An Effective and Efficient Inference Control System for Relational Database Queries

Dissertation

zur Erlangung des Grades eines

Doktors der Naturwissenschaften

der Technischen Universität Dortmund
an der Fakultät für Informatik

von

Jan-Hendrik Lochner

Dortmund

2011

Tag der mündlichen Prüfung: 10.02.2011

Dekan: Prof. Dr. Peter Buchholz

1. Gutachter: Prof. Dr. Joachim Biskup
2. Gutachter: Prof. Dr. Gabriele Kern-Isberner

Abstract

Protecting confidential information in relational databases while ensuring availability of public information at the same time is a demanding task. Unwanted information flows due to the reasoning capabilities of database users require sophisticated inference control mechanisms, since access control is in general not sufficient to guarantee the preservation of confidentiality. The policy-driven approach of Controlled Query Evaluation (CQE) turned out to be an effective means for controlling inferences in databases that can be modeled in a logical framework. It uses a censor function to determine whether or not the honest answer to a user query enables the user to disclose confidential information which is declared in form of a confidentiality policy. In doing so, CQE also takes answers to previous queries and the user's background knowledge about the inner workings of the mechanism into account.

Relational databases are usually modeled using first-order logic. In this context, the decision problem to be solved by the CQE censor becomes undecidable in general because the censor basically performs theorem proving over an ever growing user log. In this thesis, we develop a stateless CQE mechanism that does not need to maintain such a user log but still reaches the declarative goals of inference control. This feature comes at the price of several restrictions for the database administrator who declares the schema of the database, the security administrator who declares the information to be kept confidential, and the database user who sends queries to the database.

We first investigate a scenario with quite restricted possibilities for expressing queries and confidentiality policies and propose an efficient stateless CQE mechanism. Due to the assumed restrictions, the censor function of this mechanism reduces to a simple pattern matching. Based on this case, we systematically enhance the proposed query and policy languages and investigate the respective effects on confidentiality. We suitably adapt the stateless CQE mechanism to these enhancements and formally prove the preservation of confidentiality. Finally, we develop efficient algorithmic implementations of stateless CQE, thereby showing that inference control in relational databases is feasible for actual relational database management systems under suitable restrictions.

Table of Contents

List of Figures

List of Definitions

List of Lemmas, Theorems, Propositions and Corollaries

Chapter 1

Motivation

You should, of course, take security seriously but not lose sight of the fact that your database has to serve some useful purpose.

(D. Gollmann, Computer Security)

Modern database applications are usually employed in distributed environments and accessed by a variety of users with different intentions to store, process, and retrieve homogeneously structured data. In general, most of these databases should not be unconditionally publicly accessible: For example, a database system that stores sensitive data should keep this data secret from database users unless they are allowed to read it, whereas non-sensitive data may in principle be accessible by every user of the system. Moreover, in the sense of informational self-determination, a user who provides personal data to a database system should be able to freely decide whether or not this data may be revealed to other users. This gives rise to the need of secure database systems.

Providing a secure database system, however, is a demanding task in general because security is a rather diverse concept. It is generally accepted (see, for example, [CFMS94, Gol06, Bis09]) to divide the notion of security into several interests, some of which are *availability*, *confidentiality*, *integrity*, *authenticity* and *non-repudiation*. When talking about security, from the perspective of database users it seems reasonable to prioritize availability and confidentiality over the remaining interests: For a database user to complete his task it is essential that the data needed is available; on the contrary, sensitive or personal data should not be disclosed to certain users under any circumstances in order to preserve confidentiality. Obviously, there is a trade-off between the security interests of availability and confidentiality: A database system can be seen as "perfectly secure" in the sense of confidentiality if it refuses to reply user queries at all. However, such a system would be "overprotective" and useless from the point of view of a user who relies on the system to complete a task, since it completely lacks availability.

A widespread approach to guarantee the preservation of confidentiality in database systems while ensuring good availability at the same time is *access control*. Roughly speaking, in this approach (which is elaborated on in more detail in Section 1.1), access rights are declared for combinations of subjects (database users) and objects (database contents). Access to an object is only granted if it is allowed according to these access rights. On the one hand, access control mechanisms for database systems are efficiently implementable in general; on the other hand, unfortunately, access control is insufficient when aiming at preserving confidentiality on the level of information, as explained in the following.

A crucial facet of database security is the distinction between data security and information security. While the term *data* refers to the pure database entries, that is, the constant symbols that represent the contents of the database, the term *information* is a more abstract level and denotes interpreted data. In general, there is an overlap between these two notions, and it often depends on the context whether to speak of data or of information. In the context of database queries, the pure evaluation of a query may be considered as data, whereas the result of somehow "processing" this evaluation may be considered as information. For example, the statements "Smith's bank account number is 101" and "The bank account with number 101 has a balance of $ 1,000" are data, since they might be retrieved from a database using simple queries. If, however, the querying database user is aware of the fact that bank account numbers are unique, he might process the query answers together with this fact and obtain the information that "Smith has a bank account with a balance of $ 1,000". From this example, it can be seen that data security does not imply information security in general: On the data level, the combination "Smith – $ 1,000" is not disclosed to the user, but on the information level, he is able to derive this combination. Avoiding this problem requires either to employ a mechanism that detects such harmful inferences during runtime or to declare suitable access rights which, however, only shifts the problem of inference detection to declaration time. Unfortunately, detecting such harmful inferences is a costly (or, in general, even undecidable) problem, since for relational databases it can basically be reduced to first-order theorem proving. The issue of *inference control* is addressed in more detail in Section 1.2.

Consequently, when aiming at preserving confidentiality, the administrator of a database system is faced with the decision to employ an access control mechanism—which may fail on the level of information security—or to employ an inference control mechanism—which may lead to infeasible runtimes. Therefore, the objective of this thesis is the development of an efficiently implementable mechanism that preserves confidentiality even on the information level.

As indicated before, we address access control and inference control in the following Sections 1.1 and 1.2, respectively. In Section 1.3, we outline the structure of this thesis.

1.1 Access Control

In order to protect personal or sensitive data in relational databases, access control (see, for example, Gollmann [Gol06]) can be used, which is usually divided into discretionary and mandatory access control.

Discretionary access control (DAC) considers (sets of) data as objects and users, who want to access the data, as subjects. For each object, the "owner" of this object (for example, the user who has inserted the data into the database) can declare which subject should be allowed to access the object in which mode. Regarding the mode, in relational databases it is reasonable to distinguish between read and write access. Alternatively, the declaration of access rights can be performed in a "subject-oriented" way by specifying for each subject which objects it may access in which mode.

DAC can be implemented efficiently when using suitable data structures for storing the declared access rights. In fact, DAC mechanisms are implemented in many relational database management systems, for example, the database language SQL [SQL08] provides the commands `GRANT` and `REVOKE` which allow for granting and revoking access rights for database objects.

In general, however, DAC is not able to effectively protect sensitive data as the following example shows: Suppose a database user s_1 who has inserted data o into a database and is thus considered the owner of o. User s_1 now decides that user s_2 may read o, whereas user s_3 must not access o in any way, and thus s_1 declares suitable access rights regarding o. Then user s_2 reads o, which is allowed according to the declared access rights, and inserts a copy o' of o into the database. Since user s_2 is owner of o', he might decide on his own discretion that user s_3 may read o'. When user s_3 then reads o', he also knows o by definition of o'. Thus, user s_3 is, in a way, able to bypass the access rights declared by user s_1.

A step towards the protection of such unwanted *information flows* is *mandatory access control (MAC)*. In contrast to DAC, this technique does not employ "object-wide" but "system-wide" data protection. Basically, objects as well as subjects get assigned labels which are arranged in a lattice by a security administrator. For subjects, these labels play the role of *clearances*, whereas, regarding objects, security labels represent *classifications*. Access to objects is then controlled by a suitable security model. For example, the Bell-LaPadula security model, see [BL76], allows read accesses only to objects with a classification that is "lower" (or equal) than the clearance of the accessing subject, while write accesses are allowed only if the object's classification is "higher" (or equal) than the subject's clearance. An example for MAC enforcement in actual database management systems is *Oracle Label Security* [J$^+$09].

MAC principally offers the possibility to protect data from unwanted accesses by controlling harmful information flows. Reconsidering the above sketched example, when using MAC, user s_3 could be prevented from disclosing o by assigning suitable labels to s_3, o and every object possibly containing copies of o. In general, however, MAC burdens the security administrator of the database system with the suitable assignment of labels. Moreover, as already indicated in the introduction of Chapter 1, MAC as well as DAC is only able to protect data but not information that is derived from this data.

1.2 Inference Control

When aiming at protecting information instead of data, the *inference problem* emerges: A user of a database system might have, in addition to the data he receives in form of answers to his queries, additional knowledge of the system. This knowledge might then enable him to derive information from the data; since this information is never explicitly delivered to the user, it is hard to control in general.

The inference problem manifests in different forms, see Farkas/Jajodia [FJ02] for a comprehensive survey. In the context of relational databases, mainly semantic constraints in form of data dependencies can be used to obtain information from data. For example, reconsidering the bank account example from the introduction of Chapter 1, the fact that account numbers are unique would usually be expressed by a functional dependency in a relational database. “Applying” this dependency to already known data might then disclose a confidential information, as sketched in the example.

Since access control techniques only work “data-oriented”, the application of semantic constraints cannot be controlled by these techniques and consequently, for the goal of protecting information, access control mechanisms turn out inappropriate in general. Instead of assigning labels to subjects and objects (that is, users and data, respectively), it is more suitable to declare the information to be protected by means of a security policy, which can be represented, for example, as a set of “secrets”. During the execution of the system, the policy must be enforced appropriately: When a user sends a query to the database system, a mechanism should inspect whether the user would be able to disclose a declared secret if the correct answer to his query was returned. If so, this answer should be suitably modified. For example, suppose that a piece of information i is to be kept secret according to a security policy which has been declared for a database system, and consider a user who sends a query q to the database. The enforcement of the policy by an inference control mechanism should meet the following requirements (which are, in fact, manifestations of the security interests availability and confidentiality):

- On the one hand, the answer a_q to the query q should be “helpful” for the user, that is, it should provide information regarding q in the context of the database instance;
- on the other hand, a_q should not enable the user to derive i, that is, considering

the background knowledge k of the user and a suitable inference relation $\models$, the answer a_q should be suitably modified if $\{a_q, k\} \models i$.

Depending on the context, however, the latter requirement may be difficult to meet. For example, when assuming a first-order logic framework, the decision problem for logical implication (which corresponds to the general inference relation $\models$ in the above example) is undecidable in general. Therefore, the development of an inference control mechanism with feasible runtime requires suitable restrictions of the considered framework.

1.3 Structure of this Thesis

The remainder of this thesis is divided into four parts.

In Part I, we introduce some basic concepts from the areas of relational databases and Controlled Query Evaluation that are referred to throughout this thesis. Furthermore, we give an overview of related research.

- In Chapter 2, we introduce a logic-oriented view on relational databases that is adopted for the following investigations. We formally describe the representation of database relations and semantic constraints by means of first-order logic and moreover show how to express and evaluate database queries within this framework.
- In Chapter 3, we present the general approach of Controlled Query Evaluation, which serves as a starting point for efficient inference control mechanisms. We first introduce the basic concept and then concentrate on the "refusal strategy for known potential secrets".
- In Chapter 4, we outline a selection of work that is related to the topic of this thesis. More precisely, we give an overview of research in the fields of access control and inference control in databases with an emphasis on the approach of Controlled Query Evaluation. Finally, we sketch our contributions of previously published work in the context of stateless inference control.

In Part II, the main part of this thesis, we develop stateless (and thus efficiently implementable) inference control mechanisms for relational database queries. We investigate a variety of query/policy language combinations within the logical framework introduced in Part I.

- In Chapter 5, we explain our notion of stateless inference control and identify a parameter setting for the following investigations. Moreover, we formalize the idea of confidentiality preservation which is considered the most important security interest for our inference control mechanisms.

- In Chapter 6, we introduce a first approach for stateless inference control with rather simple policy and query languages. More precisely, each query has the form of a tuple and asks whether or not this tuple actually occurs in the database instance. An element of a policy might express a part of a tuple, and each tuple of the actual database instance which contains this part must not be disclosed to a querying user.

  We moreover prove this approach secure in the sense of our notion of confidentiality preservation and propose an implementation of our approach in form of a simple pattern matching technique. It follows that the declarative goals of inference control can be reached by means of efficient algorithms under suitable restrictions.

- In Chapter 7, we describe and investigate several enhancements of the basic approach from Chapter 6. All of these enhancements aim for more expressive query and policy languages. We explore whether or under what circumstances these enhancements of the approach are still secure in the sense of confidentiality preservation.

  More specifically, regarding the query language, we consider parts of tuples (also known as select-project-queries), Boolean combinations of parts of tuples (asking whether or not these Boolean expressions are true in the actual database instance), and open queries (asking for sets of tuples rather than for truth values); regarding the policy language, we consider Boolean combinations of parts of tuples, and open formulas (for expressing “secret schemas”).

In Part III, we show that the stateless inference control mechanisms from Part II are actually implementable by efficient algorithms.

- In Chapter 8, we present a fragment of the database language SQL and show how elements of the investigated query languages can be represented within this SQL fragment. Moreover, we propose a representation of potential secrets in a relational database system.

- In Chapter 9, we develop SQL representations of the investigated censors, that is, the cores of the stateless inference control mechanisms from Part II. Relying on these implementations, we then propose two algorithms for stateless inference control and prove these algorithms efficient in terms of computational complexity.

Finally, in Part IV, we conclude this thesis with a summary and an evaluation of our results. In particular, we compile a list of advantages our approach has over previous work on inference control in relational databases. Moreover, we sketch some directions for future research.

Part I

Preliminaries and Related Work

Chapter 2

Relational Databases

Among the lots of different kinds of databases and data models, nowadays the most popular one is the relational model, pioneered by Codd [Cod70], which considers data records as *tuples* of relations. Since most of the commercially used database management systems base on this model, we also take it as a basis for our investigations. In the following, we formally define the concepts of relational databases within a first-order logic framework. In Example 2.4 (which can be found in Section 2.3), we illustrate these concepts.

2.1 Schemas and Instances

A relational database consists of two elements: the *database schema* and the *database instance*. The database schema describes the syntactical material and the semantic constraints of the database, and the instance contains the actual contents of the database.

A database schema usually comprises one or more *relation schemas* of the form $R_S = \langle R, \mathcal{U}, \Sigma \rangle$. Each of these relation schemas has a *relation symbol* R, a finite set of *attributes* $\mathcal{U} = \{A_1, \dots, A_n\}$, and a finite set of *local semantic constraints* Σ. In the following, we focus on database schemas that comprise only one relation schema. For convenience, we shortly write sets of attributes without set braces in abstract examples, for example, $\{A_1, A_2, \dots, A_n\}$ is denoted by $A_1 A_2 \dots A_n$. Moreover, a database schema declares the set *Const* of constant symbols from which the actual tuples in the database instance are built. We hereafter assume *Const* to be an infinite set.

A *relation instance* r of a relation schema is an *interpretation* of the schema, considering the relation symbol as a predicate; more specifically, r is an *Herbrand interpretation*, that is, every constant is interpreted by itself. Moreover, r is supposed to be finite, meaning that the predicate symbol is interpreted by a finite set. In the following, we refer to relation instances as instances, for short.

If a formula χ of an appropriate fragment of first-order logic is true in an instance r, we write $r \models_{\mathsf{M}} \chi$, using the symbol $\models_{\mathsf{M}}$ to denote the *model-of relation*. Otherwise, if χ is false in r, we write $r \not\models_{\mathsf{M}} \chi$, which is equivalent to $r \models_{\mathsf{M}} \neg\chi$. For a set $\mathcal{F}$

of formulas it holds that $r \models_{\mathsf{M}} \mathcal{F}$ if and only if $r \models_{\mathsf{M}} \chi$ for all $\chi \in \mathcal{F}$. The *logical implication*, denoted with $\models$, is a relation between a set of formulas $\mathcal{F}$ and a single formula χ which is defined using the model-of relation: χ is logically implied by $\mathcal{F}$, $\mathcal{F} \models \chi$, if for all instances r it holds that $r \models_{\mathsf{M}} \mathcal{F}$ only if $r \models_{\mathsf{M}} \chi$. Like the model-of relation, the logical implication can be extended to sets of formulas: $\mathcal{F} \models \mathcal{G}$ holds if and only if $\mathcal{F} \models \chi$ for all $\chi \in \mathcal{G}$.

On the database level, we identify an instance r with the set of tuples $\mu \equiv R(a_1, \dots, a_n)$ (with $a_i \in Const$) such that $r \models_{\mathsf{M}} \mu$. Consequently, this set of tuples contains exactly those ground atoms[1] being true in r. We shortly say that these tuples "are in r". When we refer to the value of a particular attribute A in a tuple μ, we use the square bracket notation $\mu[A]$.

2.2 Semantic Constraints and Normalization

The most prevalent type of local semantic constraint is the *functional dependency*, or FD for short. If $\mathcal{A}$ and $\mathcal{B}$ are sets of attributes, then an instance r is said to satisfy the FD $\mathcal{A} \to \mathcal{B}$ if any two tuples μ_1 and μ_2 of r that agree on the $\mathcal{A}$-attributes, $\mu_1[A] = \mu_2[A]$ for all $A \in \mathcal{A}$, also agree on the $\mathcal{B}$-attributes, $\mu_1[B] = \mu_2[B]$ for all $B \in \mathcal{B}$. The FD $\mathcal{A} \to \mathcal{B}$ is called *trivial* if $\mathcal{B} \subseteq \mathcal{A}$. The sets of left-hand side and right-hand side attributes of an FD σ are denoted by $lhs(\sigma)$ and $rhs(\sigma)$, respectively.

An FD can be represented as a sentence of first-order logic using universal quantifiers. Consider the relation schema $\langle R, \mathcal{U}, \{\sigma\}\rangle$ with the attribute set $\mathcal{U} = \{A_1, A_2, \dots, A_i, B_1, B_2, \dots, B_j, C_1, C_2, \dots, C_k\}$ and the FD $\sigma \equiv \{A_1, A_2, \dots, A_i\} \to \{B_1, B_2, \dots, B_j\}$, that is, $A_1, A_2, \dots, A_i$ are the left-hand side attributes of σ, $B_1, B_2, \dots, B_j$ are the right-hand side attributes of σ and $C_1, C_2, \dots, C_k$ are the attributes that do not occur in σ. Moreover, without loss of generality, the attributes are supposed to occur in R in the same order as they are listed in $\mathcal{U}$ above. Then σ can be expressed by the following first-order sentence:

$$\begin{aligned}
&(\forall X_{A_1}) \dots (\forall X_{A_i})(\forall X_{B_1}) \dots (\forall X_{B_j})(\forall X_{C_1}) \dots (\forall X_{C_k}) \\
&(\forall X_{B'_1}) \dots (\forall X_{B'_j})(\forall X_{C'_1}) \dots (\forall X_{C'_k}) \\
&[\quad (R(X_{A_1}, \dots, X_{A_i}, X_{B_1}, \dots, X_{B_j}, X_{C_1}, \dots, X_{C_k}) \wedge \\
&\quad R(X_{A_1}, \dots, X_{A_i}, X_{B'_1}, \dots, X_{B'_j}, X_{C'_1}, \dots, X_{C'_k})) \\
&\quad \Rightarrow ((X_{B_1} = X_{B'_1}) \wedge \dots \wedge (X_{B_j} = X_{B'_j})) \quad]
\end{aligned} \tag{2.1}$$

Hereafter, when denoting FDs, we omit the set braces for convenience in abstract examples. For example, the FD $\{A_1, A_2, \dots, A_i\} \to \{B_1, B_2, \dots, B_j\}$ from above is

[1]A ground atom is a formula without variables and logical connectives, that is, in the relational database context, a predicate instantiated only with constants.

shortly denoted by $A_1A_2 \dots A_i \rightarrow B_1B_2 \dots B_j$.

A relation schema is normalized if it adheres to a certain *normal form*. Since database normalization theory is a widespread research area, we only introduce the concepts being needed in subsequent chapters. A building block of many database normal forms is the notion of keys. An attribute set $\mathcal{SK}$ is called a *super key* of R_S if $\Sigma \models \mathcal{SK} \rightarrow \mathcal{U}$, that is, if each instantiation of the $\mathcal{SK}$-attributes uniquely determines the respective tuple in every instance. A *candidate key* $\mathcal{K}$ (or *key* for short) is a minimal super key, that is, $\Sigma \not\models \mathcal{K} \backslash \{A\} \rightarrow \mathcal{U}$ for all attributes $A \in \mathcal{K}$. Each attribute occurring in a key of R_S is a *key attribute*. A relation schema R_S is in *Boyce-Codd normal form (BCNF)* if for every nontrivial FD $\mathcal{A} \rightarrow \mathcal{B}$ with $\Sigma \models \mathcal{A} \rightarrow \mathcal{B}$ the attribute set $\mathcal{A}$ is a super key of R_S. If additionally R_S has a unique candidate key, it is in *object normal form (ONF)*, as introduced by Biskup [Bis89].

Since FDs are in fact syntactic abbreviations for first-order logic formulas, the notion of logical implication also applies for sets of FDs. Moreover, two sets of FDs Σ and Σ' are equivalent, $\Sigma \equiv \Sigma'$, if $\Sigma \models \Sigma'$ and $\Sigma' \models \Sigma$. Throughout this thesis, we assume that sets of FDs are in some sense minimal unless otherwise stated. More precisely, FDs are supposed to have minimal left- and right-hand sides, and sets of FDs should not contain redundant FDs, that is, FDs that are already implied by the other FDs. We specify this notion of minimality in the following definition, which is adopted from [AHV95].

Definition 2.1 (Minimality of FD-sets). *Let Σ and Σ' be sets of FDs. Σ' is called a* minimal cover *of Σ if*

1. *each $\sigma \in \Sigma'$ has the form $\mathcal{A} \rightarrow B$ with $\mathcal{A}$ denoting a set of attributes and B denoting a single attribute;*
2. *$\Sigma' \equiv \Sigma$;*
3. *there exists no proper subset $\Sigma'' \subset \Sigma'$ such that $\Sigma'' \models \Sigma$;*
4. *for all $\sigma \in \Sigma'$ with $\sigma \equiv \mathcal{A} \rightarrow B$ there is no proper subset $\mathcal{A}' \subset \mathcal{A}$ such that $\Sigma \models \mathcal{A}' \rightarrow B$.*

A set of FDs is called minimal *if it is a minimal cover of itself.*

Remarks. The assumption of minimal covers is no proper restriction because for a set of FDs a minimal cover can be constructed in polynomial time, see, for example, [Mai80].

For brevity, after ensuring that a set of FDs is minimal, we recombine FDs with the same left-hand side. More precisely, (sub-)sets of FDs of the form $\{\mathcal{A} \rightarrow B_1, \mathcal{A} \rightarrow B_2, \dots, \mathcal{A} \rightarrow B_n\}$ are represented as $\mathcal{A} \rightarrow B_1B_2 \dots B_n$.

2.3 Queries and Evaluation

Since we use a logic-oriented framework for relational databases, we express database queries in terms of the relational calculus. The relational calculus is basically a fragment of first-order logic without function symbols. In addition to the syntactic material introduced in Section 2.1 (constants and relation symbols as predicates), for expressing database queries we also need *variables* and *logical connectives* (Boolean operations). In our framework, the infinite set of variable identifiers will be denoted by Var and the set of logical connectives is assumed to be $\{\neg, \wedge, \vee\}$ (all other connectives like $\Rightarrow$ and $\Leftrightarrow$ can be expressed using this set of connectives). In the following, we consider subsets of the relational calculus as query languages; thus, a database query is always expressible as a formula of the relational calculus. To distinguish easily between constants and variables in logical formulas, identifiers beginning with a lowercase letter or a number denote constants, whereas identifiers beginning with an uppercase letter denote variables hereafter.

