrieved in one simple step.

After an overview of related work in Sect. 2, in Sect. 3 we recall some database concepts and sketch previous results for static CQE for closed relational database queries. In Sect. 4 we show that these results cannot be extended straightfor-

wardly to open queries; we propose a new control mechanism for open queries and prove it confidentiality preserving. In Sect. **??** we develop an algorithm for this control mechanism. Sect. **??** concludes the paper and gives perspectives for future research.

2 Related Work

Early approaches to security in relational databases mainly focused on discretionary access control (DAC), either by granting privileges to database users with data annotated by the respective access rights (see, e. g., [**?**]), or by modifying user queries in order to enforce a discretionarily declared security policy (see, e. g., [**?**]). Later, the concept of mandatory access control (MAC) was developed and deployed in various approaches. Instead of attaching access control information directly to the data (as DAC does), in MAC system-wide security policies are enforced on the basis of security models (see, e. g., [**?**]). (Relational) databases implementing MAC are also called "multilevel secure" (MLS) and make use of techniques like polyinstantiation; see, e. g., [**?**,**?**,**?**,**?**,**?**]. Moreover, e. g. in [**?**,**?**,**?**], comprehensive systems have been proposed that integrate DAC and/or MAC into the different stages of database design.

Beyond traditional access control, inference control mechanisms have been proposed to prevent unwanted flows of information. Information emerges from the answers to database queries by, e. g., additionally taking database constraints or common sense knowledge into account. An overview of the inference problem in different areas can be found, e. g., in [**?**]. Prevention of inferences in relational and MLS databases have been investigated, e. g., in [**?**,**?**,**?**,**?**,**?**].

Being initially proposed in [**?**,**?**] the ideas of protecting information in logical databases according to security policies by lying and/or refusing to answer have been elaborated in [**?**,**?**,**?**] under the notion of *Controlled Query Evaluation*. This technique was extended for relational databases in [**?**] and optimized for specific conditions in [**?**,**?**].

3 Preliminaries

3.1 Relational Databases and Open Queries

A *relation schema* $RS = \langle R, \mathcal{U}, \Sigma \rangle$ describes the structure of a relation in a relational database. R is the *relation symbol*, $\mathcal{U}$ is a finite set of *attributes* with $|\mathcal{U}| = n$, and Σ is a finite set of *semantic constraints* on $\mathcal{U}$ which we assume to be a minimal cover (see [**?**]) of functional dependencies – the most prevalent kind of local constraints in actual relational databases.

An *instance* r of a relation schema is a finite Herbrand interpretation of the schema satisfying Σ and considering R as a predicate. The values c_i of a *tuple* $\mu = R(c_1, \ldots, c_n)$ are elements of an infinite set of constants *Const* and the value of an attribute A in a tuple μ is referred to by $\mu[A]$. With $\models_{\mathrm{M}}$ we denote the

satisfaction relation between an interpretation and a formula, so if μ is element of r, we write $r \models_{\mathrm{M}} \mu$.

Let $\mathcal{A}, \mathcal{B} \subseteq \mathcal{U}$, then r satisfies the *functional dependency (FD)* $\mathcal{A} \rightarrow \mathcal{B}$ if for any two tuples μ_1, μ_2 of r it holds that $\mu_1[B] = \mu_2[B]$ for every $B \in \mathcal{B}$ whenever $\mu_1[A] = \mu_2[A]$ for every $A \in \mathcal{A}$. $\mathcal{K} \subseteq \mathcal{U}$ is a *key* of RS if $\mathcal{K} \rightarrow \mathcal{U}$ is logically implied by Σ and $\mathcal{K}$ is minimal with this property. RS is in *Boyce-Codd normal form (BCNF)* if for each FD $\mathcal{A} \rightarrow \mathcal{B}$, logically implied by Σ and with $\mathcal{B} \not\subseteq \mathcal{A}$, $\mathcal{A}$ is a superset of a key. We assume single-relation databases (with schema $\langle R, \mathcal{U}, \Sigma \rangle$ and instance r unless otherwise stated), leaving inter-relational considerations for future research.

Database queries are expressed in a fragment of the relational calculus. Let *Var* be a set of variables, then the *query language* $\mathscr{L}_{\mathrm{q}}$ is the set of formulas of the form $(\exists X_1) \ldots (\exists X_l) R(v_1, \ldots, v_n)$ with $0 \leq l \leq n$, $X_i \in \mathit{Var}$, $v_i \in \mathit{Const} \cup \mathit{Var}$, $\{X_1, \ldots, X_l\} \subseteq \{v_1, \ldots, v_n\}$, and $v_i \neq v_j$ if $v_i, v_j \in \mathit{Var}$ and $i \neq j$. If $\{v_1, \ldots, v_n\} \cap \mathit{Var} = \{X_1, \ldots, X_l\}$ for a query from $\mathscr{L}_{\mathrm{q}}$, then it is *closed*; if there are free variables in the query, $(\{v_1, \ldots, v_n\} \cap \mathit{Var}) \backslash \{X_1, \ldots, X_l\} \neq \emptyset$, then it is *open*. We denote queries by $\Phi(\boldsymbol{V})$ where $\boldsymbol{V}$ is the vector of the free variables in Φ. When convenient we omit the variable vector $\boldsymbol{V}$ (if $\boldsymbol{V}$ is empty or not important in the context). With $\mathscr{L}_{\mathrm{q}}^{c}$ we denote the language containing exactly the closed queries from $\mathscr{L}_{\mathrm{q}}$.