A user sending a query to the database expects this query to be answered. The most intuitive way of answering a query is an evaluation of the representing formula in the actual database instance. In doing so, we have to distinguish between closed and open queries. A *closed query*, usually denoted by Φ, does not contain free variables and is thus either true or false in a database instance. Consequently, the simplest way of evaluating a closed query is to return its truth value with respect to the database instance. An alternative way is to return the query itself (in case it is true in the database) or the negation of the query (otherwise). We formally describe this answering mechanism in the following definition.

Definition 2.2 (Ordinary evaluation of closed queries). *The* ordinary query evaluation eval *is a function with a closed query* Φ *from a suitable query language* $\mathscr{L}$ *as a parameter that maps a database instance on a truth value:*

$$\mathsf{eval}(\Phi)(r) := \begin{cases} \texttt{true} & \text{if } r \models_{\mathsf{M}} \Phi, \\ \texttt{false} & \text{otherwise.} \end{cases}$$

An alternative version of the ordinary evaluation, eval^**, maps the instance* r *on the version of* Φ *being true in* r*:*

$$\mathsf{eval}^*(\Phi)(r) := \begin{cases} \Phi & \text{if } r \models_{\mathsf{M}} \Phi, \\ \neg\Phi & \text{otherwise.} \end{cases}$$

In general, we will use this alternative version eval^* in the following for denoting the ordinary evaluation of a query.

In contrast to closed queries, *open queries* contain free variables and are denoted by $\Phi(\vec{V})$ where $\vec{V}$ is the vector[2] of free variables that occur in the query. If the actual vector of variables is not important in the context, we omit it and shortly denote the query with Φ. The idea of evaluating an open query is to find all variable assignments of $\vec{V}$ that makes the query true in the database instance. This leads to the following definition.[3]

Definition 2.3 (Ordinary evaluation of open queries). *Let $\Phi(\vec{V})$ be an open query. Then the* ordinary query evaluation eval^* *is defined by*

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

Note that, unlike for closed queries, for open queries eval^* returns "negative information" only implicitly: variable assignments that make a query $\Phi(\vec{V})$ false in the database instance are not returned; thus eval^* returns the empty set if $\Phi(\vec{V})$ is false in the database instance for every variable assignment.

In this thesis, we will also investigate "mixed scenarios" where closed queries as well as open queries occur (see, for example, Section 7.3). In such cases, we consider closed queries as specific open queries with an empty vector of free variables. Then for a closed query Φ the evaluation eval^* does not return Φ or $\neg\Phi$ as in Definition 2.2 but $\{\Phi\}$ or $\emptyset$, respectively.

Having introduced several database concepts in Sections 2.1–2.3, we now give an example that illustrates these concepts.

Example 2.4. Let a relation schema be given by $R_S = \langle R, \mathcal{U}, \Sigma \rangle$ with the set $\mathcal{U} = \{A, B, C\}$ of attributes and the set $\Sigma = \{\sigma_1, \sigma_2\}$ of local semantic constraints with

$$\sigma_1 \equiv A \rightarrow BC \text{ and } \sigma_2 \equiv C \rightarrow A.$$

Moreover, an instance r of R_S is given by

$$r = \{R(a,b,c), R(d,b,e)\}.$$

The FDs in Σ are both nontrivial. A as well as C are candidate keys of R_S, since $\Sigma \models A \rightarrow ABC$ and $\Sigma \models C \rightarrow ABC$ and A as well as C are minimal with this property. The super keys of R_S are all subsets of $\mathcal{U}$ containing A or C, that is, A, C, AB, AC, BC, and ABC.

R_S is in BCNF, since $\mathit{lhs}(\sigma_1) = \{A\}$ and $\mathit{lhs}(\sigma_2) = \{C\}$ are super keys. R_S is, however, not in ONF, since there is no unique key. Strictly speaking, Σ is not minimal

[2] The usage of a vector instead of a set indicates that we assume the free variables of an open query and thus also the attributes of a relation to be linearly ordered. Note that this assumption is only technical.

[3] For convenience, we overload the function eval^*. It should, however, always become clear from the context which variant of eval^* is meant.

because property 1 of Definition 2.1 is violated, but we may read Σ as abbreviation for the minimal cover $\Sigma' = \{A \rightarrow B, A \rightarrow C, C \rightarrow A\}$.

There are two tuples in the instance, $\mu_1 \equiv R(a,b,c)$ and $\mu_2 \equiv R(d,b,e)$, and the FDs from Σ are satisfied. Considering r as (Herbrand) interpretation, this is expressed by $r \models_{\mathsf{M}} \mu_1$, $r \models_{\mathsf{M}} \mu_2$, and $r \models_{\mathsf{M}} \Sigma$. The instance $r' = \{R(a,b,c), R(a,b,d)\}$ is not an instance of R_S, since σ_1 is violated (the tuples of r' agree on A but not on C).

With the relational calculus as query language, for example the following queries can be expressed:

$$\Phi_1 \equiv (\exists X_A)(\exists X_C)R(X_A,b,X_C)$$
$$\Phi_2(X_C) \equiv (\exists X_A)R(X_A,b,X_C)$$

The first query, Φ_1, is a closed query asking if there exists a tuple μ with $\mu[B] = b$. The second query, Φ_2, contains a free variable X_C and is thus open; it asks for all values $c_1, c_2, \ldots$ of C such that there exist tuples $\mu_1, \mu_2, \ldots$ with $\mu_i[B] = b$ and $\mu_i[C] = c_i$ for $i \in \{1, 2, \ldots\}$, respectively. According to Definitions 2.2 and 2.3, Φ_1 and Φ_2 are evaluated with respect to the instance r as follows:

$$\begin{aligned}
eval(\Phi_1)(r) &= \texttt{true} \\
eval^*(\Phi_1)(r) &= (\exists X_A)(\exists X_C)R(X_A,b,X_C) \\
eval^*(\Phi_2(X_C))(r) &= \{(\exists X_A)R(X_A,b,c), (\exists X_A)R(X_A,b,e)\}
\end{aligned}$$

Note that, strictly speaking, the second query should read $\Phi_2((X_C))$ rather than $\Phi_2(X_C)$. However, the vector braces in the argument of open queries are omitted hereafter for convenience. $\Diamond$

Chapter 3

Controlled Query Evaluation

A universal policy-driven approach to the inference problem in logic-oriented databases is *Controlled Query Evaluation (CQE).* Roughly, CQE checks for a user query whether the user would be able to infer a secret by employing his knowledge (which consists of the a priori knowledge the user has *before* sending queries to the database, and the answers to previous queries) and the correct answer to his query. If so, the answer to the query is suitably modified before being returned, thereby preventing the harmful inference.

In Section 3.1, we explain the basic idea of CQE and our notion of confidentiality policy on an informal level. In Section 3.2, we then present the formal framework for CQE.

3.1 Basic Ideas and Architecture

The general architecture of CQE is depicted in Figure 3.1 on the following page: Each database query is first answered ordinarily with respect to the database instance, but the result is sent to a *censor* rather than directly to the user. The censor inspects the query result, the policy that has been declared by a security administrator, and the assumed user knowledge which is also called *user log* in the following, and determines whether or not the query result may be returned to the user. In case the query result is considered harmful, it is suitably modified before being returned to the user. This modification can be performed according to different strategies: by *lying*, by *refusal*, or by a *combination* of both.

In this thesis, we focus on the modification strategy of refusal and on potential secrets as elements of the security policy (refer to Section 5.1 for the complete set of parameters). *Potential secrets* are declared independently of the actual database instance in a suitable language, that is, a fragment of first-order logic. If a declared potential secret is true in the actual database instance, the user must be prevented from disclosing this fact; otherwise, if the potential secret is false in the actual database instance, the knowledge of this fact is considered harmless.

A *confidentiality policy*, or *policy* for short, is a set of potential secrets each of which has to be protected in the environment the policy has been defined for. More

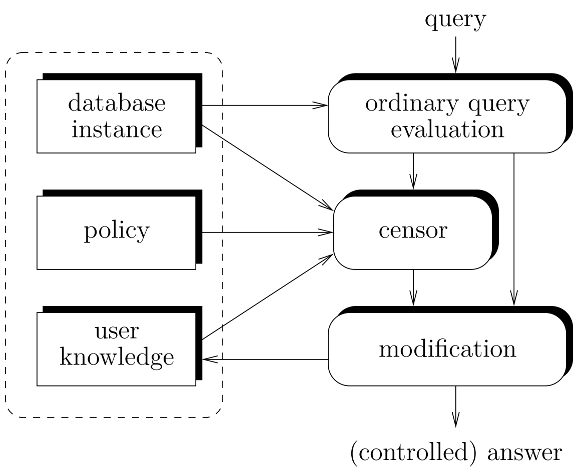


Figure 3.1: Schematic architecture of CQE.

precisely, regarding any declared potential secret of the policy, a user who is receiving (controlled) answers to his queries must at any time consider it possible that this potential secret is false in the actual database instance. In other words, for a query answering mechanism to be secure (in the sense of confidential), for each declared potential secret the following should hold:

> There exists a database instance that, on the one hand, is "compatible" with both the a priori knowledge of the user and the answers the user has received so far and, on the other hand, makes the potential secret under consideration false.

Hereafter, we denote policies by $\mathit{pot\_sec}$, signifying that policies consist of potential secrets.

3.2 Formal Framework

In order to investigate CQE on a formal level, we need to represent the above sketched concepts in a suitable framework. In the following, we shortly summarize some elaborations of Biskup and Bonatti [BB04a, BB04b], who developed CQE for different combinations of parameters.

Given a (possibly infinite) query sequence $Q = \langle \Phi_1, \Phi_2, ... \rangle$ and the a priori user knowledge log_0, CQE is a function that maps the database instance r and the confidentiality policy $\mathit{pot\_sec}$ on a sequence of pairs of answers and user logs:[4]

$$\mathit{cqe}(Q,\mathit{log}_0)(r,\mathit{pot\_sec}) := \langle(\mathit{ans}_1,\mathit{log}_1),(\mathit{ans}_2,\mathit{log}_2),\ldots\rangle \tag{3.1}$$

For the modification strategy of refusal, the answers as well as the user logs are determined subject to the following censor function which maps a policy, a user log, and a query on a truth value:

$$\begin{aligned}&\mathit{censor}(\mathit{pot\_sec},\mathit{log},\Phi) := \\ &\quad (\text{exists } \Psi)[\Psi \in \mathit{pot\_sec} \text{ and } ((\mathit{log} \cup \{\Phi\} \models \Psi) \text{ or } (\mathit{log} \cup \{\neg\Phi\} \models \Psi))]\end{aligned} \tag{3.2}$$

Basically, the censor checks whether the query or the negated query together with the user knowledge implies a potential secret. If so, the censor returns `true`, and otherwise it returns `false`. The censor has to check both the query and the negated query to block *meta-inferences* as explained in the following: Given an instance r, the censor has to check in any case, whether $\mathit{eval}^*(\Phi)(r) \in \{\Phi, \neg\Phi\}$ implies a potential secret in order to determine whether or not the answer must be modified. Now suppose, the censor *only* checks $\mathit{eval}^*(\Phi)(r)$ and *not* the complement $\neg\mathit{eval}^*(\Phi)(r)$; then the censor decision `true` would indicate that the honest answer to Φ implies a potential secret. If the policy is known to the user, the truth value of Φ in r might now be reconstructed by the user if only one of $\{\Phi, \neg\Phi\}$ implies a potential secret. Therefore, censor checks the query as well as its complement, making it impossible for the user to distinguish instances in which Φ is true from instances in which Φ is false.

With the censor function (3.1), answers and user logs are defined by

$$\begin{aligned}\mathit{ans}_i := \;&\texttt{if}\;\; \mathit{log}_{i-1} \models \mathit{eval}^*(\Phi_i)(r)\\ &\texttt{then}\;\; \mathit{eval}^*(\Phi_i)(r)\\ &\texttt{else}\;\; \texttt{if}\;\; \mathit{censor}(\mathit{pot\_sec},\mathit{log}_{i-1},\Phi_i)\\ &\qquad\;\; \texttt{then}\;\; \texttt{mum}\\ &\qquad\;\; \texttt{else}\;\; \mathit{eval}^*(\Phi_i)(r),\end{aligned} \tag{3.3}$$

with the special return value `mum` denoting a refusal, and

$$\begin{aligned}\mathit{log}_i := \;&\texttt{if}\;\; \mathit{censor}(\mathit{pot\_sec},\mathit{log}_{i-1},\Phi_i)\\ &\texttt{then}\;\; \mathit{log}_i\\ &\texttt{else}\;\; \mathit{log}_i \cup \{\mathit{eval}^*(\Phi_i)(r)\},\end{aligned} \tag{3.4}$$

respectively.

Regarding ans_i, it is first checked whether the user already knows the answer to his query, and if so, the (honest) answer is returned without further tests. This “improved

[4]At this point, we do not impose further restrictions on $\Phi_1, \Phi_2, \ldots$, log_0, and the elements of $\mathit{pot\_sec}$, but only suppose it to be expressed in a suitable fragment of first-order logic.

refusal" is necessary, since otherwise $log \cup \{\neg eval^*(\Phi)(r)\}$ would be inconsistent and thus imply every formula—especially the potential secrets, see [BB04b]. As a result, the answer would be refused although the user already knows it. If the user does not already know the answer to his query, the censor is invoked and, dependent on its decision, the refusal value `mum` or the honest answer is given.

The user log only needs to be updated if the censor decision is `false`. The "new" log is then defined as the union of the "old" log and the answer to the query. Otherwise, if the censor decision is `true`, meaning that the answer is refused, the user knowledge remains unchanged.

Hereafter, we assume that the *a priori user knowledge* log_0 is always true with respect to the considered instance r, that is, $r \models_M log_0$. Otherwise, availability would possibly become rather restricted, since even harmless answers could, together with the user log, lead to inconsistencies and thus to the disclosure of potential secrets, and would therefore be refused.

We now recall two important properties of the user log from [BB01], basically saying that the user log is always true in the actual database instance and never implies a potential secret.

Proposition 3.1 (Properties of the user log). *Consider a database instance r, a query sequence $Q = \langle \Phi_1, \Phi_2, ... \rangle$, a confidentiality policy pot_sec and the a priori user knowledge log_0 with $r \models_M log_0$. Then the user logs log_i with $i \in \{1, 2, ...\}$ that are produced by $cqe(Q,log_0)(r,pot\_sec)$ satisfy the following properties:*

$$r \models_M log_i; \tag{3.5}$$

$$log_i \not\models \Psi \text{ for all } \Psi \in pot\_sec. \tag{3.6}$$

It has been shown by Biskup and Bonatti [BB01] that CQE as defined in (3.1) with the censor (3.2) is a secure query answering mechanism according to the notion that has been outlined in Section 3.1. We omit the precise definition of security here; note, however, that it basically corresponds to Definition 5.4 in Section 5.2.

In the subsequent chapters, we investigate CQE for relational databases. Consequently, the detection of inferences by the censor function corresponds to the decision problem of implication in first-order logic, which is known to be undecidable in general. Therefore, CQE for relational databases is usually restricted to suitable fragments of first-order logic to guarantee the decidability of logical implication.

Chapter 4

Related Research

Security in databases has been investigated from different perspectives by many researchers. Consequently, most textbooks on computer security also deal with several aspects of database security; see, for example, [Den82, Gol06, Bis09].

A comprehensive overview of the history and ongoing research in the field of database security can also be found in the survey of Bertino and Sandhu [BS05]. Focusing on the requirement of confidentiality, the authors discuss techniques for relational databases such as discretionary and mandatory access control. Moreover, features for advanced systems are addressed, for example, flexible authorization models, credential-based user specification mechanisms, information dissemination strategies, and distributed cooperative data modification. The authors exemplarily sketch some of these features for object-based database systems and access control systems for XML. Furthermore, they discuss data management techniques that are suitable for preserving privacy, for example, data anonymization, data mining, and database systems being tailored to support privacy policies.

Moreover, design processes for secure databases have been proposed, for example, by Fernández-Medina and Piattini [FP05] for the multilevel secure database model—an extension of the classical relational database model. The authors identify four stages of a secure design process: requirements gathering, database analysis, multilevel relational logical design, and specific logical design. Moreover, Oracle Label Security [$J^{+}09$] is proposed as a technical means for implementing security specifications. Although being tailored to mandatory access control, the process being discussed by Fernández-Medina and Piattini may also be helpful regarding other security mechanisms.

In the following, we take a closer look at two research directions of database security: We briefly and exemplarily outline some results from the fields of access control (Section 4.1) and inference control (Section 4.2) in databases. Roughly speaking, in general, access control approaches control the disclosure of data, whereas inference control approaches control the disclosure of information. In Section 4.3, we then shortly present our contributions to Controlled Query Evaluation in relational databases and describe how they are integrated into this thesis.

4.1 Access Control

The security concept of access control has been a topic of research for decades. Many varieties of access control have been investigated so far, making it nearly impossible to give a complete overview of this area. We therefore only exemplarily discuss some of the results in the following subsections which represent, in our view, the important research directions in the field of access control in databases.

This overview is structured into the subsections "Discretionary and Mandatory Access Control" which covers "classical" access control for relational and object-oriented databases (Subsection 4.1.1), "Query Modification" which addresses a basic technique for enforcing access control (Subsection 4.1.2), and "Access Control Models and Mechanisms" which considers access control from a more general point of view (Subsection 4.1.3). We will use the abbreviations DAC and MAC for discretionary and mandatory access control, respectively.

4.1.1 Discretionary and Mandatory Access Control

An early approach to DAC in relational databases is the one of Griffiths and Wade [GW76]. The authors investigate granting and revocation of privileges on the table level of the relational database management system System R. They use a graph-based representation of the granted privileges for every user of the system and discuss a recursive revocation semantics.

MAC is often treated under the notion of multilevel security (MLS); consequently, database systems enforcing MAC are also called MLS databases. One of the first comprehensive MLS database systems, referred to as the SeaView model, is developed by Lunt et al. [LDS$^+$90]. On the one hand, this system uses a reference monitor to enforce a MAC policy; on the other hand, DAC policies can be expressed by the individual users of the system. Moreover, the authors discuss and formalize data consistency rules and propose an implementation of their system which allows to express the multilevel relations as views over single-level relations.

Since the representation of multilevel relations (which only exist at the logical level in general) as single-level relations is often needed in MLS databases, Jajodia and Sandhu [JS91a] propose an improved algorithm for decomposing and recovering multilevel relations. The authors describe the drawbacks of previous decomposition and recovery algorithms, develop a formal operational semantics for update operations on multilevel relations, and outline a decomposition algorithm by means of these operations which decomposes a multilevel relation into a set of single-level relations. In addition to this, Jajodia and Sandhu describe a corresponding recovery algorithm that constructs a multilevel relation from the single-level relations, and they propose some options for extensions of the algorithms.

In [JS91b], Jajodia and Sandhu point out that there is no exact model for MLS databases so far. In order to lay the foundations for such a model, the authors investigate integrity constraints and update operations for multilevel relations.

In [SJ92], Sandhu and Jajodia address the issue of polyinstantiation in MLS databases. Polyinstantiation occurs when two (or more) tuples with the same primary key values but different non-key values are present in the database. This inconsistency may be intentional in order to implement cover stories, that is, different views on the database for users with different clearances. Sandhu and Jajodia distinguish the notions of entity polyinstantiation and element polyinstantiation, explain how polyinstantiation can be completely prevented, and discuss the PCS (polyinstantiation for cover stories) semantics for polyinstantiation which basically provides a natural and intuitive way for implementing cover stories.

Besides relational databases, also more complex information systems like object-oriented databases (OODBs) have been investigated in the context of access control; see Olivier and von Solms [OvS94] for a comprehensive taxonomy that captures many issues relevant to modeling secure OODBs. In OODBs, data is organized in object-oriented structures, that is, classes, objects (with attributes and methods), and links between them. As an approach to multilevel security in OODBs, Cuppens and Gabillon [CG98] investigate the assignment of security levels to classes, class attributes, objects, object attributes, attribute values, methods, and inheritance links. They compile a list of 25 rules, each of which either describes the semantics of a security level assignment or is an inference rule. For example, the existence of a class attribute implies the existence of the corresponding class; therefore, the security level of the class attribute must dominate the security level of the class in order to avoid harmful inferences.

In [CG99], Cuppens and Gabillon develop a logical model for MLS databases. They propose a proof-theoretic three-layer approach: On the first layer, non-protected relational databases are formalized using a fragment of first-order logic with equality. On the second layer, they extend their model in order to express classification levels, the partial ordering between them, the actual classification of data, and rules for controlling inferences which are similar to the ones in [CG98]. Moreover, the authors discuss cover stories (see also [CG01]) as an ambiguity-free alternative to polyinstantiation. On the third layer, the MultiView layer, the model is extended in order to express which part of the database may be observed by which user. Finally, Cuppens and Gabillon adapt their investigations for object-oriented databases and sketch some implementation principles for MultiView object-oriented databases.

4.1.2 Query Modification

A further approach to enforcing access control in relational database systems is proposed by Stonebraker and Wong [SW74]. They introduce the technique of query modification which basically extends relational database queries (expressed in a high level language) by additional conditions such that the resulting queries never allow the user to access protected data when answered ordinarily. The authors investigate select-project-queries as well as aggregate queries. Many applications of query modification have been proposed in literature, for example, by Power et al. [PSPS05] for the scenario of sensitive patient data in medical information systems.

The enforcement of MAC by query modification is investigated by Keefe, Thuraisingham and Tsai [KTT89]. They first propose a strategy which stores the conditions that are used by the query modification algorithm in relational database tables. In order to take information about the user's state into account (for example, answers to previous queries), the authors propose a logic-based strategy which represents the conditions as logical rules and maintains a rule base which is accessed during query processing. To make their approach more efficient, Keefe, Thuraisingham and Tsai finally propose a graph-based strategy for representing and processing the conditions for query modification.

4.1.3 Access Control Models and Mechanisms

A security model expresses a security policy on a formal level rather than in natural language in order to avoid ambiguities and inconsistencies (see Gollmann [Gol06]). Many security models for access control and corresponding mechanisms for enforcing the expressed policies have been investigated so far, an overview of which can be found in the following papers.

An investigation of access control models focusing on information flow can be found at Sandhu [San93]. After introducing Denning's basic notion of information flow policies, he describes and compares models that base on security labels being arranged in a lattice structure. While the Bell-LaPadula model focuses on the preservation of confidentiality, the Biba model gives priority to the integrity of information. Sandhu also discusses combinations of the two models. Finally, as a prominent example for dynamic access rights, the Chinese Wall model is addressed.

Samarati and De Capitani di Vimercati [SD01] give a comprehensive overview of access control policies, security models and mechanisms for the enforcement of access control policies. Their work covers basic mechanisms for implementing DAC like the Harrison-Ruzzo-Ullman model, drawbacks of DAC approaches, MAC security models like the Bell-LaPadula model and the Biba model, combinations of DAC and MAC like the Chinese Wall model, extensions for the basic DAC policies, role-based access control (RBAC), and advanced models involving logic-based policy languages, composition of policies, and certificate-/credential-based models.

Nicomette and Deswarte [ND96] develop an adaption of the Bell-LaPadula security model for distributed object systems. In such a system, an object interacts with another object by invoking a method of this object. Nicomette and Deswarte describe the drawbacks of the original Bell-LaPadula model—in particular its restrictiveness—

and show that their adaption is less restrictive than the original model.