With $\mathit{sel}(\Phi(\boldsymbol{V})) \subseteq \mathcal{U}$ we denote the set of attributes for which a constant appears in $\Phi(\boldsymbol{V})$. Since a closed query $\Phi(\boldsymbol{V}) \in \mathscr{L}_{\mathrm{q}}^{c}$ corresponds to a projection of a tuple to $\mathit{sel}(\Phi(\boldsymbol{V}))$ we refer to formulas from $\mathscr{L}_{\mathrm{q}}^{c}$ as *select-project-queries*. The assignment of attribute A in $\Phi(\boldsymbol{V})$ is denoted by $\Phi(\boldsymbol{V})[A]$ ($\in \mathit{Const} \cup \mathit{Var}$). In the following we assume a single user sending queries to the database and call him "the user" for short.

3.2 Controlled Query Evaluation

The *ordinary evaluation* of a closed query Φ in an instance r is defined by $\mathit{eval}^*(\Phi)(r) := \texttt{if}\ r \models_{\mathrm{M}} \Phi\ \texttt{then}\ \Phi\ \texttt{else}\ \neg\Phi$. An open query $\Phi(\boldsymbol{V})$ is evaluated by replacing the free variables $\boldsymbol{V}$ with constants $\boldsymbol{c}$ such that the resulting (closed) sentence is true in r:

$$\mathit{eval}^*(\Phi(\boldsymbol{V}))(r) = \{\Phi(\boldsymbol{c}) \mid \boldsymbol{c} \in \mathit{Const} \times \cdots \times \mathit{Const} \text{ and } r \models_{\mathrm{M}} \Phi(\boldsymbol{c})\}.$$

Note that the evaluation of an open query always implies a negative part: a variable assignment $\boldsymbol{c}'$ makes $\Phi(\boldsymbol{V})$ *false* in the database instance if $\Phi(\boldsymbol{c}')$ does *not* occur in $\mathit{eval}^*(\Phi(\boldsymbol{V}))(r)$.

Controlled Query Evaluation (CQE) deviates from the ordinary evaluation if any of the previously declared potential secrets is going to be disclosed to the user. A *potential secret* Ψ is a sentence of a policy language $\mathscr{L}_{\mathrm{ps}}$ being a fragment of a suitable logic as discussed in [**?**]. The user may learn that Ψ is false in the instance r; if, however, Ψ is true in r, then this information must be kept secret. The *confidentiality policy* (or "policy" for short) is a finite set *pol*, consisting of potential secrets. From a security perspective it is desirable to reach

preservation of confidentiality even if the user knows the policy. However, this cannot always be guaranteed which justifies the option of hiding the policy, so it may be *known* or *unknown* to the user. We assume the policy language to be the set of (closed) select-project-queries over the relation R, i. e., $\mathscr{L}_{\mathrm{ps}} = \mathscr{L}_{\mathrm{q}}^{\mathrm{c}}$. Finally, the user is supposed to be aware of the semantic constraints of the database being expressed by the set Σ of the relation schema RS.[4] We thus initially set the *user knowledge*, consisting of sentences that are true in r and that the user is supposed to be aware of, to $log_0 = \Sigma$.

For closed queries, CQE with potential secrets enforced by *refusal* has been defined, depending on the *user awareness* a regarding the policy ($a = known$ or $a = unknown$), by $cqe^a(Q, log_0)(r, pol) := \langle (ans_1, log_1), (ans_2, log_2), \dots \rangle$ for a sequence $Q = \langle \Phi_1, \Phi_2, \dots \rangle$ of closed queries. It uses a *censor function* to determine the returned answers ans_i (with `mum` denoting a refusal) and the updated user knowledges log_i. The censor inspects whether the true or the negated answer to a query would enable the user to infer a potential secret (in the case of an unknown policy only true potential secrets are considered).[5] If so, the answer is refused and the user knowledge does not change. Otherwise, the answer is given honestly and the user knowledge is updated with this answer. We recall the definitions from [?] amended by the "improved refusal"[6] result from [?]:

$$
\begin{aligned}
&censor^{known}(pol, log, \Phi, r) := (\text{exists } \Psi)(\Psi \in pol \text{ and} \\
&\quad (log \cup \{eval^*(\Phi)(r)\} \models \Psi \text{ or } log \cup \{\neg eval^*(\Phi)(r)\} \models \Psi)) \\
&censor^{unknown}(pol, log, \Phi, r) := (\text{exists } \Psi)(\Psi \in pol \text{ and} \\
&\quad r \models_{\mathrm{M}} \Psi \text{ and } log \cup \{eval^*(\Phi)(r)\} \models \Psi) \\
&ans_i := \texttt{if } log_{i-1} \models eval^*(\Phi_i)(r) \texttt{ then } eval^*(\Phi_i)(r) \texttt{ else} \\
&\qquad \texttt{if } censor^a(pol, log_{i-1}, \Phi_i, r) \texttt{ then mum else } eval^*(\Phi_i)(r) \\
&log_i := \texttt{if } log_{i-1} \models eval^*(\Phi_i)(r) \texttt{ or } censor^a(pol, log_{i-1}, \Phi_i, r) \\
&\qquad \texttt{then } log_{i-1} \texttt{ else } log_{i-1} \cup \{eval^*(\Phi_i)(r)\}
\end{aligned}
$$