A general access control mechanism that is based on Ordered Logic with ordered domains is proposed by Bertino et al. [BBFR00]. The authors consider the access from subjects to objects according to privileges. These privileges are declared by other subjects or they are derived according to hierarchies or derivation rules. Subjects can be individual users, hierarchically ordered groups or hierarchically ordered roles; moreover, also objects and privileges are assumed to be organized as hierarchies. In contrast to many others models, in the approach of Bertino et al. it is possible to express negative authorizations, that is, prohibitions. Sets of explicit authorizations and rules for the derivation of implicit authorizations are expressed as logic programs and access decisions are taken using the stable model semantics of Datalog programs. Furthermore, the authors address the issues of conflicting rules and administrative authorizations, that is, "privileges for granting privileges".

Jajodia et al. [JSSS01] develop a similar approach: the Flexible Authorization Framework (FAF). This framework allows for the enforcement of multiple access control policies in a single system, thereby considering positive and negative authorizations as well as propagation of authorizations along hierarchies. Starting with the description of the system components—objects, users, groups and roles—and their hierarchies, Jajodia et al. propose a logical authorization specification language for expressing explicit authorizations and rules for deriving implicit authorizations. Moreover, rules for conflict resolution and integrity preservation can be formulated. Due to several restrictions, an authorization specification (interpreted as logic program) is guaranteed to be locally stratified and thus to have a unique stable model. Therefore, for every access request a unique decision can be made whether or not to permit this request.

4.2 Inference Control

Unlike access control (which usually aims at preventing direct accesses to sensitive data), inference control also takes indirect accesses into account which may arise when combining several non-sensitive pieces of data, possibly including background knowledge. Being originally investigated for statistical databases, inference control is nowadays a widespread research topic, as is evident from the survey of Farkas and Jajodia [FJ02]. In the following, we exemplarily outline some of the contributions to research on inference control. We are conscious that a lot of general research has been performed in the field of inference control, for example, Halpern and O'Neill [HO08] propose a framework for studying the notion of secrecy (being strongly related to most inference control requirements) in general multiagent systems; however, we do not take such general research into account, but focus on database-related research in the following.

This section is structured as follows: In Subsection 4.2.1, we address inference control in statistical databases; in Subsection 4.2.2, we compile some research results on inference control in relational databases; and in Subsection 4.2.3, we discuss the contributions to the field of Controlled Query Evaluation in detail—a particular form of inference control in databases which serves as a basis for the investigations in this thesis.

4.2.1 Statistical Databases

Inference control has its roots in the research area of statistical databases. These databases aim at providing statistical data about individuals on the one hand and protecting the privacy of these individuals on the other hand. Denning and Schlörer [DS83] show that in the context of statistical databases, sensitive information about individuals may be inferred when collecting and correlating enough statistical data. They give an overview of inference controls that work against such unwanted disclosures. Basically, these inference controls can be divided into restriction techniques which withhold certain parts of the statistics and perturbation techniques which prevent unwanted inferences by adding noise to the statistics.

Kenthapadi, Mishra and Nissim [KMN05] investigate the online auditing problem for statistical database queries, that is, given a sequence of queries and their answers, deciding whether or not the true answer to the current query enables the querying user to breach privacy. Besides the classical notion of a privacy breach (a user is able to learn a single data element in full), the authors propose a more sophisticated notion by using probability theory: A privacy breach occurs if for a data element and a small interval the a priori probability for the element falling into the interval is not equal to the a posteriori probability (after having received the answer to the posed query). Since existing offline algorithms prove insufficient to prevent privacy breaches, Kenthapadi et al. propose the online technique of simulatable auditing. Basically, an auditor (that is, a mechanism that answers the user queries) is called simulatable (by the querying user) if its decision whether or not to give the true answer is independent of the actual data and the (correct) answer to the current query. The authors propose simulatable algorithms for max queries (that is, queries asking for the maximum of a set of data elements) for both the classical notion and the probabilistic notion of a privacy breach; moreover, they show that these algorithms are “useful” in the sense of availability of data. A derivative of simulatable auditing which is called simulatable binding has been proposed by Zhang, Jajodia and Brodsky [ZJB08]. Simulatable binding improves simulatable auditing in that it yields a yet higher data availability.

Sweeney [Swe02] proposes the k-anonymity protection model for statistical databases. To avoid the problem that tuples of two databases can be linked in order to uniquely identify individuals, she introduces the notion of a quasi-identifier and the k-anonymity requirement. A quasi-identifier is a set of attributes that possibly enables to identify an individual by linking a tuple with external information; k-anonymity demands that each instantiation of a quasi-identifier occurs at least k times in the database, thereby preventing the unique identification of individuals. Sweeney also describes three attacks against k-anonymity and proposes solutions for these attacks.

Machanavajjhala et al. [MGKV07] point out two further attacks against k-anonymous tables. The homogeneity attack exploits a lack of diversity in the values of the sensitive attributes: If a sensitive attribute has the same instantiation in each tuple in a set of tuples with the same quasi-identifier instantiation, then sensitive individual information can be derived. The background knowledge attack exploits the same lack of diversity by utilizing personal knowledge that is not explicitly represented in the database. The authors therefore propose the principle of ℓ-diversity; it basically demands that a sensitive attribute must at least have ℓ different instantiations in a set of tuples with the same quasi-identifier.

Li, Li and Venkatasubramanian [LLV07] on the one hand justify why ℓ-diversity is sometimes difficult and unnecessary to achieve and on the other hand describe two attacks against ℓ-diversity: the skewness attack, exploiting skew probability distributions of sensitive attributes, and the similarity attack, exploiting semantic similarity of attribute values. They propose the principle of t-closeness to thwart these attacks. Intuitively, a set of tuples has t-closeness if the distribution of a sensitive attribute in this set and the distribution of the same attribute in the whole database table have a distance of at most t. Moreover, the authors discuss several distance measures for probability distributions.

4.2.2 Relational Databases

Some inference control approaches for relational databases are, in fact, extensions of access control mechanisms. For example, Brodsky, Farkas and Jajodia [BFJ00] present the Disclosure Monitor (DiMon) which extends a standard MAC mechanism for relational databases with the Disclosure Inference Engine (DiIE): If a user query must not be answered according to the MAC policy, the query is rejected immediately; otherwise, the information that can be derived from previous queries, the current query and the (database) constraints the user is aware of is computed by the DiIE and again checked by the MAC mechanism. Only if also this second check yields a positive result, the query is answered; if the check fails, the query is rejected. The authors consider select-project-queries on a single relation as user queries and Horn clauses as database constraints which in particular allow to express functional, multivalued and join dependencies. The DiIE can work in two different modes: In the data-dependent mode, the derived information is computed using previous queries, the corresponding answers and the constraints; in the data-independent mode, only the previous queries and the constraints are considered. Brodsky et al. propose algorithms for both modes that are sound and complete, thereby proving the underlying inference problems decidable. Moreover, the data-dependent mode is shown to achieve a higher availability at the expense of higher space complexity (since it has to maintain a history file for queries *and* answers), whereas the data-independent mode is less space consuming at the expense of availability and the restriction that queries and constraints must not involve constants.

Farkas, Toland and Eastman [FTE01] propose the Dynamic Disclosure Monitor (D^2Mon) as an extension of the data-dependent mode of the DiMon approach. Unlike DiMon, the D^2Mon approach takes database updates into account. It therefore integrates two additional components into the DiMon architecture: the Update Consolidator for the propagation of updates to the user's history file and the Cardinality Inference Detection which aims at detecting inferences due to combinatorial effects. For the Update Consolidator, the authors present a sound and complete algorithm.

Su and Özsoyoglu [SÖ91] propose a technique that "re-classifies" an MLS database in order to prevent inferences. In contrast to the DiMon approach, this technique requires a preprocessing of the database, but afterwards it only needs to perform computationally cheap access control. The authors investigate inferences by functional and multivalued dependencies (FDs and MVDs, respectively). They address the problem that unauthorized information may be derived if integrity constraints (like FDs and MVDs) are not properly reflected by the classification of the database. Su and Özsoyoglu show how FDs and MVDs can be exploited for such inferences. For both kinds of dependencies they develop an algorithm that adjusts the classification levels of the database in order to prevent such exploitations.

A similar but more general approach can be found at Stickel [Sti94]. Given a classification of objects (in the sense of MAC) and a set of inference rules, he shows how to obtain a classification of the objects that is free of inference channels by upgrading single classifications to higher levels. Stickel proposes a logical formulation of this classification problem and adapts the Davis-Putnam procedure for theorem proving in order to find minimal solutions to the problem. In this context, minimal means that no object is classified higher than necessary. Moreover, Stickel discusses the option of assigning costs to upgrade possibilities in order to control the selection of upgrades by the proposed algorithm.

Dawson, De Capitani di Vimercati and Samarati [DDS99] consider existing non-MLS databases and address the task of classifying these databases in the sense of MAC under explicit consideration of inference constraints. This classification should on the one hand be correct in terms of classification constraints declared by the security administrator, and on the other hand it should be minimal in the sense that no database entry is overclassified. The authors propose an algorithm that takes a set of classification constraints as input and computes a minimal correct classification assignment. Moreover, an algorithm is given that labels the entries in an actual database according to such a classification assignment.

Besides the extensions of MAC, many others approaches to inference control in relational databases have been proposed. For example, Delugach and Hinke [DH96] develop the “Wizard” system for detecting inferences in relational databases. The mechanism is designed as a tool for security administrators and aims at statically detecting possible inferences before a user sends queries to the database system. The information that is represented by the database is viewed at on three layers, each of which is further divided into so-called facets. The authors consider inference paths in single facets as well as inter-facet inference paths. Moreover, the identified inferences are graded by there severity. Delugach and Hinke finally describe the implementation of their system.

Yip and Levitt [YL98a] investigate data level inferences in relational databases. They develop an inference detection system with a set of five inference rules that can be applied to query answers in order to derive additional information. Unlike previous approaches, these rules operate on the data without taking schema information such as functional dependencies into account. Moreover, union queries are considered which are basically sequences of single queries. The authors also implement their inference detection system and identify conditions for a feasible runtime, for the considered problem is NP-hard in theory. In [YL98b], Yip and Levitt extend their system by an additional inference rule and elaborate on the application of inference rules to union queries in detail. They also sketch the prototypical implementation of their approach.

As a generalized approach to schema level inference detection, Hale and Shenoi [HS96] investigate fuzzy FD inference in relational databases. They introduce fuzzy tuples as generalization of classical relational tuples and develop a model in which fuzzy as well as classical relational databases can be considered. Using the notion of redundancy between fuzzy tuples, the authors define fuzzy FDs and the fuzzy counterparts of Armstrong’s axioms for computing inference closures. Moreover, they generalize the notion of FD-based inference in order to cover deductive and abductive FD reasoning in relational databases. As an application example, Hale and Shenoi describe a scenario in which a classical database is augmented by a fuzzy relation in order to express common knowledge.

Chang and Moskowitz [CM00] propose a probabilistic framework for analyzing inferences in databases. They introduce the notion of similarity between database attributes and propose the dispersion as a similarity measure. Similar attributes carry about the same information and might thus assist in drawing inferences. Moreover, Chang and Moskowitz model correlations between the attributes of the database by means of Bayesian networks and describe the techniques of blocking and aggregation in order to reduce the information and thus the possible inferences in a database.

Another facet of inference control is addressed by Chen and Chu [CC08]. They consider the case of collaborative inference attacks in databases. Their probabilistic framework contains a Semantic Inference Model (SIM) which is instantiated to a Semantic Inference Graph (SIG). The SIM captures three kinds of information: data dependencies, schema dependencies and domain-specific semantic knowledge. The SIG then instantiates the SIM with the specific entity instances of the actual database. Chen and Chu suggest to map the SIG to a Bayesian network in order to analyze inferences. If the answer to a user query enables the user to infer sensitive information in terms of a specified threshold, the answer is refused. Regarding the collaboration among users, the authors identify three parameters that affect the effectiveness of collaboration: authoritativeness, that is, accuracy of information; honesty, that is, honesty level and willingness of the knowledge provider; and fidelity, that is, effectiveness of the communication between the collaborating users. Furthermore, collaboration with overlapping inference channels as well as collaboration with non-overlapping inference channels is taken into account.

Rather than developing specific inference control mechanisms, some authors focus on providing formal models for the investigation of inferences. For example, a more general approach to inference control or, more precisely, inference detection, can be found in the work of Stouppa and Studer [SS07]. They investigate the data privacy problem (being strongly related to the detection of inferences) in a formal framework which assumes that public knowledge is expressed by means of a view instance (consisting of queries and the corresponding answers) and an ontology (representing background knowledge); the data privacy condition is given as a set of queries. Intuitively, privacy is preserved if none of the queries in the privacy condition can be inferred from the view instance and the ontology. Besides the general framework, Stouppa and Studer investigate the data privacy problem for relational databases and ontologies based on the description logic $\mathcal{ALC}$.

Cuenca Grau and Horrocks [CH08] consider logic-based information systems which can be seen as a generalization of many kinds of information systems such as relational databases. For these generalized information systems, the authors formulate three different privacy problems. Moreover, on the one hand, Cuenca Grau and Horrocks recall the probabilistic frameworks of Miklau/Suciu [MS07] and Deutsch/Papakonstantinou [DP05] in which privacy conditions can be expressed; on the other hand, they formalize privacy in a logic-based framework. Furthermore, connections between the presented frameworks are pointed out.

4.2.3 Controlled Query Evaluation

Controlled Query Evaluation (CQE) is a particular form of inference control in (logic-based) databases that enforces a security policy by inspecting the assumed user knowledge before answering a query. If necessary, the answer to the query is then suitably modified in order to preserve confidentiality. CQE is inspired by the work of Sicherman, de Jonge and van de Riet [SdJvdR83] and Bonatti, Kraus and Subrahmanian [BKS95]. In the following, we first sketch the contributions of these two papers. After that, we describe the development of CQE from the first approaches to the more recent results.

Sicherman, de Jonge and van de Riet [SdJvdR83] investigate how information can be kept secret in a question-answering system when using a refusal strategy. The authors consider propositional databases and sentences (being either true or false in the database) as queries and secrets. The aim of the system is to prevent the user from disclosing the actual truth value of a secret in the database—even if he exploits his usage history and further information about the functionality of the system. The authors systematically identify the situations in which the system should refuse to answer in order to protect the secrets. They propose one solution for the case that the user knows the set of secrets (but, of course, not their actual truth value in the database) and another solution for the case that the user does not know the secrets. Moreover, it is formally shown that both solutions prevent the user from learning secrets.

Bonatti, Kraus and Subrahmanian [BKS95] enforce security in deductive databases (which are represented as finite sets of propositional formulas) by means of a lying algorithm. In order to formalize their approach, they propose two modal extensions of propositional logic: The first extension, logic of secrecy, introduces modal operators for declaring formulas as secrets; the second extension, logic of interaction, adds further modal operators to the logic of secrecy in order to describe the communication between user and database. Moreover, the authors give a formal model-theoretic semantics for the logics of secrecy and interaction and propose an algorithm that preserves security in terms of the underlying logics. This algorithm basically gives the correct answer to a query if this answer together with the user's beliefs does not imply the disjunction of the declared secrets; otherwise, the negated answer to the query is returned. Bonatti, Kraus and Subrahmanian additionally show that the proposed algorithm is in the same complexity class as ordinary query evaluation.

Biskup [Bis00] compares the strategies of lying and refusal in a common framework. He considers structures that interpret the symbols of an appropriate logic as database instances and sentences of this logic as queries. Moreover, a policy is a set of pairs of complementary sentences, the so-called secrecies; a database user must not learn which part of such a pair is actually true in the database instance. Biskup develops an architecture which decides by means of the answer to a user query and the assumed user knowledge whether or not the answer to this query is harmful regarding the declared policy. If so, it is suitably modified before being delivered to the user. The modification can either be a lie, that is, the answer is negated, or a refusal, that is, the answer is the special return value `mum`. Biskup shows that (for most situations) the refusal strategy more often returns the true answer to a query than the lying strategy if the policy is unknown to the user. He therefore suggests to prefer refusal over lying in general.

Biskup and Bonatti [BB01] extend the investigations of [Bis00] to policies that consist of potential secrets. A potential secret is also a sentence in the underlying logic but, unlike a secrecy, a potential secret needs only to be kept secret from the user in case it is true in the database instance; otherwise, if it is false in the instance, the user may be aware of this fact. Biskup and Bonatti again consider the strategies of lying and refusal and assume that the policy is known to the user. They show that, provided the potential secrets in the policy are closed under disjunction, lying and refusal basically deliver the same information to the database user.

In [BB04b], Biskup and Bonatti propose the combined method for CQE under known policies. While the approaches discussed so far either always lie (uniform lying) or always refuse (uniform refusal) when a modification of an answer is required, the combined method uses a combination of lying and refusal. More precisely, as long as only the correct answer to a query can be exploited to infer a secret, the combined method returns a lie, that is, the negated answer; if, however, also the negated answer enables the user to infer a secret, then the combined method refuses to answer. Biskup and Bonatti formally show that this method is secure and compare it to the approaches of uniform lying and uniform refusal. It turns out that the combined method returns more correct answers than the uniform methods in general. Moreover, properties are identified under which the combined method and the uniform lying method or the combined method and the uniform refusal method coincide.

An overview of CQE in propositional databases can be found in [BB04a]. Biskup and Bonatti categorize the so far investigated approaches along the dimensions of confidentiality policy (potential secrets or secrecies), user awareness regarding policy (known or unknown) and enforcement method (lying, refusal or combined), thereby also identifying and filling the gaps.

In [BB07], Biskup and Bonatti investigate CQE for a fragment of first-order logic, thereby making the approach suitable for open queries in relational databases. In order to guarantee decidability, some restrictions must apply. More precisely, queries must be domain-independent and safe, and query language, policy language and a priori user knowledge must be suitably restricted so that all formulas are in the Bernays-Schönfinkel class (whose implication problem is known to be decidable). The controlled evaluation of an open query can then be simulated by the evaluation of a sequence of closed queries and the evaluation of a completeness sentence which basically captures the negative part of the answer, that is, the variable substitutions that make the query false in the database instance. Biskup and Bonatti propose a lying method and a refusal method for open queries, both assuming a fixed enumeration sequence of the constants which is known to the database user. Moreover, an alternative lying method and a combined method are investigated which consider the completeness sentence at the beginning of the query evaluation rather than examining it as late as possible. These alternatives, however, come at the price of cooperativeness. All of the proposed methods are proven confidentiality preserving.

Biskup and Weibert [BW08a] consider CQE with known potential secrets for incomplete propositional databases. In incomplete databases, queries may be answered by `true`, `false` or `undef` (in case the actual truth value of the query is unknown) which has to be reflected in the the CQE formalisms. It turns out to be appropriate to express the user knowledge in the modal logic S5. Moreover, the authors introduce the concept of a security configuration which expresses for a policy, the user knowledge and a query, which answers to the query would lead to the disclosure of a secret. Using these security configurations, requirements for the lying method, the refusal method and the combined method are elaborated; for each method, one security preserving censor is presented.

Regarding incomplete first-order databases, a first approach is made by Biskup, Tadros and Wiese [BTW10] who investigate CQE with known potential secrets for incomplete first-order databases. They use the so-called GFFD-databases as a restricted data model for first-order databases and the modal logic S5 for expressing user knowledge and policies. It is then shown that the resulting model can basically be reduced to the propositional case.

Biskup, Seiler and Weibert [BSW09] first investigate database updates in the context of CQE. They consider the lying approach for known potential secrets in propositional databases and define a view update as the change of the truth value of an atomic sentence (performed by a database user). Evaluated ordinarily, accepted view update requests as well as denied view update requests may lead to harmful inferences; thus, the authors propose an algorithm that performs inference-free view updates applying the lying method. This algorithm is proven confidentiality preserving, and moreover it is shown that each view update may be successfully undone if the undo operation is performed immediately after the update operation. A more general approach to updates within a client-server architecture, considering updates that are initiated by the client, updates that are initiated by the server, and transactions of update requests, is developed by Biskup et al. in [BGSW09].

Biskup and Wiese [BW08b] propose a preprocessing approach for the lying method of CQE under known potential secrets in propositional databases. Basically, an alternative database instance is constructed before the user sends a query to the database. This alternative instance possibly contains lies compared to the original instance, but user queries can be answered ordinarily with respect to the alternative instance without disclosing secrets. Therefore, the alternative instance is called inference-proof. Moreover, explicit availability policies are considered that consist of propositional formulas; in the inference-proof instance, as few of these formulas as possible should have a different truth value than in the original instance. Biskup and Wiese also develop an algorithm for constructing the alternative instance. This algorithm combines the techniques of SAT-solving and Branch and Bound. An extension of the results to the case of first-order databases is developed by Biskup and Wiese in [BW09].

Although for a suitable fragment of first-order logic CQE in relational databases becomes decidable (see [BB07]), pertinent algorithms suffer from high complexity in general. This is mainly due to the ever growing log file which has to be considered for every query evaluation. Therefore, several efforts have been made in order to abandon the log file. The results from this research direction form the basis for this thesis and are discussed in Section 4.3. A further result from this area, which is not addressed in the remainder of this thesis, has been achieved by Biskup et al. [BHLL10a]. Their approach applies for the refusal method under known potential secrets and under the presence of functional and full join dependencies. The authors identify a sufficient and necessary formal condition for the violation of the confidentiality policy. This condition characterizes a "forbidden structure" that can be represented by the hypotheses of a template dependency. If all of these hypotheses are satisfied, the user is able to infer a secret (which is represented by the conclusion of the template dependency). Consequently, the user must be prevented from fully discovering this forbidden structure which can be controlled by a suitable monitoring component. In contrast to previous approaches, however, it is not necessary to maintain a general log file for the user.

Wiese [Wie10] transfers CQE for propositional databases to a possibilistic logic setting: Rather than returning one of the values `true` or `false` (or `undef` in the scenario of incomplete databases) when answering a query, the "necessity degree" of the query in the knowledge base is computed and returned. This obviously allows for a finer-grained way of answering a query. In order to formalize her approach, Wiese uses the standard possibilistic logic SPL which basically labels each formula in a given knowledge base with a (lower bound for a) necessity degree, thereby inducing a possibility distribution; this possibility distribution, in turn, can be used to determine the necessity degrees of arbitrary formulas. Evaluation of a query then corresponds to the computation of the necessity degree of this query. A security policy is declared as a set of pairs of formulas and necessity degrees, each of which signifies that the user must not know that the formula is certain in the knowledge base at a necessity degree above the given one. In this framework, Wiese develops a method for the controlled evaluation of queries and proves it confidentiality preserving.

A recent survey on the achievements in the field of Controlled Query Evaluation from the more general perspective of a client-server model can be found at Biskup [Bis10].

4.3 Contributions of Previously Published Work

Parts of this thesis have been published in three conference papers and a journal article, which are my original work. All publications are co-authored by my advisor Joachim Biskup. His contribution comprised joint exploration of potential approaches, ongoing discussions, proof-reading, and general advisory. In the following, the contributions of the mentioned publications are shortly outlined.

- In [BL07], we lay the foundation for stateless inference control in relational databases. We explore a restricted scenario in which the declarative goals of inference control can be reached by an access control mechanism and sketch two efficient algorithms for the enforcement of these mechanism. The contributions of this article are elaborated in more detail in Chapter 6.

- In [BEL08], we investigate the impact of functional dependencies under select-project-queries and identify a situation in which again a reduction of inference control to access control is possible.

  This article, which serves as a basis for Section 7.1 of this thesis, is additionally co-authored by David W. Embley with whom we discussed several aspects of normalization theory.

- In [BLS09], some enhancements of stateless inference control are investigated, and we propose an algorithmic implementation for these enhancements. In Sections 7.2 and 7.4, these enhancements are reconsidered and the proofs of confidentiality preservation are elaborated.

  This article is additionally co-authored by Sebastian Sonntag who performed a first exploration of the enhancement possibilities for stateless inference control in his diploma thesis.

- In [BHLL10b], open queries are investigated in the context of stateless inference control with the result that the database user must not be aware of the confidentiality policy under open queries. Moreover, we sketch an algorithmic implementation of the resulting mechanism. A detailed elaboration of this enhancement including a formal proof of confidentiality preservation can be found in Section 7.3 of this thesis.

  This article is additionally co-authored by Sven Hartmann and Sebastian Link who contributed valuable ideas from the field of dependency theory to the original draft of the article, and with whom we discussed our results for open queries.

In this thesis, the results of the above publications are compiled and put into a common framework. Moreover, the gaps between the approaches are identified and filled, those confidentiality proofs that have only been sketched before are elaborated, and the limits of stateless inference control for relational databases in the developed framework are investigated. Finally, an algorithmic implementation of the comprehensive approach is provided.

Part II

Efficient Inference Control in Relational Databases

Chapter 5

Introduction to Stateless Inference Control

CQE in its original form, as introduced in Chapter 3, is stateful, that is, it is necessary to keep a user log in order to preserve confidentiality. More specifically, this user log contains the (assumed) a priori knowledge of the user and the answers to the previous queries. For each query, the CQE mechanism has to check whether the (unmodified) answer to the current query or its negation together with the user log allows for drawing harmful inferences.

From a practical point of view, however, the stateful CQE approach comes along with two major drawbacks: On the one hand, the user log is an ever growing file which, sooner or later, will lead to space problems; on the other hand, the inference check basically corresponds to the implication decision problem in first-order logic which is known to be undecidable in general or at least computationally complex in particular cases. Implementations usually have to rely on an external theorem prover and get the more costly the bigger the log file becomes. Consequently, our objective in this and the following chapters is to avoid the theorem prover and the ever growing log file by guaranteeing confidentiality preservation at the same time. In doing so, it will be necessary to suitably restrict the framework of stateful CQE, as introduced in Section 3.2.

5.1 Requirements, Parameters, and a Running Example

The stateless CQE mechanism that will be developed in the following, should especially take care of the following aspects:

Confidentiality. As already required for stateful CQE, also stateless CQE shall preserve confidentiality with respect to the declared confidentiality policy. Therefore, the notion of confidentiality preservation must be specified by a formal definition. We consider confidentiality preservation the primary goal of our approach, that is, other goals are possibly neglected if this is necessary to guarantee confidentiality preservation.

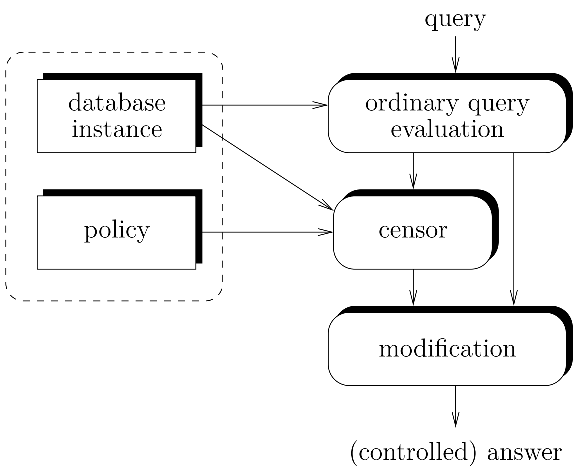


Figure 5.1: Schematic architecture of stateless CQE.

Availability. A database system is perfectly secure in the sense of confidentiality preservation if it refuses to answer at all. Clearly, from the perspective of a database user this behavior is not acceptable, since he is supposed to need information from the database in order to complete his task. We therefore require our mechanism to ensure the availability of information as long as this does not contradict the preservation of confidentiality.

Efficiency. Our mechanism shall be efficient in terms of complexity. More specifically, the decision whether or not the user is able to draw harmful inferences should no longer rely on the implication decision problem in first-order logic but rather on some kind of access control that is computationally feasible and efficiently implementable. Moreover, again for efficiency reasons, we want to avoid the log file of stateful CQE.

The general architecture of a stateless CQE mechanism is depicted in Figure 5.1. Observe that, compared to stateful CQE (as depicted in Figure 3.1), the censor of stateless CQE no longer needs the user knowledge as input.

Trying to reach all of these goals at the same time, we might encounter problems. For example, when abandoning the log file, a piece of information that is not confidential if considered isolatedly might help to disclose a secret afterwards: Suppose, the information that Smith has a salary of \$ 5,000 is to be kept confidential. In this context, the information that Smith is a manager seems to be harmless; if, however, it turns out later on that all managers get a salary of \$ 5,000 (which is, considered isolatedly, harmless as well), the seemingly harmless information helps to disclose the secret.

Because we want to eliminate the user log file from the CQE framework, our system has to be "oblivious" and consequently never (or at most to a certain degree) aware of the information the user has. We have to be well-prepared for users who want to exploit this property. More precisely, stateless CQE possibly has to modify more answers than stateful CQE in order to prevent users from drawing harmful inferences in the future. In the example sketched above, either the information that Smith has a salary of \$ 5,000 or the information that managers get a salary of \$ 5,000 should not be delivered to the user, although it seems to be harmless at first glance.

In the following, we sketch and justify the parameters that we assume for our investigations of stateless CQE. We divide these parameters into those that concern the underlying relational database, those that concern the users of the system, and those that concern the actual enforcement of inference control.

Parameters concerning the database

- We consider the domain of relational databases and therefore use a logic-oriented approach for our investigations, as introduced in Chapter 2. This approach follows the one elaborated by Abiteboul et al. [AHV95]. In particular, we assume (a fragment of) the relational calculus as query language.
- We suppose that in our scenario relational databases only contain complete information. In particular, we do not consider null values. From a logical perspective, we assume that formulas, which are not true in a database instance, are false in this instance. This assumption is also known as *closed world assumption*.
- We concentrate on single relation schemas and assume in the following that the database consists of the relation schema $\langle R, \mathcal{U}, \Sigma \rangle$, unless stated otherwise. In Section 11.2, we shortly address the issue of multiple relation schemas and constraints between them as a direction for future research.
- We consider only functional dependencies as local (intra-relational) semantic constraints. Although in database theory many other kinds of semantic constraints are investigated (like, for example, multivalued dependencies, join dependencies, and template dependencies), we consider functional dependencies the most prevalent kind of local semantic constraints in actual relational databases.
- We consider the actual contents of the database static, that is, we assume that the instance of the relation schema is constituted *before* a user sends queries to the database. In other words, we do not consider updates of the instance during query time.

Parameters concerning the database users

- Our scenario contains three actors:
  - The *database administrator* fills the database instance with content (possibly on behalf of third parties who act as “data owners”) and declares the semantic constraints;
  - the *security administrator* declares the confidentiality policy (possibly on behalf of third parties who act as “data owners”) which describes the information that should be kept confidential;
  - the *database user* sends queries to the system and receives (possibly modified) answers. Write accesses to the database are not considered. Moreover, we assume a single user, since collusion between different users is out of the scope of this thesis.

- We assume a “smart” database user, that is, a user who is aware of the declared semantic constraints Σ of the database and of the confidentiality policy. More specifically, he can use the functional dependencies for reasoning and is aware of the secrets being declared by the security administrator. Remember that these secrets are potential, that is, knowledge about a declared secret does not imply the actual truth value of this secret in the database instance. A policy that the database user is aware of is also called a *known policy* hereafter.

- We do not consider further knowledge, for example, about the contents of the database instance. Thus, given the relation schema $\langle R, \mathcal{U}, \Sigma \rangle$, the *a priori knowledge* of the database user log_0 is assumed to consist of the semantic constraints only, that is, $log_0 = \Sigma$.

Parameters concerning the CQE mechanism

- Our approach is language-based, that is, for the security administrator as well as for the database user a language is defined that declares the admissible secrets or the admissible queries, respectively.

- Our approach is policy-driven, that is, the security administrator declares the confidential information in form of a set of secrets, called confidentiality policy, or policy for short. This policy may be formulated independently of the actual database instance, which supports the functional separation of database administrator and security administrator.

- If necessary, query answers are modified by refusal, that is, for closed queries the special answer `mum` is returned rather than the ordinary query evaluation; for open queries the set of answers is suitably filtered, see Chapter 7.3. In particular, our approach never lies to the user but only answers his queries honestly or refuses to answer.

To illustrate our elaborations, in the subsequent chapters we will make use of an example from the banking context. In the following, we roughly sketch the setting of this running example.

Example 5.1. A group of banks employs a common relational database system to maintain the data of its clients. Within this database a relation schema $ACC_S = \langle ACC, \mathcal{U}, \Sigma\rangle$ is declared for the relation ACC, which captures for each combination of bank and account number the name and the balance of the respective account holder. Let the following attributes and local semantic constraints be declared:

$$\mathcal{U} = \{BANK, ACC\_NO, ACC\_HOLDER, BALANCE\}$$
$$\Sigma = \{\{BANK, ACC\_NO\} \rightarrow \{ACC\_HOLDER, BALANCE\}\}$$

An instance of ACC_S is given by:

acc	*BANK*	*ACC_NO*	*ACC_HOLDER*	*BALANCE*
	bank A	101	Smith	$ 1,000
	bank A	102	Jones	$ 2,500
	bank A	103	Smith	$ 100
	bank B	101	Anderson	$ 1,500
	bank B	105	Brown	$ 1,000
	bank C	201	Smith	$ 50

As expressed by Σ, each combination of bank and account number uniquely determines the account holder and his balance, so $\{BANK, ACC\_NO\}$ is the (unique) key of ACC_S. ◊

Whenever appropriate, we will refer to this example in the following, adjusting it to the respective situation if necessary. When going into technical details, however, we prefer abstract examples for reasons of clarity. To avoid confusion, we hereafter adapt examples that refer to Example 5.1 to the assumptions of Section 2.3, that is, we ensure that constant identifiers begin with a lowercase letter or with a number. For example, bank A will be denoted by $bankA$, Smith by $smith$ etc.

5.2 Confidentiality

After having clarified the desired properties and the essential parameters of our framework, we now introduce the *modified query evaluation*, which serves as a basis for our stateless CQE approach, and explain our notion of confidentiality preservation.

As already outlined in Chapter 3, stateful CQE deviates from the ordinary evaluation to preserve the confidentiality of the declared secrets. The stateless CQE we are about to develop also needs a special form of query evaluation. In order to be flexible with regard to enhancements of the stateless CQE, we first introduce the notion of a general modified query evaluation.

Definition 5.2 (Modified query evaluation). *A modified query evaluation m_eval maps*

- *a (possibly infinite) query sequence $Q = \langle \Phi_1, \Phi_2, \ldots \rangle$ (with $\Phi_1, \Phi_2, \ldots$ being queries expressed in a suitable query language, that is, in a fragment of first-order logic),*
- *a database instance r, and*
- *a confidentiality policy $pot\_sec = \{\Psi_1, \Psi_2, \ldots, \Psi_m\}$ (with $\Psi_1, \Psi_2, \ldots, \Psi_m$ being potential secrets expressed in a suitable policy language, that is, in a fragment of first-order logic)*

on an answer sequence

$$m\_eval(Q)(r, pot\_sec) = \langle ans_1, ans_2, \ldots \rangle.$$

Since we will talk about secure query evaluation in the following, we also need a precise notion of security. In this thesis, we concentrate on the security goal of confidentiality and thus a secure query evaluation is characterized by the preservation of confidentiality of the declared potential secrets. Consequently, we will use the terms *secure* and *confidentiality-preserving* synonymously hereafter. Informally, in our framework, the evaluation of a query preserves confidentiality if for each declared potential secret the database user considers it possible that the negated secret is true in the database instance after having received the answer to his query. More precisely, considering a fixed potential secret from the policy, we demand that there exists an alternative database instance (not necessarily different from the actual instance) that is (from the perspective of the database user) indistinguishable from the actual one and that makes the potential secret false.

In order to preserve confidentiality, we also have to ensure that the database user does not have any harmful a priori knowledge. In the following two cases, we consider the a priori knowledge of the user harmful:

- The database user knows that a potential secret is true in the database instance.
- The database user has knowledge that is false in the database instance.

In the first case, the user has already disclosed a secret whereas in the second case, the user knowledge might get inconsistent when adding knowledge that is true in the database instance. Inconsistent knowledge, however, enables the user to infer arbitrary formulas, in particular the potential secrets.

Definition 5.3 gives a precondition for a pair of database instance and confidentiality policy (r, pot_sec), ensuring that the a priori knowledge of the database user is not harmful. Definition 5.4 states the actual conditions for a modified query evaluation to be secure.

Definition 5.3 (Admissibility (for closed policies)). *Given the a priori user knowledge log_0, a pair (r, pot_sec) of a database instance r and a confidentiality policy pot_sec is called* admissible with respect to log_0 *if the following conditions hold:*

1. $r \models_M log_0$;
2. a) *if pot_sec is known to the user: $log_0 \not\models \Psi$ for all $\Psi \in pot\_sec$,*
   b) *if pot_sec is unknown to the user:*
   $log_0 \not\models \Psi$ *for all* $\Psi \in pot\_sec$ *with* $r \models_M \Psi$.

Remark. Remember that we assume $log_0 = \Sigma$ for our investigations according to Section 5.1. Therefore, the first requirement of the admissibility definition, $r \models_M log_0$, is always satisfied.

Definition 5.4 (Secure query evaluation). *Given*

- *a (possibly infinite) query sequence $Q = \langle \Phi_1, \Phi_2, ... \rangle$ (with $\Phi_1, \Phi_2, ...$ being queries expressed in a suitable query language) and*
- *a confidentiality policy $pot\_sec = \{\Psi_1, \Psi_2, \dots, \Psi_m\}$ (with $\Psi_1, \Psi_2, \dots, \Psi_m$ being potential secrets expressed in a suitable policy language),*

the modified query evaluation m_eval is secure with respect to pot_sec *if*

> *for all finite prefixes Q' of Q*
> *for all $\Psi \in pot\_sec$*
> *for all database instances r*
>   *such that (r, pot_sec) is admissible*
>   *with respect to the a priori knowledge log_0*
> *there exists an alternative instance r' and*
> *there exists an alternative policy pot_sec'*
>   *such that (r', pot_sec') is admissible*
>   *with respect to the a priori knowledge log_0*
> *satisfying the following properties:*

1. *$\mathsf{m\_eval}(Q')(r,\mathsf{pot\_sec}) = \mathsf{m\_eval}(Q')(r',\mathsf{pot\_sec}')$;*
2. *$\mathsf{eval}^*(\Psi)(r') = \neg\Psi$;*
3. *if $\mathsf{pot\_sec}$ is known to the database user, then $\mathsf{pot\_sec}' = \mathsf{pot\_sec}$.*

The modified query evaluation $\mathsf{m\_eval}$ is secure *if it is secure with respect to every possible policy $\mathsf{pot\_sec}$.*

Remark. Although we fixed $log_0 = \Sigma$ in Section 5.1, Definition 5.4 can also be applied for more general a priori knowledge sets log_0 that contain elements from the underlying logic.

Observe that parts of Definitions 5.3 and 5.4 refer to the awareness of the database user regarding the confidentiality policy, although in the parameter setting of our investigations we assume the policy to be known to the user. In Section 7.3, however, we will see that this assumption cannot be kept up in every parameter setting. Consequently, to be prepared for this situation, we formulated Definitions 5.3 and 5.4 a bit more general. If the policy is unknown, there is one more degree of freedom with respect to confidentiality proofs because the policy is not fixed but can (not necessarily uniquely) be constructed from the answers to the queries; if, however, the policy is known, this degree of freedom disappears.

The most important property of a secure modified query evaluation is that the user is never able to infer a potential secret with his knowledge. More precisely, he is never able to learn that a declared potential secret is actually true in the database instance under consideration. This property follows intuitively from Definition 5.4 which, considering a fixed potential secret, demands for the existence of an instance that leads to the same answers that the user got from the system (and thus leads to the same user knowledge) but makes the potential secret false. Thus, for each declared potential secret, the user always considers a database instance possible in which this secret is false and consequently never knows that a potential secret is true in the actual database instance. This property is shown for the modification strategy of refusal by the following Lemma.

Lemma 5.5 (The user log is harmless). *Let $\mathsf{m\_eval}$ be a secure modified query evaluation that uses the modification strategy of refusal, $Q = \langle \Phi_1, \Phi_2, ...\rangle$ a (possibly infinite) query sequence, $\mathsf{pot\_sec}$ a confidentiality policy, $log_0 = \Sigma$ the a priori user knowledge, and r a database instance. Furthermore, let log_j denote the user knowledge after having received the answer to the j-th query from $\mathsf{m\_eval}(Q)(r,\mathsf{pot\_sec})$. Then for all potential secrets $\Psi \in \mathsf{pot\_sec}$ it holds that $log_j \not\models \Psi$.*

Proof. Since $\mathsf{m\_eval}$ is supposed to use the modification strategy of refusal, each returned answer is either a refusal (`mum`) or it is true in the considered database instance r. Moreover, only non-refused answers are added to the user log (see the definition (3.4) of the user log in Chapter 3), so $log_j = log_0 \cup \{ans_i \mid i \in \{1, \dots, j\}, ans_i \neq \texttt{mum}\}$. Consequently, it holds that

$$r \models_{\mathsf{M}} log_j. \tag{5.1}$$

Now consider a potential secret $\Psi \in pot\_sec$. We prove the claim by induction on the sequence number of the user log (denoted by j).

$j = 0$: According to the remark to Definition 5.3, (r, pot_sec) is admissible with respect to log_0. Thus, we know that $log_0 \not\models \Psi$.

$j + 1$: By the induction hypothesis it holds that

$$log_j \not\models \Psi. \tag{5.2}$$

If $ans_{j+1} = \texttt{mum}$, then $log_{j+1} = log_j$ and consequently, $log_{j+1} \not\models \Psi$. We therefore assume that $ans_{j+1} \neq \texttt{mum}$ in the following. By definition of the user knowledge,

$$log_{j+1} = log_j \cup \{ans_{j+1}\}. \tag{5.3}$$

We now complete the proof by contradiction and therefore indirectly assume that $log_{j+1} \models \Psi$, that is, with (5.3),

$$log_j \cup \{ans_{j+1}\} \models \Psi. \tag{5.4}$$

By definition of logical implication, (5.4) means that for all instances r the following holds:

$$\text{If } r \models_{\mathsf{M}} log_j \cup \{ans_{j+1}\} \text{ then } r \models_{\mathsf{M}} \Psi. \tag{5.5}$$

Since m_eval is supposed to be secure, by Definition 5.4 there exists an alternative instance r' and an alternative policy pot_sec' such that the a priori knowledge log_0 is true in r' and m_eval yields the same answers on (r', pot_sec') and (r, pot_sec). Thus, with (5.1) it follows that

$$r' \models_{\mathsf{M}} log_j \cup \{ans_{j+1}\}. \tag{5.6}$$

Moreover, again by Definition 5.4, Ψ is false in r', that is,

$$r' \not\models_{\mathsf{M}} \Psi. \tag{5.7}$$

Consequently, r' is a witness instance against (5.5) and therefore contradicts (5.4) which completes the proof. □

5.3 Outlook

Having defined the framework for our approach as well as the main requirements, the parameters for our investigations, and our notion of confidentiality preservation, we will develop an efficient inference control approach for relational databases in Chapters 6 and 7. From a “logical perspective”, these chapters are structured in a bottom-up fashion:

- In Chapter 6, we start our investigations with syntactically simple query and policy languages, that is, we consider sets of atomic sentences with existential quantifiers for policy elements and without existential quantifiers for queries. It will turn out that these restrictions allow for an efficient mechanism that reaches the declarative goals of inference control, as given by Definition 5.4.
- In Chapter 7, we stepwisely enhance these simple languages to more expressive ones. More precisely, we consider the following enhancements:
  - In Section 7.1, we enhance the query language by existential quantifiers. As a consequence, we will have to further restrict the policy language in order to keep up preservation of confidentiality.
  - In Section 7.2, we additionally consider Boolean operations for both the query and the policy language. We will basically learn that the user must be prevented from using disjunction in queries; moreover, the security administrator must neither use negation nor conjunction when declaring policy elements.
  - In Section 7.3, we investigate free variables for the query language, that is, we enable the user to express open queries. This enhancement will lead to the additional assumption that the user must not be aware of the policy.
  - In Section 7.4, we investigate free variables for the policy language. This enables the security administrator to express a set of “uniform” secrets with a single policy element. This enhancement will require us to slightly adapt some definitions, but it will not be necessary to assume further restrictions.
  - Finally, in Section 7.5, we combine the enhancements from Sections 7.1–7.4. It will turn out that negation in combination with free variables might lead to unsafe queries, that is, queries whose evaluation is infinite.

In Section 7.6, we conclude Part II of this thesis with an overview of the investigated languages. Moreover, we justify that our investigations are, in some sense, “complete”, that is, they basically cover the space of possibilities for stateless CQE in relational databases.

Chapter 6

A Basic Case: Select-Queries

In this chapter, we start our investigations with a rather basic scenario, that is, with simple query and policy languages. On the one hand, this approach enables us to develop a simple but effective inference control mechanism; on the other hand, unfortunately, "basic" also means "restrictive", especially from the point of view of the security administrator and the database user. To compensate for this drawback, we will refine our approach in Chapter 7.

In Section 6.1, we introduce the query and policy languages being investigated in this chapter. For the policy language, we describe and prove a couple of important properties in Section 6.2 which are also referred to in subsequent chapters. In Section 6.3, we develop a stateless CQE mechanism for our basic case and formally show that it preserves confidentiality in the sense of Definition 5.4. Moreover, we propose an efficient algorithmic implementation for this mechanism in Section 6.4. Finally, we shortly summarize our contributions in Section 6.5.

This chapter is based on the work of Biskup and Lochner [BL07], who originally investigated stateless CQE for select-queries.

6.1 Simple Query and Policy Languages

For our basic approach, we assume that the database user may only ask for *full tuples*. More precisely, given a database instance r and a tuple μ, the user may ask whether r contains μ or not. Here, μ must be a tuple of n constants where n denotes the number of attributes of the relation as declared in the relation schema. Other forms of queries are not allowed for now. We call these queries *select-queries*[5] in the following because in terms of the relational algebra, evaluating such a query is equivalent to performing a selection operation on a relation. According to the assumption in Section 5.1, however, our approach is logic-based rather than algebra-based and consequently we express queries in relational calculus.[6] As an example for select-queries, consider the

[5]More precisely, we even only consider *closed* select-queries, that is, queries that can be answered by `true` or `false`. For conciseness, however, we shortly refer to select-queries in the following.

[6]It is well known (see, for example, [AHV95]) that relational algebra and relational calculus are equivalent in terms of expressiveness.

relation schema $\langle R, ABC, \Sigma\rangle$: The user may express queries of the form $\Phi \equiv R(a,b,c)$ where a, b, c denote constant symbols.

When interpreting a select-query in a more general way as a logical formula, it is a sentence, since it does not contain free variables. We will therefore speak of it as an *R-sentence*. The language of R-sentences used as query language in this chapter is equivalent to the set of n-ary tuples where n is the number of attributes of the relation R.

Definition 6.1 (R-sentences). *The language $\mathscr{L}_R$ is defined by*

$$\mathscr{L}_R := \{R(v_1, \dots ,v_n) \mid v_i \in Const\}.$$

Remark. The language $\mathscr{L}_R$ is also equivalent to the set of ground atoms of the underlying first-order logic.

The security administrator also needs a language that defines the syntax of potential secrets. In this chapter, we assume that a potential secret is either an R-sentence from the language $\mathscr{L}_R$ or an *existential R-sentence* which is basically an R-sentence that additionally contains existentially quantified variables. As an additional restriction we demand that each of these variables may not occur more than once in an element of the language. Interpreted as queries, existential R-sentences may also be referred to as *select-project-queries* because in terms of the relational algebra an existential R-sentence can be expressed by selection and projection operations. Considering the schema $\langle R, ABC, \Sigma\rangle$ again, an example for an existential R-sentence is $(\exists X_A)R(X_A,b,c)$; it corresponds to the selections $B = b$ and $C = c$, and the subsequent projection to attributes B and C.