To model "correct" and "harmless" user knowledge we assume $r \models_{\mathrm{M}} log_0$ and we require each instance-policy-pair (r, pol) to satisfy a precondition depending on the user awareness: if $a = known$ the precondition for (r, pol) is $log_0 \not\models \Psi$, for every $\Psi \in pol$; if $a = unknown$ the precondition is $\texttt{if } r \models_{\mathrm{M}} \Psi \texttt{ then } log_0 \not\models \Psi$, for every $\Psi \in pol$. According to [?] the CQE cqe^a preserves confidentiality in the sense of the following definition:

Definition 1. *A CQE is* confidentiality preserving for a policy *pol if for every finite prefix Q' of a sequence Q of queries the following holds: For every $\Psi \in pol$ and for every instance r (with (r, pol) satisfying the precondition) there exists an instance r' and a policy pol' (with (r', pol') satisfying the precondition) with*

(1) (r', pol') leads to the same answers for Q' as (r, pol);

[4] These semantic constraints may reflect business rules the user is aware of.

[5] Inspecting the negated answer is necessary to avoid meta-inferences [?].

[6] If the user already knows the answer to his query, the censor is bypassed.

(2) $eval^*(\Psi)(r') = \neg\Psi$*;*
(3) if $a = known$*:* $pol' = pol$*.*

A CQE is confidentiality preserving *if it is confidentiality preserving for all possible policies.*

Example 1. Consider the schema $\langle BANK, \mathcal{U}, \Sigma\rangle$ of a bank database with $\mathcal{U} = \{acc_no, acc_holder\}$, $\Sigma = \emptyset$, and the known policy $pol = \{\Psi\}$ with $\Psi \equiv (\exists X_N) BANK(X_N, \text{Smith})$, protecting that Smith has an account at the bank. The instance *bank* of *BANK* and a query Φ_1 are given by $bank = \{BANK(123, \text{Smith}), BANK(456, \text{Jones})\}$ and $\Phi_1 \equiv BANK(123, \text{Smith})$. Since $eval^*(\Phi_1)(bank) = \Phi_1$ and $\Phi_1 \models \Psi$ we get $censor^{known} = true$ and thus $cqe^{known}(\langle\Phi_1\rangle, log_0)(bank, pol) = \langle(\texttt{mum}, log_0)\rangle$.

For convenience, we consider closed queries as specific open queries without free variables in the following. Consequently, the evaluation of a closed query Φ will be $\{\Phi\}$ (if $r \models_{\mathrm{M}} \Phi$) or $\emptyset$ (if $r \not\models_{\mathrm{M}} \Phi$), respectively.

3.3 Static CQE for Closed Queries

In previous work we investigated static forms of CQE for closed database queries. In this context "static" means that we proposed suitable restrictions regarding query and policy languages as well as schema constraints to avoid the costly inference control mechanism and the ever growing log file. In the following we shortly summarize our contributions.

In [**?**] we considered a simple query language only allowing for select-queries in combination with a policy language being equivalent to the query language $\mathscr{L}_{\mathrm{q}}^{c}$ introduced in Subsect. 3.1. It turned out that no further schema restrictions are necessary when confining the user and the security administrator to these languages. We showed that the declarative goals of inference control (as in Def. 1) can be reached by applying a simple pattern matching algorithm to the single queries.

When relaxing the query language such that select-project-queries can be expressed we had to impose certain restrictions as elaborated in [**?**]. Basically, the relation schema must be in *object normal form (ONF)*, i. e., it must be in BCNF and have a unique key [**?**] (an assumption that occurs frequently in practice), and potential secrets must adhere to a syntactic constraint to still guarantee preservation of confidentiality.

In [**?**] we refined the results of [**?**]. We found conditions for using logical connectives in query and policy languages and we relaxed the syntactic constraint for potential secrets. Since we refer to this constraint in the following section we recall the definition from [**?**]:

Definition 2. *Let* $RS = \langle R, \mathcal{U}, \Sigma\rangle$ *be a relation schema in ONF with* Σ *being a minimal cover of FDs. The left-hand side of an FD* $\sigma \in \Sigma$ *is denoted by* $lhs(\sigma)$*. The* set of fact schemas of *RS is then defined by*

$$fs(RS) = \{A \mid A \in \mathcal{U}\} \cup \{\mathcal{A} \mid \text{exists } \sigma \in \Sigma : \mathcal{A} \subseteq lhs(\sigma)\} \cup \\ \{\mathcal{A}B \mid \text{exists } \sigma \in \Sigma \text{ with } \mathcal{A} \subseteq lhs(\sigma) \text{ and } B \in \mathcal{U} \backslash lhs(\sigma)\}.$$

4 Static Controlled Evaluation of Open Queries

Several CQE approaches to open queries have been proposed by Biskup and Bonatti [**?**] for the enforcement methods of lying and refusal. Although being effectively computable due to suitable syntactic restrictions, these approaches still suffer from high computational complexity and the need of a log file. Our goal is the development of a static CQE for open queries with refusal as enforcement method. We only consider policies $pol \subseteq \mathscr{L}_{\mathrm{ps}}$ satisfying $sel(\Psi) \in fs(RS)$ for each $\Psi \in pol$ and assume for Subsect. 4.1 and **??** that pol is known to the user.

4.1 A First Approach

As described in Subsect. 3.2, the ordinary evaluation of an open query yields a set of sentences being true in the database instance. Consequently, the original form of static CQE cannot be applied to open queries, since the censor was constructed with singleton answer sets in mind. Moreover, it does not seem appropriate to either allow or refuse entire queries; regarding an open query, some assignments of the free variables may compromise confidentiality whereas others may not. Thus, evaluating open queries in a controlled way should basically determine which variable assignments will lead to a disclosure of secrets and exclude them from the answer. A variable assignment not being returned as part of the answer to a query then can be interpreted in two different ways: it may make the query false in the database instance; or it may lead to the disclosure of a potential secret. The user is able to distinguish these two cases since he is supposed to be aware of the policy.

The positive part ans of the controlled answer to an open query $\Phi(\boldsymbol{V})$ is determined by means of the ordinary query evaluation $eval^*$ and the set $ref(used)$ possibly being infinite and containing every "harmful" $\Phi(\boldsymbol{c})$:

$$ref(\Phi(\boldsymbol{V}), pol) = \{\Phi(\boldsymbol{c}) \mid \boldsymbol{c} \in Const \times \cdots \times Const \text{ and exists } \Psi \in pol : \Phi(\boldsymbol{c}) \models \Psi\} \tag{1}$$

$$ans(\Phi(\boldsymbol{V}), pol, r) = eval^*(\Phi(\boldsymbol{V}))(r) \backslash ref(\Phi(\boldsymbol{V}), pol) \tag{2}$$

Besides this positive part, the user is aware of a completeness information, as introduced in [**?**], basically saying that each substitution of the variables in $\boldsymbol{V}$ is either false in the instance r or true in r and part of the answer or true in r and not part of the answer. This is expressed by the following *completeness sentence* (with ans denoting the positive part (2) of the controlled answer, i. e., a finite set of ground substitutions of $\Phi(\boldsymbol{V})$):

$$comp(\Phi(\boldsymbol{V}), ans) \equiv \tag{3}$$
$$(\forall \boldsymbol{V})[\neg\Phi(\boldsymbol{V}) \vee (\Phi(\boldsymbol{V}) \wedge \bigvee_{\Phi(\boldsymbol{c}) \in ans} \boldsymbol{V} = \boldsymbol{c}) \vee (\Phi(\boldsymbol{V}) \wedge \bigwedge_{\Phi(\boldsymbol{c}) \in ans} \boldsymbol{V} \neq \boldsymbol{c})]$$

Observe that this completeness sentence is actually a tautology because it is equivalent to $(\forall \boldsymbol{V})[\neg\Phi(\boldsymbol{V}) \vee \Phi(\boldsymbol{V})]$; therefore the user can construct it by himself and it needs not to be added to the user knowledge explicitly.

Having sent an open query $\Phi(\boldsymbol{V})$ to the database, the answer according to (2) will be returned to the user and the assumed user knowledge *log* will be updated, i. e., if log_i denotes the user knowledge before answering $\Phi(\boldsymbol{V})$, then the user knowledge log_{i+1} after having answered $\Phi(\boldsymbol{V})$ is determined by $log_{i+1} = log_i \cup ans(\Phi(\boldsymbol{V}), pol, r)$.

4.2 The "Known Policy Problem"

At first glance, a known policy does not give the user any useful information with respect to the disclosure of secrets: each element of the policy is a *potential* secret, i. e., it has to be kept secret if it is true in the database. However, the user is not able to gain information about the actual truth value of a potential secret when only considering the policy. Nevertheless, a problem might emerge from the user awareness regarding the policy in combination with the completeness sentence which can be rewritten as an implication and then be exploited for disclosing a potential secret. We first illustrate this problem by an example and then analyze it more generally.

Example 2. Consider schema, policy and instance from Example 1, but assume that $\Sigma = \{acc_no \rightarrow acc_holder\}$. Observe that *BANK* is in ONF. Two queries are given by $\Phi_1(X_N, X_H) \equiv BANK(X_N, X_H)$ and $\Phi_2 \equiv (\exists X_H) BANK(123, X_H)$. Φ_1 is an open query, asking for all tuples in *bank*. According to the (tentative) mechanism from Subsect. 4.1 we get:

$$eval^*(\Phi_1)(bank) = \{BANK(123, \text{Smith}), BANK(456, \text{Jones})\} \tag{4}$$

$$ref(\Phi_1, pol) = \{BANK(123, \text{Smith})\} \tag{5}$$

$$ans(\Phi_1, pol, bank) = \{BANK(456, \text{Jones})\} = ans_1 \tag{6}$$

$$\begin{aligned} comp(\Phi_1, ans_1) \equiv (\forall X_N)(\forall X_H)[\quad & \neg BANK(X_N, X_H) \\ \vee\, (BANK(X_N, X_H) \wedge (X_N = 456 \wedge X_H = \text{Jones})) & \\ \vee\, (BANK(X_N, X_H) \wedge (X_N \neq 456 \vee X_H \neq \text{Jones})) \quad &] \end{aligned} \tag{7}$$