The following definition formally describes the language of existential R-sentences which is equivalent to the set of partial tuples over the relation R. It is used as policy language in this chapter.

Definition 6.2 (Existential R-sentences). *The language $\mathscr{L}_{\exists R}$ is defined by*

$$\begin{aligned}\mathscr{L}_{\exists R} := \{\ & (\exists X_1) \dots (\exists X_m)R(v_1, \dots ,v_n) \mid 0 \le m \le n,\\ & X_i \in Var,\\ & v_i \in Var \cup Const,\\ & \{X_1, \dots , X_m\} \subseteq \{v_1, \dots , v_n\},\\ & v_i \in Var \Rightarrow v_i = X_j \textit{ for a } j \in \{1, \dots , m\},\\ & v_i, v_j \in Var \Rightarrow v_i \neq v_j\ \}.\end{aligned}$$

Remark. Considering a set of existential R-sentences, for example, a set of potential secrets, we assume the elements of the set to be “variable-disjoint”, unless otherwise stated. That is, each variable occurs at most once in the whole set.

Note that $\mathscr{L}_R$ is a subset of $\mathscr{L}_{\exists R}$ and consequently elements from $\mathscr{L}_R$ are existential R-sentences, too. An existential R-sentence refers to exactly one relation which we call the *underlying relation*. Analogously, the associated relation schema is the *underlying schema*. We may interpret an existential R-sentence as a generalized tuple, that is, a tuple possibly containing null values of the kind "value existing but unknown". We therefore adapt the square bracket notation from Section 2.1 for existential R-sentences: The value of an attribute A (of the underlying relation R) in an existential R-sentence χ will be denoted by $\chi[A]$ from now on. Moreover, we use the notation $\mathit{select}(\chi)$ for the set of attributes being instantiated with a constant from Const in χ. When we replace the occurrence of zero or more constants in an existential R-sentence χ with (pairwise different) existentially quantified variables, the resulting existential R-sentence χ^w is called a *weakening* of χ. We illustrate these notions with the following example.

Example 6.3. Consider the relation schema $\langle R, ABC, \emptyset\rangle$ and two existential R-sentences:

$$\chi_1 \equiv (\exists X_A) R(X_A,b,c)$$
$$\chi_2 \equiv (\exists X_A)(\exists X_B) R(X_A,X_B,c)$$

The underlying relation of both χ_1 and χ_2 is R; consequently, $\langle R, ABC, \emptyset\rangle$ is the underlying schema of χ_1 and χ_2. The existential R-sentences may be interpreted as the tuples $R(-,b,c)$ and $R(-,-,c)$, respectively, where "$-$" denotes an existing but unknown value. Using the square bracket notation, we can refer to the attribute values of χ_1 and χ_2:

$$\chi_1[A] = X_A, \quad \chi_1[B] = b, \quad \chi_1[C] = c,$$
$$\chi_2[A] = X_A, \quad \chi_2[B] = X_B, \quad \chi_2[C] = c.$$

The attributes being instantiated with constants of χ_1 and χ_2 are $\mathit{select}(\chi_1) = \{B,C\}$ and $\mathit{select}(\chi_2) = \{C\}$, respectively. Finally, χ_2 is a weakening of χ_1 because χ_2 can be constructed by replacing b in χ_1 with the existentially quantified variable X_B. $\diamondsuit$

The query language $\mathscr{L}_R$ and the policy language $\mathscr{L}_{\exists R}$ are clearly rather restrictive, but there is one crucial benefit: unlike with more expressive query and policy languages, we need not to impose any restrictions on the relation schema, but confidentiality can be preserved regardless of the declared schema constraints. We will formulate this insight in the context of Theorem 6.12 in Section 6.3.

6.2 Properties of Existential R-Sentences

An important property of the language $\mathscr{L}_{\exists R}$ of existential R-sentences is that no "sophisticated implications" are possible within this language. That is to say that each formula from $\mathscr{L}_{\exists R}$ being implied by a set of formulas from $\mathscr{L}_{\exists R}$ is already implied by a single element of this set of formulas. We state this property more precisely in Lemma 6.8, but prior to this we define and illustrate our notion of relevance between two existential R-sentences which will be used throughout this thesis, and we present Lemma 6.7, which captures an important statement about existential R-sentences that are true in a database instance.

Definition 6.4 (Relevance between existential R-sentences). *Consider two existential R-sentences χ and χ' with the same underlying relation R. We say that χ is* relevant *for χ' if for every $A \in \mathit{select}(\chi)$ it holds that $\chi'[A] = \chi[A]$.*

For existential R-sentences, the notion of relevance is closely related to the logical implication, as stated by the following lemma.

Lemma 6.5 (Relevance and logical implication). *An existential R-sentence χ is relevant for another existential R-sentence χ' if and only if $\chi' \models \chi$.*

Proof. First, we show the "$\Rightarrow$"-part. Let χ be relevant for χ', which means that

$$\chi'[A] = \chi[A] \text{ for every } A \in \mathit{select}(\chi) \tag{6.1}$$

according to Definition 6.4. By the definition of $\mathit{select}(\chi)$ and the structure of existential R-sentences, we know that

$$\chi[B] \in \mathit{Var} \text{ for every } B \notin \mathit{select}(\chi) \tag{6.2}$$

and, moreover, that each such variable is existentially quantified in χ. From (6.1), (6.2), the structure of existential R-sentences and the definition of logical implication, we then conclude that $\chi' \models \chi$.

Second, we show the "$\Leftarrow$"-part and therefore assume that $\chi' \models \chi$. Thus, again by the structure of existential R-sentences and the definition of logical implication, we know that each constant occurring in χ must occur in χ' at the same position. More formally: $\chi'[A] = \chi[A]$ for every $A \in \mathit{select}(\chi)$, which is, according to Definition 6.4, equivalent with χ being relevant for χ'. $\square$

The notion of relevance according to Definition 6.4 is now illustrated by the following example.

Example 6.6. Let χ, χ' and χ'' be three existential R-sentences with the same underlying schema $\langle R, ABC, \emptyset\rangle$:

$$\begin{aligned} \chi &\equiv R(a,b,c) \\ \chi' &\equiv (\exists X_A)R(X_A,b,c) \\ \chi'' &\equiv (\exists X_C)R(a,b,X_C) \end{aligned}$$

We first determine for each of these existential R-sentences the attributes that are instantiated with constants:

$$\begin{array}{lcl} \mathit{select}(\chi) & = & \{A, B, C\} \\ \mathit{select}(\chi') & = & \{B, C\} \\ \mathit{select}(\chi'') & = & \{A, B\} \end{array}$$

Now we can see that χ' is relevant for χ because $\chi'[B] = \chi[B]$ and $\chi'[C] = \chi[C]$. Likewise, χ'' is relevant for χ because $\chi''[A] = \chi[A]$ and $\chi''[B] = \chi[B]$. However, χ'' is not relevant for χ' because $\chi''[A] \neq \chi'[A]$. $\Diamond$

An important property of existential R-sentences is captured by Lemma 6.7: If an existential R-sentence is true in a database instance r, then there exists an R-sentence that is true in r as well and implies the existential R-sentence. We will need this result in the following whenever reasoning about existential R-sentences being true in the considered database instance.

Lemma 6.7 (Existential R-sentences and ground atoms). *Consider an instance r and an existential R-sentence $\chi \in \mathscr{L}_{\exists R}$. It holds that $r \models_{\mathsf{M}} \chi$ if and only if there exists a ground atom $\chi_g \in \mathscr{L}_R$ with $r \models_{\mathsf{M}} \chi_g$ and $\chi_g \models \chi$.*

Proof. If χ is a ground atom, the lemma obviously holds by setting $\chi_g :\equiv \chi$. We thus assume that χ is not a ground atom, that is, $\chi \notin \mathscr{L}_R$, and prove both directions of the lemma separately.

First, we show the "$\Rightarrow$"-part: Consider $\chi \equiv (\exists X_1) \dots (\exists X_l) R(v_1, \dots, v_n)$ and let

$$r \models_{\mathsf{M}} \chi. \tag{6.3}$$

By the semantics of existential quantifiers,

$$\begin{array}{l} r \models_{\mathsf{M}} \chi \text{ if and only if} \\ \text{there exists a variable assignment } \alpha \text{ such that } r \models_{\mathsf{M}} \alpha(R(v_1, \dots, v_n)). \end{array} \tag{6.4}$$

Observe that $\alpha(R(v_1, \dots, v_n)) \in \mathscr{L}_R$ is a ground atom and set $\chi_g :\equiv \alpha(R(v_1, \dots, v_n))$. From (6.3) and (6.4) it then follows that $r \models_{\mathsf{M}} \chi_g$ for the ground atom χ_g. It remains to show that $\chi_g \models \chi$ but this immediately follows from Lemma 6.5.

Second, we show the "$\Leftarrow$"-part: Consider a ground atom χ_g with

$$r \models_{\mathsf{M}} \chi_g \tag{6.5}$$

and

$$\chi_g \models \chi. \tag{6.6}$$

By the definition of logical implication, (6.6) can also be written as

$$\text{for all instances } r: \text{ if } r \models_{\mathsf{M}} \chi_g \text{ then } r \models_{\mathsf{M}} \chi. \tag{6.7}$$

Together with (6.5), we conclude that $r \models_{\mathsf{M}} \chi$. □

As already indicated at the beginning of this section, we now show that sets of existential R-sentences only allow for "trivial" inferences in the sense that an existential R-sentence that is implied by a set of existential R-sentences is already implied by a single element of this set.

Lemma 6.8 (Logical implication for existential R-sentences). *Let the formulas χ and $\chi_1, \dots, \chi_m$ be existential R-sentences from $\mathscr{L}_{\exists R}$. Then the following properties are equivalent:*

1. $\{\chi_1, \dots, \chi_m\} \models \chi$.
2. *There exists a $\chi_i \in \{\chi_1, \dots, \chi_m\}$ such that χ is relevant for χ_i.*

Proof. First, we show "1. ⇒ 2." by contraposition, that is, we assume that

$$\chi \text{ is not relevant for any } \chi_i \in \{\chi_1, \dots, \chi_m\} \tag{6.8}$$

and show that $\{\chi_1, \dots, \chi_m\} \not\models \chi$ follows. From (6.8) we conclude with Lemma 6.5 that $\chi_i \not\models \chi$ for every $\chi_i \in \{\chi_1, \dots, \chi_m\}$, and consequently χ and χ_i "disagree" on at least one constant attribute value. It follows that

$$\alpha(\chi_i^*) \not\models \chi \tag{6.9}$$

for an arbitrary variable assignment α, where χ_i^* denotes the matrix, that is, the quantifier-free part of the existential R-sentence χ_i.

We now construct a witness instance r against property 1 by completing the elements of $\{\chi_1, \dots, \chi_m\}$ to ground atoms. For this purpose, we choose a variable assignment α and identify r with the set $\{\alpha(\chi_1^*), \dots, \alpha(\chi_m^*)\}$. Consequently,

$$r \models_{\mathsf{M}} \alpha(\chi_i^*) \quad \text{for every } i \in \{1, \dots, m\} \tag{6.10}$$

and $r \not\models_{\mathsf{M}} \chi_g$ for every ground atom $\chi_g \notin \{\alpha(\chi_1^*), \dots, \alpha(\chi_m^*)\}$. Since, by the semantics of the existential quantifier, $\alpha(\chi_i^*) \models \chi_i$ for every $i \in \{1, \dots, m\}$, we get

$$r \models_{\mathsf{M}} \{\chi_1, \dots, \chi_m\} \tag{6.11}$$

from (6.10). Moreover, by construction of r and (6.9), we know from Lemma 6.7 that

$$r \not\models_{\mathsf{M}} \chi. \tag{6.12}$$

Finally, from (6.11), (6.12) and the definition of logical implication, we conclude that $\{\chi_1, \dots, \chi_m\} \not\models \chi$.

Second, we show "2. $\Rightarrow$ 1." and therefore assume that χ is relevant for a $\chi_i \in \{\chi_1, \dots, \chi_m\}$. From Lemma 6.5 it follows that $\chi_i \models \chi$. Then, by monotonicity of first-order logic, $\{\chi_1, \dots, \chi_m\} \models \chi$ holds as well which completes the proof. □

Since a positive (ordinary) evaluation of a query $\Phi \in \mathscr{L}_R \subset \mathscr{L}_{\exists R}$ in the sense of Definition 2.2 is an existential R-sentence, the lemma can basically be interpreted as follows:

> In the absence of further knowledge, the database user will be able to infer an existential R-sentence from the set of positive query answers *if and only if* already a single positive answer would enable him to do so.

The database user, however, may exploit further knowledge when trying to disclose a secret: If a query is false in the database instance, the ordinary query evaluation according to Definition 2.2 will return the *negated query* as an answer; moreover, in Section 5.1 we assumed the database user to know the declared *semantic constraints* of the database (which are expressed in form of functional dependencies). The following lemma is an adaption of Lemma 6.8 considering every form of information the database user might learn when receiving answers from the database.[7]

Lemma 6.9 (Implications of R-sentences and FDs). *Let $\mathcal{F} = \mathcal{F}_1 \uplus \mathcal{F}_2 \uplus \mathcal{F}_3$ be a finite and consistent set of formulas such that $\mathcal{F}_1 \subset \mathscr{L}_R$, $\mathcal{F}_2 \subset \{\neg\chi \mid \chi \in \mathscr{L}_R\}$, and $\mathcal{F}_3$ is a set of FDs. Then for an existential R-sentence $\Psi \in \mathscr{L}_{\exists R}$, the following properties are equivalent:*

1. *$\mathcal{F} \models \Psi$.*
2. *There exists a $\chi \in \mathscr{L}_R$ with $\chi \in \mathcal{F}$ and $\chi \models \Psi$.*

Proof. First, we show "1. $\Rightarrow$ 2." by contraposition and therefore assume that for every $\chi \in \mathscr{L}_R$ it holds that

$$\chi \notin \mathcal{F} \quad \text{or} \quad \chi \not\models \Psi. \tag{6.13}$$

We now construct a witness instance r against property 1, that is, $r \models_{\mathsf{M}} \mathcal{F}$ and $r \not\models_{\mathsf{M}} \Psi$, by identifying r with $\mathcal{F}_1$; in other words:

$$\begin{aligned} r &\models_{\mathsf{M}} \chi \quad \text{if } \chi \in \mathcal{F}_1, \\ r &\not\models_{\mathsf{M}} \chi \quad \text{otherwise.} \end{aligned} \tag{6.14}$$

[7]Note that, in contrast to Lemma 6.8, we do not consider arbitrary existential R-sentences in Lemma 6.9 but only those that represent full tuples, that is, R-sentences.

Since $\mathcal{F}$ is finite, $\mathcal{F}_1$ must be finite, too, and thus r may be interpreted as a finite relation. We now show that

$$r \models_{\mathsf{M}} \mathcal{F} \tag{6.15}$$

by exemplarily considering a $\chi \in \mathcal{F}$ via case distinction:

1. $\chi \in \mathcal{F}_1$. By (6.14) it holds that $r \models_{\mathsf{M}} \chi$.

2. $\chi \in \mathcal{F}_2$. By the definition of $\mathcal{F}_2$, χ is of the form $\neg\chi'$ with $\chi' \in \mathscr{L}_R$. Since $\mathcal{F}$ is supposed to be consistent, there is no formula in $\mathcal{F}_1$ whose negative complement is in $\mathcal{F}_2$. Thus, by the construction of r, $r \not\models_{\mathsf{M}} \chi'$. By the definition of the $\models_{\mathsf{M}}$-operator, it then follows that $r \models_{\mathsf{M}} \neg\chi'$, leading to $r \models_{\mathsf{M}} \chi$.

3. $\chi \in \mathcal{F}_3$. We perform an indirect proof and assume that the FD $\chi \equiv \mathcal{A} \rightarrow \mathcal{B}$ (with $\mathcal{A}, \mathcal{B}$ denoting sets of attributes) is violated in r ($r \not\models_{\mathsf{M}} \chi$). For an FD to be violated, there must exist two tuples μ_1, μ_2 in r that correspond on the (constant values of the) $\mathcal{A}$-attributes but differ on (at least one value of) the $\mathcal{B}$-attributes. Then the set $\{\mu_1, \mu_2, \chi\}$ is inconsistent and, since $\{\mu_1, \mu_2, \chi\} \subset \mathcal{F}$, $\mathcal{F}$ must be inconsistent, too. This, however, contradicts the precondition that $\mathcal{F}$ is a consistent set.

This completes the proof of (6.15). It remains to show that $r \not\models_{\mathsf{M}} \Psi$. We can derive from (6.13) that $\chi \not\models \Psi$ for every $\chi \in \mathcal{F}_1$, so (by Lemma 6.5) there does not exist a $\chi \in \mathcal{F}_1$ such that Ψ is relevant for χ. By applying Lemma 6.8 we can conclude that $\mathcal{F}_1 \not\models \Psi$ and therefore, by (6.14), $r \not\models_{\mathsf{M}} \Psi$.

Second, we show "2. $\Rightarrow$ 1.". By $\chi \in \mathcal{F}$, $\chi \models \Psi$ and the monotonicity of first-order logic, it obviously follows that $\mathcal{F} \models \Psi$. $\square$

Again, we provide an intuitive interpretation of Lemma 6.9:

> The database user will be able to infer an existential R-sentence from the knowledge he gets during the answering of queries (which are supposed to be full tuples, that is, R-sentences) *if and only if* already a single positive answer would enable him to do so.

Remember that, in this chapter, a confidentiality policy is a set of existential R-sentences. Consequently, the above interpretation of Lemma 6.9 already gives a hint on how to develop a confidentiality-preserving query answering mechanism by considering the answers to the user's queries isolatedly, which enables us to avoid the user log. We elaborate on this question in Section 6.3.

Finally, observe that, due to the structure of existential R-sentences, a negated existential R-sentence cannot be exploited to infer another (positive) existential R-sentence. We will need this property to rule out inference possibilities in the context of the confidentiality proof in Section 6.3.

Lemma 6.10 (Implications of negated existential R-sentences). *Let χ_1 and χ_2 be existential R-sentences. Then, independent of the actual definition of χ_1 and χ_2, it holds that $\neg\chi_1 \not\models \chi_2$.*

Proof. We assume the contrary, $\neg\chi_1 \models \chi_2$, for some existential R-sentences χ_1 and χ_2. By the definition of logical implication this can also be written as

$$\text{for all instances } r: \text{ if } r \models_{\mathsf{M}} \neg\chi_1 \text{ then } r \models_{\mathsf{M}} \chi_2. \tag{6.16}$$

By the semantics of first-order logic, (6.16) is equivalent to

$$\text{for all instances } r: \text{ if } r \not\models_{\mathsf{M}} \chi_1 \text{ then } r \models_{\mathsf{M}} \chi_2. \tag{6.17}$$

Now consider the empty instance $r = \emptyset$. Since no ground atoms are true in r, it follows by Lemma 6.7 that $r \not\models_{\mathsf{M}} \chi_1$ and $r \not\models_{\mathsf{M}} \chi_2$. This is a contradiction to (6.17).□

6.3 Confidentiality-Preserving Stateless CQE

From the intuitive interpretation of Lemma 6.9 in Section 6.2, we can derive a mechanism for answering a sequence of user queries that preserves confidentiality regarding a declared confidentiality policy. Whenever a single query of the sequence is harmless, that is, when it (considered isolatedly) does not disclose one of the declared secrets, it is answered correctly; otherwise, the answer to the query is refused. Observe that this mechanism is stateless in the sense that no log file for the assumed user knowledge is needed any longer. Moreover, instead of (costly) theorem prover calls, (efficient) access control is sufficient to enforce the mechanism (refer to Section 6.4 for details).

We describe a general form of our mechanism (for queries from $\mathscr{L}_{\exists R}$) in the following Definition 6.11 and afterwards prove it confidentiality-preserving when considering only queries from $\mathscr{L}_R$ in Theorem 6.12. In Chapter 7, we will reuse the mechanism from Definition 6.11 and prove it confidentiality-preserving also for queries from $\mathscr{L}_{\exists R}$ under certain assumptions.

Definition 6.11 (Stateless CQE for existential R-sentences). *The stateless CQE $scqe$ is a specific modified query evaluation according to Definition 5.2. Given a policy $pot\_sec \subset \mathscr{L}_{\exists R}$, the answers $ans_1, ans_2, \ldots$ to a query sequence $Q = \langle \Phi_1, \Phi_2, \ldots \rangle$, with $\Phi_1, \Phi_2, \ldots \in \mathscr{L}_{\exists R}$, are determined subject to the censor function*

$$censor_{\exists R}(pot\_sec, \Phi) := (\textit{exists } \Psi)[\Psi \in pot\_sec \textit{ and } \Phi \models \Psi].$$

The stateless CQE scqe *is then defined as follows:*

$$\mathsf{scqe}(Q)(r,\mathsf{pot\_sec}) := \langle \mathsf{ans}_1, \mathsf{ans}_2, \ldots \rangle \qquad \textit{with}$$

```
ans_i  :=  if censor_∃R(pot_sec,Φ_i)
           then mum
           else eval*(Φ_i)(r)
```

Remarks. In order to avoid meta-inferences (see Chapter 3), the censor of scqe should actually not only check if the positive formula Φ allows for harmful inferences but also if the negative formula $\neg\Phi$ does. From Lemma 6.10, however, we know that in the setting of Definition 6.11 such inferences cannot occur when considering negative formulas. Even in combination with the user log, the formula $\neg\Phi$ does not enable the user to draw harmful inferences. This will be proven formally in Theorem 6.12 for R-sentences and in Theorem 7.10 for existential R-sentences.

Moreover, we need not to propose an "improved" version of scqe (like in Section 3.2 for stateful CQE), since the possible inconsistency during the censor evaluation cannot occur for scqe.

The following theorem shows that scqe, as defined above but with $\mathscr{L}_R$ as query language, is an effective inference control mechanism, that is, it preserves confidentiality with respect to the declared policy.