Φ_2 is a closed query (since it does not contain free variables) and asks whether there is an account with the number 123. Confidentiality is not compromised since $\Phi_2 \not\models \Psi$. Therefore, Φ_2 is answered as follows:

$$ans(\Phi_2, pol, bank) = \{(\exists X_H) BANK(123, X_H)\} \tag{8}$$

The user knowledge now consists of $log_0 (= \Sigma)$, the answers to Φ_1 (**??**) and Φ_2 (**??**), and the completeness sentence (**??**). From (**??**) the user knows that $BANK(123, c)$ holds in *bank* for some suitable constant $c \in Const$. The variable assignment $(123, c)$ belongs to the third part of the disjunction in (**??**): it makes Φ_1 true in *bank* but is not part of the answer to Φ_1. Thus, $BANK(123, c)$ must be a secret. Since the only secret is $(\exists X_N) BANK(X_N, \text{Smith})$, the constant c must be identified with Smith. As a result the user knows that the secret is true in *bank*: $bank \models_{\mathrm{M}} BANK(123, \text{Smith})$ and $BANK(123, \text{Smith}) \models \Psi$ and thus $bank \models_{\mathrm{M}} \Psi$.

$r \models_{\mathrm{M}} \Phi(\boldsymbol{c})$	$r \not\models_{\mathrm{M}} \Phi(\boldsymbol{c})$	
A	$B(=\emptyset)$	$\Phi(\boldsymbol{c}) \in ans$
C ("harmful")	D	$\Phi(\boldsymbol{c}) \notin ans$

Fig. 1. Classification of variable assignments

For a more general analysis first reconsider the completeness sentence (3) and its visualization in Fig. **??**: The box depicts the (infinite) set of all possible variable assignments $\boldsymbol{c}$ for the free variables $\boldsymbol{V}$ of an open query $\Phi(\boldsymbol{V})$ and is partitioned into four disjoint subsets. The horizontal partition differentiates between $\boldsymbol{c}$ being part of the explicitly returned controlled answer or not whereas the vertical partition differentiates between $\boldsymbol{c}$ making $\Phi(\boldsymbol{V})$ true or false in r. Variable assignments being covered by the first part of the disjunction in (3) belong to either subset B or subset D of the figure; variable assignments being covered by the second part of the disjunction belong to subset A; and variable assignments being covered by the third part of the disjunction belong to subset C. B is empty (variable assignments not making $\Phi(\boldsymbol{V})$ true are not part of the answer) and C contains exactly the "harmful" variable assignments that are true in r but not part of the answer (because they imply a secret). Thus, a variable assignment neither belonging to A nor to D belongs to C. Second, a known policy carries additional information: the variable substitutions not belonging to A and D in Fig. **??** may be identified by inspecting the policy: if $\Phi(\boldsymbol{c}) \models \Psi$ for a $\Psi \in pol$ then $\boldsymbol{c}$ belongs to C. Assuming a unary relation[7] we formalize this information for $pol = \{R(c_1), \ldots, R(c_m)\}$:

$$pol_inf_i \equiv (\forall X)[(\Phi_i(X) \wedge \bigwedge_{\Phi_i(c) \in ans_i} X \neq c) \implies \bigvee_{j=1}^{m} X = c_j] \tag{9}$$

If $(\exists X)R(X) \in pol$, each query would be refused since pol would protect the existence of any tuple in the instance. The system would thus be useless in some sense and therefore we neglect this case in the following.

Now consider a more formal variant of Example **??** for a unary relation with instance $r = \{R(c_1)\}$. The policy and the two queries are given by $pol = \{R(c_1)\}$ with $c_1 \in Const$, $\Phi_1(X) \equiv R(X)$ (asking for all tuples in r), and $\Phi_2(X) \equiv (\exists X)R(X)$ (asking for the existence of a tuple in r). According to the (tentative) control mechanism described above we get the following answers, completeness

[7] Our considerations can easily be adapted to general n-ary relations.

information, and policy information:

$$
\begin{aligned}
&ans_1 = \emptyset, \quad ans_2 = (\exists X)R(X)\\
&comp_1 = (\forall X)[\ \neg R(X) \vee \\
&\qquad (R(X) \wedge \textstyle\bigvee_{R(c)\in ans_1} X = c) \vee (R(X) \wedge \bigwedge_{R(c)\in ans_1} X \neq c)]\\
&\qquad \equiv (\forall X)[\ R(X) \implies (\textstyle\bigvee_{R(c)\in ans_1} X = c \vee \bigwedge_{R(c)\in ans_1} X \neq c)]\\
&pol_inf_1 \equiv (\forall X)[(R(X) \wedge \textstyle\bigwedge_{R(c)\in ans_1} X \neq c) \implies X = c_1]
\end{aligned}
$$

Combining ans_2 and $comp_1$ then yields[8]

$$
\begin{aligned}
&(\exists X)[R(X) \wedge (\textstyle\bigvee_{R(c)\in ans_1} X = c \vee \bigwedge_{R(c)\in ans_1} X \neq c)]\\
&\equiv (\exists X)[R(X) \wedge \textstyle\bigwedge_{R(c)\in ans_1} X \neq c]
\end{aligned}
$$

which, together with pol_inf_1, leads to $X = c_1$. It follows that $r \models_{\mathrm{M}} R(c_1)$ resulting in the user being able to disclose a potential secret.

4.3 Inference Control with Unknown Policies

One possibility to avoid the problem sketched in the previous subsection is to reconsider the query language and the policy language. Confining the user or the security administrator in expressing queries or policies, respectively, might help to avoid the harmful inference problem. We, however, refrain from adjusting the query and policy languages since we consider this a step backwards in our attempt to develop a system being convenient from the perspective of database users and security administrators.

A more promising approach is a reconsideration of the user's knowledge about the policy which is a crucial part of the harmful inference in Example **??**. The completeness information together with the information from the policy may be exploited to infer a secret; consequently, if we hide the potential secrets from the user, we might be able to preserve confidentiality again. The completeness sentence (3) is independent of the user awareness since it does not refer to specific policy elements. We may thus assume the policy to be unknown to the user without having to change the control mechanism as sketched in Subsect. 4.1. This mechanism is secure in the sense of Def. 1 when assuming an unknown policy.

Theorem 1. *Static CQE for open queries with unknown policies preserves confidentiality in the sense of Def. 1.*

Remark. Since static CQE does not need a user log, queries from a sequence may be evaluated separately without compromising confidentiality.

To prepare for a proof sketch for Theorem **??** we now explain our notion of the chase for elements from $\mathscr{L}_{\mathrm{q}}^{c}$ and present three technical lemmas.

[8] The equivalence holds since $\bigvee_{R(c)\in ans_1} X = c$ is empty being equivalent with *false*.

The Chase for Generalized Tuples. Originally, the chase is defined for sets of (full) tuples[9], whereas we consider subsets of the language $\mathscr{L}_{\mathrm{q}}^{c}$. Because of the structure of these formulas, however, we are able to interpret them as generalized tuples, i. e., tuples possibly containing null values, by (temporarily) neglecting the existential quantifiers and interpreting the variables as null values of the type "existing but unknown". Considering elements from *Const* as distinguished and elements from *Var* as non-distinguished variables the notion of chasing can be applied to subsets of $\mathscr{L}_{\mathrm{q}}^{c}$ as well. We will use this mechanism in the proof of Theorem **??**.

Proof of Confidentiality. For each formula being true in a database instance there must be a ground atom, i. e., a sentence not containing any variables, implying this formula and also being true in the instance:

Lemma 1. *Consider an instance r and a $\chi \in \mathscr{L}_{\mathrm{q}}^{c}$. It holds that $r \models_{\mathrm{M}} \chi$ iff there exists a ground atom $\chi_g \in \mathscr{L}_{\mathrm{q}}^{c}$ with $r \models_{\mathrm{M}} \chi_g$ and $\chi_g \models \chi$.*[10]

When we chase a set $\mathcal{S}$ of closed select-project-queries (considered as generalized tuples by neglecting the existential quantifiers) with FDs Σ to construct a database instance (represented by a set of ground atoms) then each formula being true in this instance and containing only existentially bound variables or constants that already occur in $\mathcal{S}$ is implied by $\mathcal{S} \cup \Sigma$:

Lemma 2. *Consider a disjoint partition of the set of constants, $Const = Const_A \uplus Const_B$, and let $\mathcal{S} \subset \mathscr{L}_{\mathrm{q}}^{c}$ such that each constant occurring in an element of $\mathcal{S}$ is from $Const_A$; Σ a set of FDs; $\chi_c \in \mathscr{L}_{\mathrm{q}}^{c}$ a generalized tuple from the result of chasing $\mathcal{S}$ with Σ; $\chi_g \in \mathscr{L}_{\mathrm{q}}^{c}$ a ground atom resulting from replacing all (existentially quantified) variables in χ_c with constants from $Const_B$; and $\chi \in \mathscr{L}_{\mathrm{q}}^{c}$ with $\chi_g \models \chi$ and $\chi[A] \in Var$ or $\chi[A] \in Const_A$ for every attribute A. Then it holds that $\mathcal{S} \cup \Sigma \models \chi$.*

By Lemma **??**, a potential secret being implied by a set $\mathcal{S}$ of closed select-project-queries and a set of FDs is already implied by a single formula from $\mathcal{S}$ under the assumption of ONF.

Lemma 3. *Consider a relation schema $RS = \langle R, \mathcal{U}, \Sigma \rangle$ being in ONF, a set $\mathcal{S} \subset \mathscr{L}_{\mathrm{q}}^{c}$, and a potential secret $\Psi \in \mathscr{L}_{\mathrm{q}}^{c}$ with $sel(\Psi) \in fs(RS)$. It holds that $\mathcal{S} \cup \Sigma \models \Psi$ iff $\chi' \models \Psi$ for a $\chi' \in \mathcal{S}$.*

We are now well prepared for providing a proof sketch for Theorem **??**.

Sketch of Proof. By Def. 1, considering an instance-policy-pair (r, pol), for each potential secret Ψ and each (prefix of a) query sequence Q we have to find an alternative instance-policy-pair (r', pol') (satisfying the precondition) with these properties: (1) (r', pol') leads to the same answers for Q as (r, pol); (2)

[9] Refer to [**?**] for an explanation of the original chase.
[10] The symbol $\models$ denotes logical implication.

r' makes Ψ false. There is no additional requirement regarding pol' since we consider unknown policies. For constructing r' we basically chase the answers to Q regarding r with Σ and replace the remaining variables with new constants. Then, pol' is constructed by adding a potential secret for each constant being newly introduced during the chase. Property (1) follows from this construction and Lemmas **??**–**??**, and property (2) is proven indirectly by assuming that $eval^*(\Psi)(r') = \Psi$ and applying Lemmas **??**–**??**, leading to the disclosure of Ψ by the answers to Q regarding r and pol which contradicts the definition of the CQE mechanism. □