Theorem 6.12 (scqe preserves confidentiality under $\mathscr{L}_\mathsf{R}$). *Consider an instance* r*, a confidentiality policy* $\mathsf{pot\_sec} \subset \mathscr{L}_{\exists R}$ *and a query sequence* $Q = \langle \Phi_1, \Phi_2, \ldots \rangle$ *with* $\Phi_1, \Phi_2, \ldots \in \mathscr{L}_R$*. Moreover, let* $\mathsf{log}_0 = \Sigma$ *be the a priori user knowledge and assume that* $(r,\mathsf{pot\_sec})$ *is admissible with respect to* log_0*. Then the stateless CQE* $\mathsf{scqe}(Q)(r,\mathsf{pot\_sec})$ *is secure in the sense of Definition 5.4.*

Proof. We consider a finite prefix $Q' = \langle \Phi_1, \Phi_2, \ldots, \Phi_n \rangle$ of the query sequence Q and a potential secret $\Psi \in \mathsf{pot\_sec}$. Since $\mathsf{pot\_sec}$ is assumed to be known to the user, we have to find an alternative instance r' with $(r',\mathsf{pot\_sec})$ being admissible with respect to log_0 and r' satisfying the following properties (according to Definition 5.4):

1. $\mathsf{scqe}(Q')(r,\mathsf{pot\_sec}) = \mathsf{scqe}(Q')(r',\mathsf{pot\_sec})$;
2. $\mathsf{eval}^*(\Psi)(r') = \neg\Psi$;

We start by constructing an auxiliary set log capturing the a priori knowledge of the user and the non-refused answers during the answer of the query sequence Q':

$$\mathsf{log} := \mathsf{log}_0 \cup \{\mathsf{ans}_i \mid i \in \{1, \ldots, n\}, \mathsf{ans}_i \neq \mathtt{mum}\} \tag{6.18}$$

We show by an indirect proof that log does not imply the potential secret Ψ, and therefore start with assuming the contrary:

$$\mathit{log} \models \Psi. \tag{6.19}$$

Note that log meets the requirements of the set $\mathcal{F}$ in Lemma 6.9 which, together with (6.19), then leads to

$$\mathit{ans}_i \models \Psi \tag{6.20}$$

for a positive $\mathit{ans}_i \in \mathit{log}$. By (6.18), ans_i is the non-refused answer to query Φ_i and thus we get

$$\mathit{eval}^*(\Phi_i)(r) \models \Psi. \tag{6.21}$$

Since $\mathit{eval}^*(\Phi_i)(r) \in \{\Phi_i, \neg\Phi_i\}$ and $\neg\Phi_i \not\models \Psi$ by Lemma 6.10, (6.21) can be reduced to

$$\Phi_i \models \Psi. \tag{6.22}$$

According to Definition 6.11, this leads to $\mathit{ans}_i = \texttt{mum}$, contradicting that ans_i is a non-refused answer. Consequently,

$$\mathit{log} \not\models \Psi. \tag{6.23}$$

By (6.23) and the definition of the $\models$-operator, there exists a database instance r' with the following properties:

$$r' \models_{\mathsf{M}} \mathit{log} \quad \text{and} \tag{6.24}$$

$$r' \not\models_{\mathsf{M}} \Psi. \tag{6.25}$$

Since $\mathit{log} \models \mathit{ans}_i$ for every $\mathit{ans}_i \neq \texttt{mum}$ by (6.18), we can derive from (6.24) that

$$r' \models_{\mathsf{M}} \mathit{ans}_i \quad \text{for every } \mathit{ans}_i \neq \texttt{mum} \tag{6.26}$$

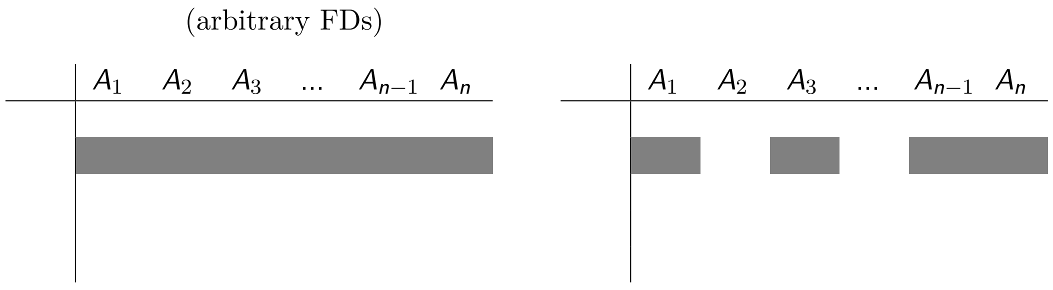
which is, by Definition 2.2, equivalent to

$$\mathit{eval}^*(\Phi_i)(r') = \mathit{ans}_i. \tag{6.27}$$

Furthermore, note that according to the censor definition (Definition 6.11) the decision whether or not an answer has to be refused is independent of the actual database instance; it consequently follows that

$$\begin{aligned} &\mathit{scqe}(\langle\Phi_i\rangle)(r,\mathit{pot\_sec}) = \langle\texttt{mum}\rangle \quad \text{if and only if} \\ &\mathit{scqe}(\langle\Phi_i\rangle)(r',\mathit{pot\_sec}) = \langle\texttt{mum}\rangle. \end{aligned} \tag{6.28}$$

Now we are ready to complete the proof:



(a) schema and query restrictions (b) policy restrictions

Figure 6.1: Visualization of the restrictions for select-queries.

- Since *scqe* either returns the correct answer to a query or refuses the answer, (6.27) together with (6.28) imply the first property: $scqe(Q')(r,pot\_sec) = scqe(Q')(r',pot\_sec)$.
- By (6.25), also the second property is satisfied: $eval^*(\Psi)(r') = \neg\Psi$.
- By (6.24), (6.18) and $log_0 = \Sigma$, r' satisfies the semantic constraints Σ and is thus an instance of the assumed relation schema.
- Finally, the pair (r',pot_sec) is admissible in the sense of Definition 5.3 (for pot_sec being known):
  - (6.24) and (6.18) lead to $r' \models_M log_0$;
  - (6.25) together with (6.18) yield $log_0 \not\models \Psi$ (for every $\Psi \in pot\_sec$).

As a result, r' serves as an alternative instance in the sense of Definition 5.4 and thus *scqe* is secure. □

From Definition 6.11 it is easy to see that stateless CQE considers each query of a sequence isolatedly and thus no user log is needed any longer. The insight that confidentiality is preserved in spite of that, as expressed in Theorem 6.12, is a first step in enforcing the declarative goals of inference control with the operational means of access control. After having enhanced the stateless CQE in the subsequent chapter, we will develop an algorithmic implementation of our approach in Part III of this thesis.

For each combination of query language and policy language being investigated in this thesis, we will provide a visualization of the respective restrictions that guarantee the preservation of confidentiality. In general, these restrictions may affect schema design, query language and policy language. In Figure 6.1, the restrictions for confidentiality-preserving answering of select-queries by means of a stateless

CQE are visualized by means of a “generic” relation with attributes $A_1, A_2, \dots, A_n$; Figure 6.1(a) visualizes schema and query language restrictions, whereas Figure 6.1(b) visualizes policy language restrictions. Actually, for select-queries the schema design does not need to be restricted at all, thus arbitrary FDs may be declared. The query language must be restricted to (full) tuples, which is indicated by the gray table row in Figure 6.1(a). The policy language must be restricted to existential R-sentences, that is, parts of tuples; this is indicated by the interrupted gray table row in Figure 6.1(b), where the gray parts indicate those positions that are instantiated with a constant (all other positions are assumed to be instantiated with existentially quantified variables).

6.4 Complexity Considerations

As stated in Section 5.1, in this thesis we aim at developing an inference control mechanism that is efficiently implementable. In Chapter 9, we will present algorithms for our approach and show that they are efficient in the sense of computational complexity. In this section, we want to justify that our investigations are headed in the right direction, that is, we roughly outline an efficient implementation of the censor $\mathit{censor}_{\exists R}$ as introduced in Definition 6.11. In the following, we consider isolated queries rather than query sequences. Moreover, we adopt the assumptions from Section 6.3, that is, we consider queries and potential secrets from the languages $\mathscr{L}_R$ and $\mathscr{L}_{\exists R}$, respectively.

According to Definition 6.11, answering a query is actually a process that consists of two steps:

1. The censor is invoked;
2. a) in case the censor returns `true`: `mum` is returned,
   b) otherwise: the query is answered ordinarily.

We first analyze the censor invocation. In the context of stateful CQE approaches, the censor has to check whether the (uncontrolled) answer to the query in combination with the user log enables the user to compute harmful logical implications. The decision problem for logical implication is, however, known to be costly or (for first-order logic) even undecidable in general.

Although Definition 6.11 suggests that in the scenario of Section 6.3 the censor still has to check for logical implications, we know from Definition 6.4 and Lemma 6.5 that this inference control can in fact be reduced to some kind of pattern matching as explained in the following example.

Example 6.13. Assume $\langle R, \mathcal{U}, \emptyset\rangle$ with $\mathcal{U} = \{A, B, C\}$ as underlying relation schema and let a query and a policy be given by

$$\Phi \equiv R(a,b,c) \quad \text{and}$$
$$pot\_sec = \{\Psi\} \quad \text{with } \Psi \equiv (\exists X_A)R(X_A,b,c),$$

respectively. According to Definition 6.11, $censor_{\exists R}(pot\_sec,\Phi)$ has to check if $\Phi \models \Psi$ when answering Φ. By Definition 6.4 and Lemma 6.5, $\Phi \models \Psi$ is equivalent to $\Phi[A] = \Psi[A]$ for every $A \in select(\Psi)$ which is effectively checked by the censor:

$$\left.\begin{array}{l} select(\Psi) = \{B,C\} \\ \Phi[B] = b = \Psi[B] \\ \Phi[C] = c = \Psi[C] \end{array}\right\} \Psi \text{ is relevant for } \Phi\text{, the censor returns } \texttt{true}.$$

This test can obviously be performed in linear time in the number of attributes. ◊

The above given Example 6.13 assumes a singleton policy and thus oversimplifies the censor decision. If the policy contains multiple elements, a pattern matching must be carried out for each of these elements. Remember that the policy elements may be interpreted as generalized tuples (see Section 6.1), and therefore we assume these elements to be stored as a relation instance. More precisely, for each relation schema and the corresponding instance of a database we assume an additional schema and an instance, respectively, that represents the declared policy. We call these additional schema and instance *classification schema* and *classification instance*, respectively. Formally, they are defined as follows.

Definition 6.14 (Classification schema). *Let $R_S = \langle R, \mathcal{U}, \Sigma\rangle$ be a relation schema with the set $Const$ of constants. The* classification schema *of R_S is given by $R_S^{pot\_sec} = \langle R^{pot\_sec}, \mathcal{U}, \emptyset\rangle$ and $Const^{pot\_sec} = Const \cup \{\#\}$ such that $\# \notin Const$.*

Definition 6.15 (Classification instance). *Let R_S be a relation schema with the classification schema $R_S^{pot\_sec}$ and $pot\_sec \subset \mathscr{L}_{\exists R}$ a policy. The* classification instance *with respect to pot_sec, denoted by r^{pot_sec}, is defined as follows:*

$$\begin{array}{rl} r^{pot\_sec} := \{ & R^{pot\_sec}(v_1^*,v_2^*,\dots,v_n^*) \mid \\ & \textit{exists } \Psi \equiv (\exists X_1)\dots(\exists X_m)R(v_1,v_2,\dots,v_n) \in pot\_sec \\ & \textit{such that for every } i \in \{1,2,\dots,n\}: \\ & v_i \in Const \Rightarrow v_i^* = v_i, \\ & v_i \in Var \Rightarrow v_i^* = \# \quad \} \end{array}$$

Example 6.16. Reconsidering relation schema and policy from Example 6.13, the classification instance r^{pot_sec} emerges from pot_sec by replacing each occurring variable with the "new" constant symbol #: $r^{pot\_sec} = \{R^{pot\_sec}(\#,b,c)\}$. ◊

Given a query Φ, a policy $\mathit{pot\_sec}$, and the corresponding classification instance $r^{pot-sec}$, an implementation of $\mathit{censor}_{\exists R}$ has to check if there is a tuple $\mu \in r^{pot-sec}$ such that $\Phi[A] = \mu[A]$ if $\mu[A] \neq \#$. It is easy to see that this is the case if and only if there exists a $\Psi \in \mathit{pot\_sec}$ such that Ψ is relevant for Φ. We now sketch two algorithms for this relevance check.

The first algorithm scans through the tuples of $r^{pot-sec}$ and checks whether the tuple μ under consideration satisfies the condition for relevance ($\Phi[A] = \mu[A]$ if $\mu[A] \neq \#$). With n denoting the number of attributes of the underlying relation and m denoting the number of tuples in $r^{pot-sec}$, that is, the size of the policy, the complexity of this algorithm can be estimated by $\mathcal{O}(m \cdot n)$.

The second algorithm constructs the set P_Φ of all tuples that satisfy the condition for relevance; this can be done by successively replacing each subset of constants in Φ with $\#$, for example, if $\Phi \equiv R(a,b,c)$ then

$$\begin{aligned} P_\Phi = \{ \quad & R(a,b,c), R(\#,b,c), R(a,\#,c), R(a,b,\#), R(\#,\#,c), R(\#,b,\#), \\ & R(a,\#,\#), R(\#,\#,\#) \quad \}. \end{aligned}$$

Then the algorithm checks if at least one of the tuples in P_Φ occurs in $r^{pot-sec}$; if so, it returns `true` as censor decision. We estimate the complexity of this algorithm in two steps:

- The complexity of the construction of P_Φ depends on the number n of attributes in the underlying relation. For each subset of these attributes, an element is added to P_Φ. Consequently, $|P_\Phi| = 2^n$ and thus the complexity of this step can be estimated by $\mathcal{O}(2^n)$.

- For analyzing the second step, we assume that the classification instance $r^{pot-sec}$ is indexed by a B-tree. It has been shown by Bayer and McCreight [BM72] that the height h of a B-tree is bounded by $h \leq 1 + \mathit{log}_{k+1}\left(\frac{m+1}{2}\right)$, where k is the minimal number of entries in each node of the tree[8] and m is the number of tuples in $r^{pot-sec}$. In the worst case, the B-tree must be searched for each element of P_Φ. The complexity for a single search operation is bounded by the height h of the B-tree and thus the complexity of the entire second step is bounded by $2^n \cdot (1 + \mathit{log}_{k+1}\left(\frac{m+1}{2}\right))$. Note that k is an implementation-dependent constant, so the complexity of the second step can be estimated by $\mathcal{O}(2^n \cdot \mathit{log}(m))$.

Altogether, the complexity of the proposed censor algorithm can be estimated by $\mathcal{O}(2^n + 2^n \cdot \mathit{log}(m))$ for a single query.

Whether the first or the second algorithm should be chosen obviously depends on the size of the parameters m and n. Assuming that the optimal choice is made, the

[8]More precisely, each node of the B-tree holds between k and $2k$ entries.

censor complexity is $\mathcal{O}(\mathit{min}\{m \cdot n, 2^n + 2^n \cdot \mathit{log}(m)\})$. According to Definition 6.11, $\mathit{censor}_{\exists R}$ has a policy $\mathit{pot\_sec}$ and a query Φ as input parameters. In particular, we do not regard the database schema as input but assume it to be set up in advance by the database administrator. Thus, the number n of attributes, which is part of the database schema, is considered a constant hereafter. As a consequence, the complexity of the censor algorithm simplifies to $\mathcal{O}(\mathit{min}\{m, \mathit{log}(m)\}) = \mathcal{O}(\mathit{log}(m))$.

Having estimated the complexity of the censor, it remains to investigate the complexity of the ordinary query evaluation which is performed if the censor returns `false`. The evaluation of a select-query is performed by scanning the tuples of the relation instance r for the correct instantiation of the selection attributes. Provided that the tuples of r are indexed by a suitable data structure, for example, a B-tree, the ordinary evaluation of a query $\Phi \in \mathscr{L}_R$ can thus be carried out in time $\mathcal{O}(\mathit{log}(|r|))$, where $|r|$ denotes the size of r, that is, the number of tuples in the instance.

We summarize the insights of this section in the following proposition.

Proposition 6.17 (scqe is efficient). *Let $R_S = \langle R, \mathcal{U}, \Sigma \rangle$ be a relation schema, r an instance of R_S, $\mathit{pot\_sec} \subset \mathscr{L}_{\exists R}$ a policy of size m, and $\Phi \in \mathscr{L}_R$ a query. The controlled evaluation of Φ according to Definition 6.11, $\mathit{scqe}(\langle \Phi \rangle)(r, \mathit{pot\_sec})$, takes time $\mathcal{O}(\mathit{log}(m) + \mathit{log}(|r|))$.*

6.5 Summary

In this chapter, we investigated a scenario for stateless CQE in relational databases. This scenario is rather basic which manifests in restricted query and policy languages. More precisely, the proposed query language enables a database user to ask for full tuples, that is, to express (closed) select-queries; the proposed policy language is the set of existential R-sentences, that is, it enables a security administrator to protect parts of tuples.

We identified and formally proved some useful properties of existential R-sentences and developed a stateless CQE mechanism. This mechanism basically reduces the computation of inferences to a simple pattern matching algorithm. We showed that our mechanism is both secure and efficient, that is, confidentiality-preserving and implementable by a logarithmic-time algorithm, respectively.

Chapter 7

Enhancements of the Languages

In the previous chapter, we considered a basic case for the preservation of confidentiality under inferences in relational databases. We formally proved that our approach is confidentiality-preserving under the assumption of simple policy and query languages, thereby restricting the security administrator as well as the user who sends queries to the database in their expressive means.

From a practical point of view, the assumed restrictions are not acceptable: The proposed languages for queries and elements of the confidentiality policy are not expressive enough with regard to a convenient database management system. In the following, we shortly sketch some desirable enhancements of the restricted languages from the view of the database user as well as from the view of the security administrator.

The Database User The database user is only able to express elementary queries with a rather coarse granularity so far. Particularly, he cannot express the following kinds of queries with the means at his disposal:

- *Select-project-queries*, that is, queries that can be expressed by using the selection and the projection operation in the relational algebra. In the relational calculus, selection is expressed by constants and projection is expressed by existentially quantified variables. For example, regarding the relation ACC from Example 5.1, the user possibly wants to know if bank A has a client called Smith. In terms of relational calculus, this can be expressed by

  $$(\exists X_{ACC\_NO})(\exists X_{BAL})ACC(bankA,X_{ACC\_NO},smith,X_{BAL}).$$

  The query language $\mathscr{L}_R$ that we investigated in Chapter 6, however, does not enable the user to express queries containing existentially quantified variables.

- *Boolean queries*, that is, queries that are composed from elementary (select-project-) queries using Boolean operations. In the banking example, the user might want to find out if client Jones has accounts at bank A as well as at bank B.[9] This can be expressed in relational calculus as follows:

$$(\exists X_{ACC\_NO})(\exists X_{BAL})ACC(bankA, X_{ACC\_NO}, jones, X_{BAL}) \wedge$$
$$(\exists X_{ACC\_NO})(\exists X_{BAL})ACC(bankB, X_{ACC\_NO}, jones, X_{BAL})$$

Again, $\mathscr{L}_R$ does not allow to express this kind of queries.

- *Open queries*, that is, queries with free variables, asking for variable assignments that make the query true in the database instance. Regarding Example 5.1 again, the database user possibly wants to retrieve all clients of bank B who have an account balance of \$ 20,000. The corresponding relational calculus query is

  $$(\exists X_{ACC\_NO})ACC(bankB, X_{ACC\_NO}, X^f_{ACC\_H}, 20000)$$

  with $X^f_{ACC\_H}$ denoting a free variable. This query is not expressible in $\mathscr{L}_R$ either.

The Security Administrator The security administrator is only able to express existential R-sentences as secrets so far. It is, however, desirable, that he has a more expressive language at his disposal. For example, consider the following issues:

- *Boolean secrets*, that is, Boolean combinations of existential R-sentences. Regarding Example 5.1, the security administrator might want to protect that client Anderson has an account at bank A or at bank B.[10] This may be expressed using disjunction:

  $$(\exists X_{ACC\_NO})(\exists X_{BAL})ACC(bankA, X_{ACC\_NO}, anderson, X_{BAL}) \vee$$
  $$(\exists X_{ACC\_NO})(\exists X_{BAL})ACC(bankB, X_{ACC\_NO}, anderson, X_{BAL})$$

  So far, however, the security administrator is not allowed to use disjunction.

- *Secret schemas*, that is, formulas that cover several secrets at a time as some kind of "template". In the banking example, the ACC relation is possibly published with the restriction that no account numbers from bank A must be readable. Instead of specifying a secret for every (potential) client of bank A, the security administrator could specify a single formula capturing the desired information by using free variables:

  $$(\exists X_{BAL})(\exists X_{ACC\_H})ACC(bankA, X^f_{ACC\_NO}, X_{ACC\_H}, X_{BAL})$$

[9] Disjunctive queries like "Does client Jones have an account at bank A *or* at bank B?" turn out to be harmful, see Subsection 7.2.1.

[10] Conjunctive secrets like "Client Anderson has an account at bank A *and* at bank B." turn out to be harmful, see Subsection 7.2.2.

This formula represents the set of secrets that emerges from replacing $X^f_{ACC\_NO}$ with every possible constant. So far, however, the security administrator is not allowed to use free variables.

In general, these lists of enhancements for the query language and the policy language raise no claim to completeness. However, for the parameters fixed in Section 5.1, we consider this selection of enhancements reasonable. For example, considering real-world database applications, it would also be desirable to express join queries which are outside the scope of this thesis. In Section 11.2, however, we provide some reflections on how to deal with join queries.

In the following, we aim at developing a confidentiality-preserving query answering system for relational databases that supports the kinds of queries and potential secrets sketched above. Therefore, we will investigate several enhancements of our basic languages in the following Sections 7.1–7.5. Our goal is to substantially relax the restrictions for the database user and the security administrator on the one hand and to maintain the preservation of confidentiality on the other hand.

In reaching this goal, as we shall see, relaxing one restriction often requires us to tighten another restriction. This makes it difficult to determine a "maximal" enhancement of our languages that improves expressiveness and allows for an efficient preservation of confidentiality at the same time. Nevertheless, in Section 7.6 we present a tabular summary of our achievements which outlines the space of possibilities being investigated in Sections 7.1–7.5.

7.1 Select-Project-Queries

First, we consider an enhancement of the query language referring to the granularity of expressible queries. Instead of restricting the expressiveness to the tuple level, as in Chapter 6, we consider the next finer level—the "partial tuple level". As already explained in Chapter 6, a partial tuple, expressed by an existential R-sentence, may be considered as a projection of a full tuple to a subset of the attributes. Therefore, partial tuple queries are also called select-project-queries. For example, considering a relation R over the attributes ABC, the database user can express queries of the form $\Phi \equiv (\exists X_B) R(a,X_B,c)$ where a, c denote constant symbols and X_B denotes a variable. This query asks whether or not the value combination (a,c) occurs as instantiation of the attributes A and C in the actual database instance.

In Subsection 7.1.1, we point out the arising problems when enabling the user to express existential R-sentences as queries. Moreover, we develop a solution to these problems in form of additional schema- and policy-restrictions for our stateless CQE mechanism. In Subsection 7.1.2, we formally show that under these restrictions the stateless CQE is confidentiality-preserving.

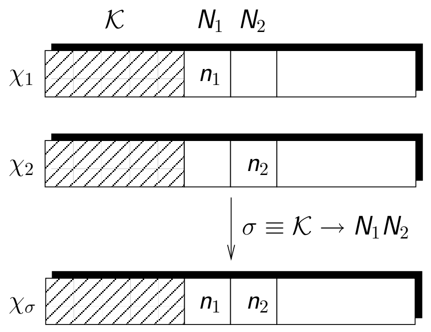


Figure 7.1: Application of FDs.

Select-project-queries have originally been investigated by Biskup, Embley and Lochner [BEL08], whose work serves as a basis for this section.

7.1.1 Query Language and Arising Problems

We now assume that the query language as well as the policy language are given by $\mathscr{L}_{\exists R}$, that is, the language of select-project-queries, as introduced in Definition 6.2.

In contrast to the basic query language containing only select-queries, as investigated in Chapter 6, functional dependencies play a major role in the context of select-project-queries. More precisely, when the database user is able to express select-project-queries, he might exploit the declared FDs together with the existential R-sentences he receives as answers to disclose a potential secret. We say the user *applies* FDs to existential R-sentences. For example, consider Figure 7.1: χ_1 and χ_2 are existential R-sentences agreeing on the $\mathcal{K}$-attributes which are supposed to be instantiated with constants. Applying the FD $\sigma \equiv \mathcal{K} \rightarrow N_1N_2$ now means to "collect" the N_1- and N_2-constants from χ_1 and χ_2 in a new formula χ_σ as depicted in the figure. Obviously, N_1 and N_2 must not be instantiated with different constants in χ_1 and χ_2, since otherwise the FD σ would be violated and thus not be applicable. These notions shall now be defined formally and illustrated by an example afterwards.