The transition from known to unknown policies slightly changes the semantics of the refusal method. If pol is known to the user, he is able to determine for every variable assignment $\boldsymbol{c}$ not being returned as part of the answer to $\Phi(\boldsymbol{V})$ whether $\boldsymbol{c}$ makes $\Phi(\boldsymbol{V})$ false in r or $\boldsymbol{c}$ leads to the disclosure of a potential secret (i. e., belongs to the *ref*-part of the answer). Refusals are not returned explicitly because the user is able to figure them out himself. If, however, pol is unknown, the *ref*-part of the answer cannot be determined by the user since hiding the policies from the user aims at making the subsets C and D of Fig. **??** indistinguishable. We must still not return explicit refusals in order to keep up this indistinguishability. Therefore, with unknown policies static CQE is enforced rather by filtering than by refusing the answer. Summing up we regain the goal of confidentiality-preservation at the price of leaving the user uncertain about the *ref*-part of the answer to his query.

5 A Control Mechanism for Open Queries

We now sketch an algorithmic implementation of static CQE for open queries. We assume a single query rather than a query sequence which is justified by the remark to Theorem **??**. Our implementation makes use of "classification instances", introduced in [**?**]. A classification instance is a relational representation of a policy: each secret from pol is represented by a tuple where existentially bound variables are replaced by the symbol #.

Static CQE for open queries

input : database instance r, policy $pol = \{\Psi_1, \Psi_2, \ldots, \Psi_m\}$, query Φ
output : controlled answer *ans*

1. Preprocessing
   For each potential secret[11] $\Psi_i \equiv (\exists X_1) \ldots (\exists X_{l_1}) R(\underbrace{X_1, \ldots, X_{l_1}}_{\text{bound variables}}, c_1, \ldots, c_{l_2})$:

   Add a tuple $\mu = R^c(\underbrace{\#, \ldots, \#}_{l_1 \text{ times}}, c_1, \ldots, c_{l_2})$ to the classification instance r^c.
2. Answering Queries
   (a) If Φ is a *closed* query:

[11] The positions of variables and constants are w. l. o. g.

If exists $\mu \in r^c$ such that for all attributes A:
- if $\Phi[A] \in \mathit{Const}$ then $\mu[A] \in \{\Phi[A], \#\}$ (*)
- if $\Phi[A] \in \mathit{Var}$ then $\mu[A] = \#$ (**)

then return `mum`,
else set $\mathit{ans} := \mathit{eval}^*(\Phi)(r)$.

(b) If $\Phi(\boldsymbol{V})$ is an *open* query:
  i. Compute the ordinary query evaluation $\mathit{eval}^*(\Phi(\boldsymbol{V}))(r)$.
  ii. Set $\mathit{ref}(\Phi(\boldsymbol{V}), \mathit{pol}) := \emptyset$.
  iii. For each $\Phi(\boldsymbol{c}) \in \mathit{eval}^*(\Phi(\boldsymbol{V}))(r)$:
    If exists $\mu \in r^c$ satisfying (*) and (**) for all attributes A,
    then set $\mathit{ref}(\Phi(\boldsymbol{V}), \mathit{pol}) := \mathit{ref}(\Phi(\boldsymbol{V}), \mathit{pol}) \cup \{\Phi(\boldsymbol{c})\}$.
  iv. Set $\mathit{ans} := \mathit{eval}^*(\Phi(\boldsymbol{V}))(r) \backslash \mathit{ref}(\Phi(\boldsymbol{V}), \mathit{pol})$.

(c) Return ans.

In the first step the policy is converted into the classification instance which can be done in linear time (in the size of the policy). The second step differs depending on Φ being a closed query or an open query. If Φ is closed, r^c is searched for a tuple satisfying (*) and (**) ($\Phi \models \Psi$ for a $\Psi \in \mathit{pol}$ iff there exists such a tuple). This can be done by constructing a set P_Φ that contains *all* tuples satisfying (*) and (**) and then checking whether $r^c \cap P_\Phi$ is non-empty. With n denoting the number of attributes in R, we get $|P_\Phi| \leq 2^n$ and when storing r^c in a suitable data structure like a B-tree the control mechanism has a runtime of $O(2^n \cdot \mathit{log}(m))$ or $O(\mathit{log}(m))$ if we consider the number of attributes in R fixed and reasonably small (cf. [**?**]). If Φ is open, a set $P_{\Phi(\boldsymbol{c})}$ can be constructed analogously for each $\boldsymbol{c}$ with $\Phi(\boldsymbol{c}) \in \mathit{eval}^*(\Phi(\boldsymbol{V}))(r)$. The runtime can then be estimated by $O(k \cdot \mathit{log}(m))$ with k denoting the number of tuples in $\mathit{eval}^*(\Phi(\boldsymbol{V}))(r)$.

6 Conclusion and Future Work

We presented an inference control approach for open relational database queries which is static in the sense that the goals of computationally expensive inference control can be reached at runtime by actually performing an efficient filtering mechanism. We developed our approach in the framework of CQE. It turned out that assuming the user to be aware of the confidentiality policy is incompatible with open queries but we formally showed that hiding the policy is sufficient to preserve confidentiality. We also proposed an implementation of our mechanism with a linear preprocessing time (in the size of the policy) and a runtime of $O(k \cdot \mathit{log}(m))$ per query (with k denoting the answer size and m denoting the policy size).

Future research should address several enhancements. For example, more expressive query and policy languages could be investigated, allowing for conjunction, disjunction and negation. Also further kinds of constraints like (intra-relational) multivalued dependencies or (inter-relational) inclusion dependencies could be taken into account. Finally, an implementation of our approach with actually employed access control mechanisms like Oracle's virtual private databases would be desirable.

References

1. S. Abiteboul, R. Hull, and V. Vianu. Foundations of Databases. Addison-Wesley, 1995.
2. E. Bertino and R. Sandhu. Database security – concepts, approaches, and challenges. IEEE Trans. Dependable Sec. Comput., 2(1):2–18, 2005.
3. J. Biskup. Boyce-Codd normal form and object normal forms. Inf. Process. Lett., 32(1):29–33, 1989.
4. J. Biskup and P. Bonatti. Controlled query evaluation for enforcing confidentiality in complete information systems. Int. J. Inf. Sec., 3(1):14–27, 2004.
5. J. Biskup and P. Bonatti. Controlled query evaluation with open queries for a decidable relational submodel. Ann. Math. Artif. Intell., 50:39–77, 2007.
6. J. Biskup and P. A. Bonatti. Lying versus refusal for known potential secrets. Data Knowl. Eng., 38(2):199–222, 2001.
7. J. Biskup and P. A. Bonatti. Controlled query evaluation for known policies by combining lying and refusal. Ann. Math. Artif. Intell., 40:37–62, 2004.
8. J. Biskup, D. W. Embley, and J.-H. Lochner. Reducing inference control to access control for normalized database schemas. Inf. Process. Lett., 106(1):8–12, 2008.
9. J. Biskup and J.-H. Lochner. Enforcing confidentiality in relational databases by reducing inference control to access control. In J. A. Garay, A. K. Lenstra, M. Mambo, and R. Peralta, editors, Proc. Information Security (ISC), volume 4779 of *LNCS*, pages 407–422. Springer, 2007.
10. J. Biskup, J.-H. Lochner, and S. Sonntag. Optimization of the controlled evaluation of closed relational queries. In D. Gritzalis and J. Lopez, editors, *Proc. IFIP SEC*, volume 297 of IFIP AICT, pages 214–225. Springer, 2009.
11. P. Bonatti, S. Kraus, and V. S. Subrahmanian. Foundations of secure deductive databases. IEEE Trans. Knowl. Data Eng., 7(3):406–422, 1995.
12. A. Brodsky, C. Farkas, and S. Jajodia. Secure databases: constraints, inference channels, and monitoring disclosures. IEEE Trans. Knowl. Data Eng., 12(6):900–919, 2000.
13. J.-W. Byun and E. Bertino. Micro-views, or on how to protect privacy while enhancing data usability—concepts and challenges. ACM SIGMOD Record, 35(1):9–13, 2006.
14. F. Cuppens and A. Gabillon. Cover story management. Data Knowl. Eng., 37(2):177–201, 2001.
15. S. Dawson, S. De Capitani di Vimercati, and P. Samarati. Specification and enforcement of classification and inference constraints. In IEEE Symposium on Security and Privacy, pages 181–195, 1999.
16. H. S. Delugach and T. H. Hinke. Wizard: A database inference analysis and detection system. IEEE Trans. Knowl. Data Eng., 8(1):56–66, 1996.
17. C. Farkas and S. Jajodia. The inference problem: a survey. SIGKDD Explorations, 4(2):6–11, 2002.
18. A. Galinovic and V. Antoncic. Polyinstantiation in relational databases with multilevel security. In Proc. ITI, pages 127–132. IEEE, 2007.
19. P. P. Griffiths and B. W. Wade. An authorization mechanism for a relational database system. ACM Trans. Database Syst., 1(3):242–255, 1976.
20. J. Hale and S. Shenoi. Analyzing FD inference in relational databases. Data Knowl. Eng., 18(2):167–183, 1996.
21. S. Jajodia and R. S. Sandhu. Toward a multilevel secure relational data model. In J. Clifford and R. King, editors, SIGMOD Conference, pages 50–59. ACM Press, 1991.

22. T. F. Lunt, D. E. Denning, R. R. Schell, M. Heckman, and W. R. Shockley. The SeaView security model. IEEE Trans. Software Eng., 16(6):593–607, 1990.
23. W. Rjaibi and P. Bird. A multi-purpose implementation of mandatory access control in relational database management systems. In M. A. Nascimento, M. T. Özsu, D. Kossmann, R. J. Miller, J. A. Blakeley, and K. B. Schiefer, editors, Proc. VLDB, pages 1010–1020, 2004.
24. R. Sandhu. Lattice-based access control models. Computer, 26(11):9–19, 1993.
25. G. L. Sicherman, W. de Jonge, and R. P. van de Riet. Answering queries without revealing secrets. ACM Trans. Database Syst., 8(1):41–59, 1983.
26. M. Stonebraker and E. Wong. Access control in a relational data base management system by query modification. In Proc. ACM/CSC-ER Annual Conference, pages 180–186. ACM Press, 1974.
27. T.-A. Su and G. Özsoyoglu. Controlling FD and MVD inferences in multilevel relational database systems. IEEE Trans. Knowl. Data Eng., 3(4):474–485, 1991.