Definition 7.1 (Compatibility and applicability of FDs). *Consider two existential R-sentences $\chi_1, \chi_2 \in \mathscr{L}_{\exists R}$ with the same underlying relation schema R_S and an FD σ. We call σ* compatible *with χ_1 and χ_2 if all attributes occurring in σ are attributes of R_S, that is, $\mathit{lhs}(\sigma) \cup \mathit{rhs}(\sigma) \subseteq \mathcal{U}$. Moreover, σ is* applicable *to χ_1 and χ_2 if*

- *σ is compatible with χ_1 and χ_2,*
- *$\chi_1[K] = \chi_2[K] \in \mathit{Const}$ for all attributes $K \in \mathit{lhs}(\sigma)$,*[11] *and*

[11] If $\chi_1[K], \chi_2[K] \in \mathit{Var}$ then $\chi_1[K] \neq \chi_2[K]$ due to the remark to Definition 6.2.

- $\chi_1[N] = \chi_2[N]$ *for all attributes* $N \in \mathit{select}(\chi_1) \cap \mathit{select}(\chi_2) \cap \mathit{rhs}(\sigma)$.

The result χ_σ *of applying* σ *to* χ_1 *and* χ_2 *is defined as follows:*[12]

$$\chi_\sigma[A] := \begin{cases} \chi_1[A] & \text{if } A \in \mathit{select}(\chi_1), \\ \chi_2[A] & \text{otherwise.} \end{cases}$$

Example 7.2. Let $R_S = \langle R, \mathcal{U}, \emptyset \rangle$ be a relation schema with $\mathcal{U} = \{A, B, C, D\}$ and $\chi_1, \ldots, \chi_4 \in \mathscr{L}_{\exists R}$ existential R-sentences with the underlying schema R_S:

$$\begin{aligned} &\chi_1 \equiv (\exists X_D) R(a,b,c,X_D) \\ &\chi_2 \equiv R(a,c,b,d) \\ &\chi_3 \equiv (\exists X_C) R(a,b,X_C,d) \\ &\chi_4 \equiv (\exists X_B) R(a,X_B,c,d) \end{aligned}$$

Now consider the FD $\sigma \equiv AB \rightarrow CD$ which is compatible with $\chi_1, \ldots, \chi_4$ because $\mathit{lhs}(\sigma) \cup \mathit{rhs}(\sigma) = \{A, B, C, D\} \subseteq \mathcal{U}$. Then for example the following statements with respect to applicability hold:

- σ is applicable to χ_1 and χ_3 because
  - $\mathit{lhs}(\sigma) = \{A, B\}$ and $\chi_1[A] = \chi_3[A]$ as well as $\chi_1[B] = \chi_3[B]$;
  - $\mathit{select}(\chi_1) \cap \mathit{select}(\chi_3) \cap \mathit{rhs}(\sigma) = \{A, B, C\} \cap \{A, B, D\} \cap \{C, D\} = \emptyset$.

  The result of applying σ to χ_1 and χ_3 is χ_σ with $\chi_\sigma[A] = \chi_1[A]$, $\chi_\sigma[B] = \chi_1[B]$, $\chi_\sigma[C] = \chi_1[C]$ and $\chi_\sigma[D] = \chi_2[D]$, that is, $\chi_\sigma \equiv R(a,b,c,d)$.

- σ is not applicable to χ_1 and χ_2 because $B \in \mathit{lhs}(\sigma)$ but $\chi_1[B] \neq \chi_2[B]$.

- σ is not applicable to χ_1 and χ_4 because $B \in \mathit{lhs}(\sigma)$ but $\chi_4[B] \notin \mathit{Const}$.

(In fact, σ is only applicable to χ_1 and χ_3 in this example.) $\Diamond$

The application of an FD is "logically sound" in the sense that the result of such an application is logically implied by the FD and the existential R-sentences it is applied to. This is captured by the following lemma.

Lemma 7.3 (FD application is sound). *Let* $\chi_1, \chi_2 \in \mathscr{L}_{\exists R}$ *and* σ *an FD being applicable to* χ_1 *and* χ_2*. Then for the result of applying* σ *to* χ_1 *and* χ_2*, denoted by* χ_σ *it holds that* $\{\chi_1, \chi_2, \sigma\} \models \chi_\sigma$.

[12] Observe that, due to the definition of applicability, the application of an FD is (up to variable identifiers) independent of the order of χ_1 and χ_2.

Proof. According to the definition of logical implication, $\{\chi_1, \chi_2, \sigma\} \models \chi_\sigma$ if for all instances r it holds that

$$\text{if } r \models_{\mathsf{M}} \chi_1, r \models_{\mathsf{M}} \chi_2 \text{ and } r \models_{\mathsf{M}} \sigma \text{ then } r \models_{\mathsf{M}} \chi_\sigma. \tag{7.1}$$

We choose an instance r and assume that $r \models_{\mathsf{M}} \chi_1$, $r \models_{\mathsf{M}} \chi_2$ and $r \models_{\mathsf{M}} \sigma$. Then, by Lemma 6.7, there exist ground atoms χ_{g_1} and χ_{g_2} with

$$r \models_{\mathsf{M}} \chi_{g_i} \text{ and } \chi_{g_i} \models \chi_i \text{ for } i \in \{1, 2\}. \tag{7.2}$$

According to Definition 6.4 and Lemma 6.5, each χ_{g_i} agrees with the corresponding χ_i on the constants:

$$\text{if } \chi_i[A] \in \mathit{Const} \text{ then } \chi_{g_i}[A] = \chi_i[A] \text{ for } i \in \{1, 2\}. \tag{7.3}$$

Since σ is applicable to χ_1, χ_2, these existential R-sentences must be instantiated with the same constants on $\mathit{lhs}(\sigma)$ (see Definition 7.1) and thus, by (7.3), also χ_{g_1} and χ_{g_2} are instantiated with these constants on $\mathit{lhs}(\sigma)$. Regarding the first-order logic representation of FDs (2.1), from χ_{g_1} and χ_{g_2} agreeing on $\mathit{lhs}(\sigma)$ it follows that

$$\chi_{g_1}[A] = \chi_{g_2}[A] \text{ for all } A \in \mathit{rhs}(\sigma). \tag{7.4}$$

Thus, by construction of χ_σ, we get

$$\chi_\sigma[A] = \chi_{g_i}[A] \text{ for all } A \in \mathit{select}(\chi_{g_i}) \tag{7.5}$$

for $i \in \{1, 2\}$ according to Definition 7.1. Applying Definition 6.4 and Lemma 6.5 again, we conclude that

$$\chi_{g_i} \models \chi_\sigma \text{ for } i \in \{1, 2\} \tag{7.6}$$

which, together with (7.2), leads to $r \models_{\mathsf{M}} \chi_\sigma$. □

Remark. In Section 7.3, we will revisit the notion of applicability of FDs when we define the chase of existential R-sentences. Basically, this mechanism "exhaustively applies" a set of FDs to a set of existential R-sentences, see Definition 7.28.

We now investigate the impact of enabling the user to formulate select-project-queries. Astonishingly enough, already the slight modification of the query language $\mathscr{L}_R$ by allowing the use of existentially quantified variables for expressing select-project-queries may lead to the disclosure of potential secrets. Consider the following example:

Example 7.4. Reconsider the relation schema ACC_S and the instance acc from Example 5.1. We assume $\mathscr{L}_{\exists R}$ as query and policy language and consider the policy $\mathit{pot\_sec} = \{\Psi\}$ with

$$\Psi \equiv (\exists X_{BANK})(\exists X_{ACC\_NO})\mathit{ACC}(X_{BANK}, X_{ACC\_NO}, \mathit{jones}, 2500).$$

A user with a priori knowledge $\mathit{log}_0 = \Sigma$, that is, $\mathit{log}_0 = \{\{\mathit{BANK}, \mathit{ACC\_NO}\} \rightarrow \{\mathit{ACC\_HOLDER}, \mathit{BALANCE}\}\}$, now sends the following queries to the database:

$$\Phi_1 \equiv (\exists X_{ACC\_H})\mathit{ACC}(\mathit{bankA}, 102, X_{ACC\_H}, 2500)$$
$$\Phi_2 \equiv (\exists X_{BAL})\mathit{ACC}(\mathit{bankA}, 102, \mathit{jones}, X_{BAL})$$

Evaluating Φ_1 and Φ_2 with the stateless CQE scqe, as introduced in Definition 6.11, leads to $\mathit{ans}_1 = \mathit{eval}^*(\Phi_1)(\mathit{acc}) = \Phi_1$ and $\mathit{ans}_2 = \mathit{eval}^*(\Phi_2)(\mathit{acc}) = \Phi_2$, that is, none of the answers is refused, and consequently, the updated user log after the answer to Φ_2 is

$$\begin{aligned} \mathit{log}_2 = \{ \; & \{\mathit{BANK}, \mathit{ACC\_NO}\} \rightarrow \{\mathit{ACC\_HOLDER}, \mathit{BALANCE}\}, \\ & (\exists X_{ACC\_H})\mathit{ACC}(\mathit{bankA}, 102, X_{ACC\_H}, 2500), \\ & (\exists X_{BAL})\mathit{ACC}(\mathit{bankA}, 102, \mathit{jones}, X_{BAL}) \quad \}. \end{aligned}$$

The user can now apply the FD

$$\sigma \equiv \{\mathit{BANK}, \mathit{ACC\_NO}\} \rightarrow \{\mathit{ACC\_HOLDER}, \mathit{BALANCE}\}$$

to the returned answers according to Definition 7.1. He gets

$$\chi_\sigma \equiv \mathit{ACC}(\mathit{bankA}, 102, \mathit{jones}, 2500)$$

as result of the application. It obviously holds that $\chi_\sigma \models \Psi$. $\Diamond$

In order to protect against this exploitation of FDs, we must analyze the problem in more detail. In Example 7.4, the instantiation of the key attributes of the relation schema serves as tuple identifier: Since ans_1 and ans_2 agree on the instantiation of BANK and $\mathit{ACC\_NO}$, the user knows that the answers refer to the same tuple. Consequently, the harmful instantiation $(\mathit{jones}, 2500)$ of the attributes $\mathit{ACC\_HOLDER}$ and $\mathit{BALANCE}$ may be distributed to different queries, thereby taking advantage of the “obliviousness” of the system: When the user asks for the combination $\mathit{BANK} = \mathit{bankA}$, $\mathit{ACC\_NO} = 102$ and $\mathit{ACC\_HOLDER} = \mathit{jones}$, the system “forgot” that he is already aware of the fact that the combination $\mathit{BANK} = \mathit{bankA}$, $\mathit{ACC\_NO} = 102$ and $\mathit{BALANCE} = 2500$ occurs in acc.

Observe that the key property of the attributes $\mathit{BANK}, \mathit{ACC\_NO}$, expressed by the set Σ, is crucial: If the security administrator is allowed to express arbitrary secrets (in the language $\mathscr{L}_{\exists R}$), FDs may be exploited to disclose these secrets, as illustrated by Example 7.4. Intuitively, the disclosure of a secret "connects" two or more non-key attribute values via the key of the relation schema by applying an FD to two or more existential R-sentences. Consequently, the security administrator should not be able to declare secrets that protect more than one non-key attribute value.

Moreover, not only key constraints may be exploited for the disclosure of secrets: Suppose that the ACC-relation from Example 5.1 had an additional attribute $\mathit{CONSULTANT}$ that captures for each account holder the contact person of the respective bank. Then $\{\mathit{BANK}, \mathit{ACC\_NO}\}$ would not be a key any longer, since $\Sigma \not\models \{\mathit{BANK}, \mathit{ACC\_NO}\} \rightarrow \{\mathit{CONSULTANT}\}$ and thus $\Sigma \not\models \{\mathit{BANK}, \mathit{ACC\_NO}\} \rightarrow \mathcal{U}$. Nevertheless, the FD $\{\mathit{BANK}, \mathit{ACC\_NO}\} \rightarrow \{\mathit{ACC\_HOLDER}, \mathit{BALANCE}\}$ might still be exploited as in Example 7.4.

On the one hand, if we allow the database administrator to declare arbitrary FDs, we are not able to control all possible "connections" of attribute values in the sense of Example 7.4. On the other hand, the database administrator should at least be able to declare the key constraints for the relation schema. We therefore assume the relation schema to be in ONF (see Section 2.2) hereafter which basically means that

- the schema has a unique key $\mathcal{K}$ and
- the declared FDs are key dependencies, that is, all left-hand sides are supersets of $\mathcal{K}$.

With this schema restriction it is possible to control the harmful "attribute value connections" in the sense of Example 7.4 by slightly restricting the policy language. In the following, we (syntactically) define this policy language restriction.

In relational databases, non-key attributes are also called *property attributes* because they describe the properties of objects (represented by tuples) that are uniquely determined by the instantiation of the key attributes. Therefore, a part of a tuple (expressed by a select-project-query) that consists of key attribute values and at most one property attribute value carries in some sense a "minimal information" about the object it refers to. We consequently call such tuple parts *basic facts*. If χ is a basic fact then $\mathit{select}(\chi)$, the set of attributes being instantiated with constants in χ, is a *basic fact schema*. Formally, basic fact schemas are defined as follows.

Definition 7.5 (Basic fact schemas). *Let $R_S = \langle R, \mathcal{U}, \Sigma\rangle$ be a relation schema in ONF. The set of* basic fact schemas *of R_S is then defined by*

$$\mathit{bfs}(R_S) := \bigcup_{\mathcal{S}' \in \mathcal{S}} \wp(\mathcal{S}') \backslash \emptyset$$

with $\mathcal{S} := \{\mathcal{K} \cup \{N\} \mid \mathcal{K}$ *is key of* $R_S, N \in \mathcal{U} \backslash \mathcal{K}\}$,

where $\wp$ is the powerset operator.

Remark. Observe that $bfs(R_S)$ in particular contains all attributes of R_S as singletons: Each set $\mathcal{K} \cup \{N\}$ consists of the key and one additional attribute from $\mathcal{U} \backslash \mathcal{K}$, thus in $\mathcal{S}$ each attribute from $\mathcal{U}$ occurs at least once; since $bfs(R_S)$ consists of all subsets of the elements of $\mathcal{S}$, each attribute from $\mathcal{U}$ occurs in $bfs(R_S)$ as singleton.

We illustrate the notions of basic fact schemas and basic facts with the following example.

Example 7.6. Reconsider the relation schema ACC_S from Example 5.1. According to Definition 7.5, the set $\mathcal{S}$ is given by

$$\mathcal{S} = \{\{BANK, ACC\_NO, ACC\_HOLDER\}, \{BANK, ACC\_NO, BALANCE\}\},$$

and consequently, the basic fact schemas of ACC_S are

$$\begin{aligned} bfs(ACC_S) = \{\ & \{BANK\}, \{ACC\_NO\}, \{ACC\_HOLDER\}, \{BALANCE\}, \\ & \{BANK, ACC\_NO\}, \{BANK, ACC\_HOLDER\}, \\ & \{BANK, BALANCE\}, \{ACC\_NO, ACC\_HOLDER\}, \\ & \{ACC\_NO, BALANCE\}, \{BANK, ACC\_NO, ACC\_HOLDER\}, \\ & \{BANK, ACC\_NO, BALANCE\} \quad \}. \end{aligned}$$

In particular, the secret Ψ from Example 7.4 is *not* a basic fact, since

$$select(\Psi) = \{ACC\_HOLDER, BALANCE\} \notin bfs(ACC_S).$$

In contrast, the following potential secrets *are* basic facts (with $c_1, \ldots, c_6$ denoting arbitrary constants):

$\Psi_1 \equiv (\exists X_{ACC\_NO})(\exists X_{ACC\_H})(\exists X_{BAL}) ACC(c_1, X_{ACC\_NO}, X_{ACC\_H}, X_{BAL})$
is a basic fact, since $select(\Psi_1) = \{BANK\} \in bfs(ACC_S)$;

$\Psi_2 \equiv (\exists X_{BANK})(\exists X_{BAL}) ACC(X_{BANK}, c_2, c_3, X_{BAL})$
is a basic fact, since $select(\Psi_2) = \{ACC\_NO, ACC\_HOLDER\} \in bfs(ACC_S)$;

$\Psi_3 \equiv (\exists X_{ACC\_H}) ACC(c_4, c_5, X_{ACC\_H}, c_6)$
is a basic fact, since $select(\Psi_3) = \{BANK, ACC\_NO, BALANCE\} \in bfs(ACC_S)$.

(Note that Ψ_1, Ψ_2, Ψ_3 are only examples and thus this is not an exhaustive list of basic facts.) ◊

7.1.2 Preservation of Confidentiality

We claim that stateless CQE with $\mathscr{L}_{\exists R}$ as query and policy language is secure in the sense of Definition 5.4 if the relation schema is in ONF and each declared potential secret is a basic fact, that is, an instantiation of a basic fact schema according to Definition 7.5. We express this result formally in Theorem 7.10; for the proof, however, we need two preparatory lemmas which shall be stated in the following.

Lemma 7.7 (cqe is secure). *The stateful CQE cqe, as introduced in Section 3.2, is a secure query evaluation in the sense of Definition 5.4.*

This lemma originates from [BB01] and therefore we omit the proof. Together with Lemma 5.5 it follows that the user logs of cqe never imply a potential secret. Stateless CQE does not actually need the user log, but in the proof of Theorem 7.10 we show that stateless and stateful CQE are equivalent in this context and therefore we construct the user logs for stateless CQE in order to compare it to the user logs of stateful CQE.

The second lemma basically says that the restriction of a confidentiality policy to basic facts, as described in Subsection 7.1.1, results in the database user not being able to disclose secrets solely by applying FDs.

Lemma 7.8 (Basic facts prevent the exploitation of FDs). *Let χ_1 and χ_2 be existential R-sentences with the underlying relation schema R_S in ONF, $\Psi \in \mathscr{L}_{\exists R}$ a potential secret with the same underlying relation schema R_S and $\mathsf{select}(\Psi) \in \mathsf{bfs}(R_S)$, and $\sigma \in \Sigma$ an FD that is applicable to χ_1 and χ_2. Then for χ_σ, the result of applying σ to χ_1 and χ_2, it holds that $\chi_\sigma \models \Psi$ only if $\chi_1 \models \Psi$ or $\chi_2 \models \Psi$.*

Proof. By Definition 7.1, each constant occurring in χ_σ occurs in χ_1 or in χ_2 at the same position:

$$\chi_\sigma[A] \in \mathsf{Const} \text{ only if } \chi_1[A] = \chi_\sigma[A] \text{ or } \chi_2[A] = \chi_\sigma[A]. \tag{7.7}$$

Moreover, again by Definition 7.1, for all left-hand side attributes K of σ it holds that

$$\chi_\sigma[K] = \chi_1[K] = \chi_2[K] \ (\in \mathsf{Const}). \tag{7.8}$$

Now we assume that $\chi_\sigma \models \Psi$ holds and show that then also $\chi_1 \models \Psi$ or $\chi_2 \models \Psi$ holds. Because of $\mathsf{select}(\Psi) \in \mathsf{bfs}(R_S)$ and from Definition 7.5 we know that besides the left-hand sides attributes of σ, in Ψ at most one additional attribute N is instantiated by a constant. Since both χ_σ and Ψ are existential R-sentences and $\chi_\sigma \models \Psi$, we can apply Definition 6.4 and Lemma 6.5 and derive that for all $A \in \mathsf{select}(\Psi)$ it holds that $\chi_\sigma[A] = \Psi[A]$. In particular, $\chi_\sigma[N] = \Psi[N]$ and together with (7.7) we get

$$\chi_1[N] = \chi_\sigma[N] \text{ or } \chi_2[N] = \chi_\sigma[N]. \tag{7.9}$$

From (7.8), (7.9), $\mathsf{select}(\Psi) \in \mathsf{bfs}(R_S)$, and Lemma 6.5, we now conclude that $\chi_1 \models \Psi$ or $\chi_2 \models \Psi$ holds. □

We now verify the statement of Lemma 7.8 by means of the following example.

Example 7.9. The relation schema $R_S = \langle R, ABCD, A \to BCD \rangle$ (which is in ONF with A as unique key) has the basic fact schemas $\mathit{bfs}(R_S) = \{A, B, C, D, AB, AC, AD\}$. A potential secret Ψ in the sense of Lemma 7.8 is a basic fact, that is, it instantiates exactly one attribute set from $\mathit{bfs}(R_S)$. Moreover, two existential R-sentences χ_1, χ_2 in the sense of Lemma 7.8 must agree on attribute A such that $\sigma \equiv A \to BCD$ is applicable.

Consider $\chi_1 \equiv (\exists X_C)(\exists X_D)R(a,b,X_C,X_D)$ and $\chi_2 \equiv (\exists X_B)(\exists X_C)R(a,X_B,X_C,d)$. The FD σ is applicable to these existential R-sentences, the result of the application is $\chi_\sigma \equiv (\exists X_C)R(a,b,X_C,d)$. Here, the basic fact schemas A, B, D, AB and AD are instantiated, that is, it possibly holds that $\chi_\sigma \models \Psi$ for a potential secret Ψ. However, all of the instantiated basic fact schemas are already instantiated in χ_1 and/or χ_2 with the same constants as depicted in the following table:

basic fact schema	A	B	D	AB	AD
instantiation in χ_σ	a	b	d	(a,b)	(a,d)
instantiation in χ_1	a	b	–	(a,b)	–
instantiation in χ_2	a	–	d	–	(a,d)

From this representation, it is clear that if $\chi_\sigma \models \Psi$ then also $\chi_1 \models \Psi$ or $\chi_2 \models \Psi$. $\Diamond$

With Lemmas 7.7 and 7.8 we are now able to show that the stateless CQE *scqe* according to Definition 6.11 preserves confidentiality under the query language $\mathscr{L}_{\exists R}$ if we assume the underlying relation schema to be in ONF and restrict the potential secrets to basic facts.

Theorem 7.10 (*scqe* preserves confidentiality under $\mathscr{L}_{\exists R}$). *Consider a relation schema $R_S = \langle R, \mathcal{U}, \Sigma \rangle$ in ONF, an instance r of R_S, a confidentiality policy $\mathit{pot\_sec} \subset \mathscr{L}_{\exists R}$ with $\mathit{select}(\Psi) \in \mathit{bfs}(R_S)$ for every $\Psi \in \mathit{pot\_sec}$, and a query sequence $Q = \langle \Phi_1, \Phi_2, \ldots \rangle$ with $\Phi_1, \Phi_2, \ldots \in \mathscr{L}_{\exists R}$. Moreover, let $\mathit{log}_0 = \Sigma$ be the a priori user knowledge and assume that $(r,\mathit{pot\_sec})$ is admissible with respect to log_0. Then the stateless CQE $\mathit{scqe}(Q)(r,\mathit{pot\_sec})$ is secure in the sense of Definition 5.4.*

Proof. We recall the refusal censor from stateful CQE as investigated by Biskup and Bonatti [BB01] and already explained in Chapter 3 (see (3.2)):

$$\begin{aligned} &\mathit{censor}(\mathit{pot\_sec},\mathit{log},\Phi) := \\ &\quad \text{(exists } \Psi)[\Psi \in \mathit{pot\_sec} \text{ and } ((\mathit{log} \cup \{\Phi\} \models \Psi) \text{ or } (\mathit{log} \cup \{\neg\Phi\} \models \Psi))] \end{aligned} \tag{7.10}$$

This censor has already been proven secure in the sense of Definition 5.4 in [BB01]. We now show that $\mathit{censor}_{\exists R}$ from Definition 6.11 and the censor (7.10) essentially do the same, that is, the following statements are equivalent for each query Φ_i from Q:

$$\mathit{censor}(\mathit{pot\_sec},\mathit{log}_{i-1},\Phi_i) = \mathtt{true} \tag{7.11}$$

$$\mathit{censor}_{\exists R}(\mathit{pot\_sec},\Phi_i) = \mathtt{true} \tag{7.12}$$

Observe that "(7.12) $\Rightarrow$ (7.11)" is obviously satisfied: if $\Phi_i \models \Psi$ for a $\Psi \in \mathit{pot\_sec}$ then also $\mathit{log}_{i-1} \cup \{\Phi_i\} \models \Psi$ by the monotonicity of first-order logic.

We now show "(7.11) $\Rightarrow$ (7.12)" and first of all note that in the given context there are in principle four possibilities to draw an inference:

- inconsistencies, for example, the set $\{R(a), \neg R(a)\}$ allows for arbitrary inferences;
- application of FDs in the sense of Definition 7.1 and Lemma 7.3, for example,
  $$\{A \rightarrow BC, (\exists X_C)R(a,b,X_C), (\exists X_B)R(a,X_B,c)\} \models R(a,b,c);$$
- combinatorial effects, for example, given that $\mathit{Const} = \{a, b\}$, we get
  $$\{(\exists X_A)R(X_A), \neg R(a)\} \models R(b);$$
- weakening formulas, for example,
  $$R(a,b) \models (\exists X_A)R(X_A,b).$$

Since we only consider (subsets of) the assumed user knowledge as a basis for drawing inferences in the following, inconsistencies may be neglected: According to Proposition 3.1, the user knowledge is always true in the instance, $r \models_M \mathit{log}_i$ for all i, thus rendering an inconsistent user knowledge impossible. Combinatorial effects cannot occur either, since we assume Const to be an infinite set. In the following, we consider only harmful inferences, that is, inferences that lead to the disclosure of a secret; thus we may neglect weakening formulas as well because if a weakened formula allowed for a harmful inference then already the stronger formula would by transitivity: If for two formulas χ, χ' and a secret Ψ it holds that $\chi \models \chi'$ (meaning that χ' is a weakening of χ) and $\chi' \models \Psi$ then also $\chi \models \Psi$. As a result, we only have to consider (sequences of) inferences due to the application of FDs in the following.

We now accomplish the proof by contraposition and therefore show that $\mathit{censor}_{\exists R}(\mathit{pot\_sec},\Phi_i) = \texttt{false}$ only if $\mathit{censor}(\mathit{pot\_sec},\mathit{log}_{i-1},\Phi_i) = \texttt{false}$. According to Definition 6.11, $\mathit{censor}_{\exists R}(\mathit{pot\_sec},\Phi_i) = \texttt{false}$ is equivalent with

$$\Phi_i \not\models \Psi \text{ for all } \Psi \in \mathit{pot\_sec}, \tag{7.13}$$

and, according to (7.10), $\mathit{censor}(\mathit{pot\_sec},\mathit{log}_{i-1},\Phi_i) = \texttt{false}$ is equivalent with

$$\mathit{log}_{i-1} \cup \{\Phi_i\} \not\models \Psi \text{ and } \mathit{log}_{i-1} \cup \{\neg\Phi_i\} \not\models \Psi \text{ for all } \Psi \in \mathit{pot\_sec}. \tag{7.14}$$

We assume that (7.13) holds and consider the two parts of (7.14) separately:

$$\mathit{log}_{i-1} \cup \{\Phi_i\} \not\models \Psi \text{ for all } \Psi \in \mathit{pot\_sec} \tag{7.15}$$
$$\mathit{log}_{i-1} \cup \{\neg\Phi_i\} \not\models \Psi \text{ for all } \Psi \in \mathit{pot\_sec} \tag{7.16}$$

First, we show that (7.15) holds. Consider the set L_0 consisting of all query answers being contained in log_{i-1} and the query Φ_i:

$$L_0 := (log_{i-1} \cup \{\Phi_i\}) \backslash \Sigma \tag{7.17}$$

A set L_{j+1} emerges from its predecessor L_j by applying an FD $\sigma \in \Sigma$ to $\chi_1^j, \chi_2^j \in L_j$ (with σ being applicable to χ_1^j and χ_2^j in the sense of Definition 7.1) and adding the result χ_σ^j to L_j. We now prove by induction on j that for every $\chi \in L_j$ and for every $\Psi \in pot\_sec$ it holds that $\chi \not\models \Psi$ which, in turn, shows that (7.15) holds.

$j = 0$: By Lemmas 7.7 and 5.5, we know that $log_{i-1} \not\models \Psi$ for every $\Psi \in pot\_sec$. Moreover, $\Phi_i \not\models \Psi$ for every $\Psi \in pot\_sec$ by assumption (7.13). This leads to $\chi \not\models \Psi$ for every $\chi \in L_j$ and for every $\Psi \in pot\_sec$.

$j + 1$: By construction of the L_j-sets it holds that $L_{j+1} \backslash L_j = \{\chi_\sigma^j\}$. Since for every $\chi \in L_j$ and every $\Psi \in pot\_sec$ it holds that $\chi \not\models \Psi$ by the induction hypothesis, we have to show that

$$\chi_\sigma^j \not\models \Psi \text{ for every } \Psi \in pot\_sec. \tag{7.18}$$

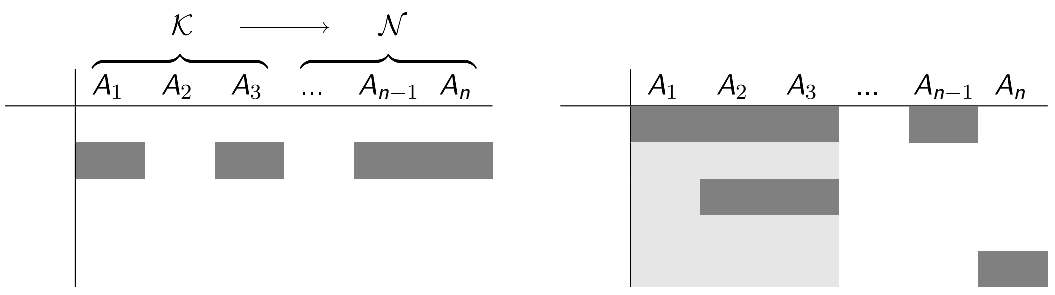
We indirectly show this by assuming that $\chi_\sigma^j \models \Psi$ for a $\Psi \in pot\_sec$. Since R_S is in ONF and $select(\Psi) \in bfs(R_S)$, we can apply Lemma 7.8 which leads to $\chi_1^j \models \Psi$ or $\chi_2^j \models \Psi$. However, since $\chi_1^j, \chi_2^j \in L_j$, this contradicts the induction hypothesis.

With the L_j-sets we simulated the application of FDs which has been identified as the only way to draw (harmful) inferences in our scenario. Therefore, the induction shows that arbitrary sequences of FD applications do not lead to the disclosure of secrets in the sense of (7.15).

Second, we show that (7.16) holds and therefore indirectly assume that $log_{i-1} \cup \{\neg\Phi_i\} \models \Psi$ for a $\Psi \in pot\_sec$. By Lemmas 7.7 and 5.5, $log_{i-1} \not\models \Psi$ for every $\Psi \in pot\_sec$, and thus $\neg\Phi_i$ must be involved in the deduction of Ψ. As argued above, in the considered setting inferences can only be drawn by the application of FDs, which is only possible with positive formulas. Thus, $\neg\Phi_i$ cannot be involved in the deduction of Ψ which leads to a contradiction.

Summarized, by assuming (7.13) we showed that (7.15) and (7.16) hold which implies that also (7.14) holds. This completes the proof. □

Remark. Theorem 7.10 requires that potential secrets are basic facts of the underlying relation. Note that we could additionally allow the declaration of the "fully quantified" secret $\Psi \equiv (\exists X_{A_1})(\exists X_{A_2}) \dots (\exists X_{A_n}) R(X_{A_1}, X_{A_2}, \dots, X_{A_n})$ without compromising



$\mathcal{K} \longrightarrow \mathcal{N}$
A_1 A_2 A_3 ... A_{n-1} A_n
A_1 A_2 A_3 ... A_{n-1} A_n

(a) schema and query restrictions (b) policy restrictions

Figure 7.2: Visualization of the restrictions for select-project-queries.

confidentiality, since *scqe* would then refuse every answer according to Definition 6.11. However, we consider this special case unreasonable with regard to availability of data and therefore refrain from adding Ψ to the policy language. If the security administrator actually aims at refusing any query for a specific user, he should rather employ a table-level access control mechanism than an inference control mechanism like stateless CQE. This is, however, out of the scope of this thesis.

The restrictions for confidentiality-preserving answering of select-project-queries, as demanded by Theorem 7.10, are depicted in Figure 7.2. Figure 7.2(a) visualizes schema and query language restrictions, and Figure 7.2(b) visualizes policy language restrictions. Regarding schema restrictions, the FD $\mathcal{K} \rightarrow \mathcal{N}$ in Figure 7.2(a) indicates that only FDs with the unique key $\mathcal{K}$ on the left-hand side[13] and a set of non-key attributes $\mathcal{N}$ on the right-hand side may be declared due to the ONF restriction (in this visualization, we assume $A_1A_2A_3$ as key without loss of generality). The query language is restricted to existential R-sentences, visualized by an interrupted table row as in Figure 7.2(b). Finally, an element of the policy, that is, a potential secret, is an existential R-sentence with the additional restriction that the "gray part" is a subset of the union of the key (table cells that belong to the key attributes are marked light gray in Figure 7.2(b)) and a single non-key attribute. Figure 7.2(b) shows three examples for potential secrets that comply with this "basic facts restriction" (see Definition 7.5).

[13]ONF actually allows supersets of $\mathcal{K}$ as left-hand side of an FD, but since $\mathcal{K} \rightarrow \mathcal{A}$ holds for every attribute set $\mathcal{A}$, the minimality assumption according to Definition 2.1 implies that no proper superset of $\mathcal{K}$ occurs as left-hand side of an FD.

7.2 Boolean Expressions

This section deals with Boolean combinations of select-project-queries. More specifically, we investigate whether and how the Boolean operations negation, conjunction and disjunction can be used together with existential R-sentences in query and policy languages. It is well-known that all other kinds of Boolean operations (like implication and equivalence) can be simulated using these "basic" operations.

In Subsections 7.2.1–7.2.2, we sketch the limits of our query and policy languages when using Boolean operations. On the one hand, we present several examples showing that confidentiality cannot be preserved under certain Boolean operations when adhering to the stateless CQE approach from Section 6.3. On the other hand, we identify the Boolean operations that are "compatible" with our approach. We formally show in Subsection 7.2.3 that stateless CQE still preserves confidentiality when we enhance query and policy languages by these Boolean operations.

This section is based on the work of Biskup, Lochner and Sonntag [BLS09] who originally investigated stateless CQE for queries and potential secrets with Boolean operations.

7.2.1 Limits of the Query Language

As illustrated in Example 7.4, the database user might utilize FDs in order to infer potential secrets. More precisely, he might exploit the implicative structure of an FD, which can be described informally as follows:

> Whenever two tuples agree on the left-hand side attributes of an FD, then they also agree on the right-hand side attributes.

Implicative structures, however, may not only arise from FDs but also from queries if Boolean combinations are allowed in the query language. Consider the following example.

Example 7.11. Recall the relation schema ACC_S and the instance acc from Example 5.1. ACC_S is in ONF, since the unique key is $\mathcal{K} = \{BANK, ACC\_NO\}$ and $lhs(\sigma) = \mathcal{K}$ for the sole FD $\sigma \equiv \{BANK, ACC\_NO\} \rightarrow \{ACC\_HOLDER, BALANCE\}$. Let a policy be given by $pot\_sec = \{\Psi\}$ with

$$\Psi \equiv (\exists X_{BAL})ACC(bankB, 101, anderson, X_{BAL}).$$

Assume that the database user may express arbitrary Boolean combinations of existential R-sentences as queries and consider the query sequence $Q = \langle \Phi_1, \Phi_2 \rangle$ with

$$\begin{aligned} \Phi_1 \equiv\ & (\exists X_{BAL})ACC(bankB, 101, anderson, X_{BAL}) \vee \\ & (\exists X_{BAL})ACC(bankB, 999, anderson, X_{BAL}) \quad \text{and} \\ \Phi_2 \equiv\ & (\exists X_{BAL})ACC(bankB, 999, anderson, X_{BAL}). \end{aligned}$$

Observe that $\mathit{eval}^*(\Phi_1)(\mathit{acc}) = \Phi_1$ and $\mathit{eval}^*(\Phi_2)(\mathit{acc}) = \neg\Phi_2$, and moreover $\Phi_1 \not\models \Psi$ and $\neg\Phi_2 \not\models \Psi$. Suppose that the user's queries are controlled using a suitably adapted version of the stateless CQE *scqe* from Definition 6.11; in this adaptation the censor does not only work on positive existential R-sentences but on arbitrary Boolean combinations of positive existential R-sentences. The adapted *scqe* then yields $\langle \mathit{ans}_1, \mathit{ans}_2\rangle$ with

$$\begin{aligned} \mathit{ans}_1 = \;& (\exists X_{BAL})ACC(\mathit{bankB},101,\mathit{anderson},X_{BAL}) \vee \\ & (\exists X_{BAL})ACC(\mathit{bankB},999,\mathit{anderson},X_{BAL}) \quad \text{and} \\ \mathit{ans}_2 = \;& \neg(\exists X_{BAL})ACC(\mathit{bankB},999,\mathit{anderson},X_{BAL}). \end{aligned}$$

However, $\{\mathit{ans}_1, \mathit{ans}_2\} \models (\exists X_{BAL})ACC(\mathit{bankB},101,\mathit{anderson},X_{BAL}) \equiv \Psi$ and thus the user is able to infer the secret by combining the answers to his queries. $\Diamond$

Example 7.11 demonstrates the exploitation of a "hidden implicative structure": Disjunction is equivalent to implication in the sense that $\chi_1 \vee \chi_2$ is equivalent to $\neg\chi_1 \Rightarrow \chi_2$ for first-order formulas χ_1 and χ_2. We therefore refrain from incorporating disjunction into the query language in the following. Moreover, we have to keep in mind that disjunction may be expressed by a combination of conjunction and negation: $\neg(\chi_1 \wedge \chi_2)$ is equivalent to $\neg\chi_1 \vee \neg\chi_2$ for first-order formulas χ_1 and χ_2. Thus we allow negation only in combination with (atomic) existential R-sentences and not with conjunctions of existential R-sentences.

Enhancing existential R-sentences with negation leads to the language $\mathscr{L}_{\exists R}^{\neg}$ which then can be used to define our enhanced query language $\mathscr{L}_{\exists R}^{\neg,\wedge}$.

Definition 7.12 ($\mathscr{L}_{\exists R}$ with negation). *The language $\mathscr{L}_{\exists R}^{\neg}$ is defined by*

$$\mathscr{L}_{\exists R}^{\neg} := \mathscr{L}_{\exists R} \cup \{\neg\chi \mid \chi \in \mathscr{L}_{\exists R}\}.$$

Definition 7.13 ($\mathscr{L}_{\exists R}$ with negation and conjunction). *The language $\mathscr{L}_{\exists R}^{\neg,\wedge}$ is inductively defined as follows:*

- *If* $\chi \in \mathscr{L}_{\exists R}^{\neg}$ *then* $\chi \in \mathscr{L}_{\exists R}^{\neg,\wedge}$;
- *if* $\chi_1, \chi_2 \in \mathscr{L}_{\exists R}^{\neg,\wedge}$ *then* $\chi_1 \wedge \chi_2 \in \mathscr{L}_{\exists R}^{\neg,\wedge}$.

In Example 7.11, we already pointed out that the stateless CQE *scqe*, as given by Definition 6.11, is, strictly speaking, not appropriate for queries with Boolean combinations, since it is defined for atomic (positive) existential R-sentences only. We therefore have to develop a modified version of *scqe*.

A straightforward approach to handling a query Φ from $\mathscr{L}_{\exists R}^{\neg,\wedge}$ would check whether or not Φ implies a potential secret Ψ; if $\Phi \not\models \Psi$ for every $\Psi \in \mathit{pot\_sec}$, the ordinary evaluation of Φ would be returned; otherwise, if $\Phi \models \Psi$ for a $\Psi \in \mathit{pot\_sec}$, the mechanism would return `mum` which has been introduced as special return value for refusals in Section 3.2.

This approach, however, is too restrictive in general because if one conjunct of Φ implies a secret, the whole query has to be refused: Suppose that $\Phi \equiv \Phi_1 \wedge \ldots \wedge \Phi_i \wedge \ldots \wedge \Phi_n$ and $\Phi_i \models \Psi$ for a potential secret Ψ; then, by the monotonicity of first-order logic, $\{\Phi_1, \ldots, \Phi_i, \ldots, \Phi_n\} \models \Psi$, and thus, by the semantics of logical implication, $\Phi_1 \wedge \ldots \wedge \Phi_i \wedge \ldots \wedge \Phi_n \models \Psi$. Consequently, the mechanism refuses to answer Φ and therefore returns `mum`, although the conjuncts $\Phi_1, \ldots, \Phi_{i-1}, \Phi_{i+1}, \ldots, \Phi_n$ possibly could have been answered separately without enabling the user to disclose a secret. Therefore, we propose a stateless CQE for queries from $\mathscr{L}_{\exists R}^{\neg,\wedge}$ which breaks up a query into its conjuncts and answers them separately. In other words, the query is modified by the mechanism in order to provide (possibly) more information to the user.

Observe that disjunction in queries could actually be handled similarly: In Example 7.11, the harmful inference arises from the first disjunct of query Φ_1 which is equivalent to the potential secret. The reason why the query is not refused is that the second disjunct weakens the query such that the potential secret is not implied only by this query. A stateless CQE mechanism for disjunctive queries, however, could inspect the first query and find that the first disjunct might disclose a secret; thus, the mechanism could refuse to answer the whole query but return an answer to the second disjunct instead. We refrain from proposing such a mechanism for the following reason: Splitting up a conjunctive query into the single conjuncts does not change the semantics of the query as we shall see in the proof of Theorem 7.21; for a disjunctive query $\Phi \equiv \Phi_1 \vee \ldots \vee \Phi_n$, however, it makes a difference whether we consider Φ or the sequence $\langle \Phi_1, \ldots, \Phi_n \rangle$, since $\{\Phi_1, \ldots, \Phi_n\} \models \Phi$ but $\Phi \not\models \{\Phi_1, \ldots, \Phi_n\}$, that is, the sequence is in some sense stronger than the disjunctive query.

Besides conjunction, we have to take account of negation that possibly occurs in a query from $\mathscr{L}_{\exists R}^{\neg,\wedge}$. The stateless CQE *scqe* is defined only for positive existential R-sentences, but it can easily be adapted for negative existential R-sentences: According to the remark to Definition 6.11, the censor of *scqe* only checks the (positive) query for implications, since the negative form of the query cannot imply a secret due to Lemma 6.10. Considering negative queries, however, the query itself might be negative and consequently the censor needs to check only the negated query which, in turn, is a positive existential R-sentence. We therefore assume a slightly modified censor for *scqe* in the following. It is given by

$$\mathit{censor}_{\exists R}^{\neg}(\mathit{pot\_sec}, \Phi) := (\text{exists } \Psi)[\Psi \in \mathit{pot\_sec} \text{ and } \Phi^{+} \models \Psi], \tag{7.19}$$

where Φ^{+} denotes the positive form of a query Φ.

With this deliberations in mind, we define the stateless CQE $\mathit{scqe}^{\neg,\wedge}$ which is an adaption of *scqe*, as given by Definition 6.11, for query sequences containing negation and/or conjunction.

Definition 7.14 (Stateless CQE for queries from $\mathscr{L}_{\exists \mathsf{R}}^{\neg,\wedge}$). *Consider*

- *a query sequence $Q = \langle \Phi_1, \Phi_2, ... \rangle$ with $\Phi_1, \Phi_2, ... \in \mathscr{L}_{\exists R}^{\neg,\wedge}$, that is,* $\Phi_i \equiv (\neg)\Phi_{i,1} \wedge (\neg)\Phi_{i,2} \wedge ... \wedge (\neg)\Phi_{i,m_i}$ *for all* $i \in \{1, 2, ...\}$ *such that $\Phi_{i,j} \in \mathscr{L}_{\exists R}$ for all $\Phi_{i,j}$,*
- *a policy $pot\_sec \subset \mathscr{L}_{\exists R}$ and*
- *an instance r.*

Then $scqe^{\neg,\wedge}$ is defined as follows:

$$
\begin{aligned}
&scqe^{\neg,\wedge}(Q)(r,pot\_sec) := scqe(Q')(r,pot\_sec) \quad \textit{with}\\
&Q' := \langle (\neg)\Phi_{1,1}, (\neg)\Phi_{1,2}, \dots, (\neg)\Phi_{1,m_1}, (\neg)\Phi_{2,1}, ... \rangle
\end{aligned}
$$

Example 7.15. Consider the schema $\langle R, ABC, A \rightarrow BC \rangle$ and the instance $r = \{R(a,b,c), R(d,b,e)\}$. A policy is given by $pot\_sec = \{\Psi_1, \Psi_2\}$ with

$$
\begin{aligned}
&\Psi_1 \equiv (\exists X_A)(\exists X_C)R(X_A,b,X_C) \quad \text{and}\\
&\Psi_2 \equiv (\exists X_B)R(a,X_B,e).
\end{aligned}
$$

Now the user sends the query sequence $Q = \langle \Phi_1, \Phi_2 \rangle$ to the database with $\Phi_1, \Phi_2 \in \mathscr{L}_{\exists R}^{\neg,\wedge}$ given by

$$
\begin{aligned}
&\Phi_1 \equiv R(a,b,e) \wedge (\exists X_A)R(X_A,f,e) \quad \text{and}\\
&\Phi_2 \equiv \neg(\exists X_B)R(d,X_B,e) \wedge (\exists X_B)(\exists X_C)R(a,X_B,X_C).
\end{aligned}
$$

For answering Q, according to Definition 7.14, Q is first transformed into $Q' = \langle \Phi_{1,1}, \Phi_{1,2}, \Phi_{2,1}, \Phi_{2,2} \rangle$ with

$$
\begin{aligned}
&\Phi_{1,1} \equiv R(a,b,e),\\
&\Phi_{1,2} \equiv (\exists X_A)R(X_A,f,e),\\
&\Phi_{2,1} \equiv \neg(\exists X_B)R(d,X_B,e) \quad \text{and}\\
&\Phi_{2,2} \equiv (\exists X_B)(\exists X_C)R(a,X_B,X_C).
\end{aligned}
$$

Second, Q is answered as follows:

$$
\begin{aligned}
scqe^{\neg,\wedge}(Q)(r,pot\_sec) &= scqe(Q')(r,pot\_sec)\\
&= \langle ans_{1,1}, ans_{1,2}, ans_{2,1}, ans_{2,2} \rangle
\end{aligned}
$$

with

$$\begin{aligned}
&\mathit{ans}_{1,1} = \texttt{mum}\\
&[\Phi_{1,1} \models \Psi_2 \text{ and thus } \mathit{censor}^{\neg}_{\exists R}(\mathit{pot\_sec},\Phi_{1,1}) = \texttt{true}]\\
&\mathit{ans}_{1,2} = \neg(\exists X_A)R(X_A,f,e)\\
&[\mathit{censor}^{\neg}_{\exists R}(\mathit{pot\_sec},\Phi_{1,2}) = \texttt{false} \text{ and } \mathit{eval}^*(\Phi_{1,2})(r) = \neg\Phi_{1,2}]\\
&\mathit{ans}_{2,1} = (\exists X_B)R(d,X_B,e)\\
&[\mathit{censor}^{\neg}_{\exists R}(\mathit{pot\_sec},\Phi_{2,1}) = \texttt{false} \text{ and } \mathit{eval}^*(\Phi_{2,1})(r) = \neg\Phi_{2,1}]\\
&\mathit{ans}_{2,2} = (\exists X_B)(\exists X_C)R(a,X_B,X_C)\\
&[\mathit{censor}^{\neg}_{\exists R}(\mathit{pot\_sec},\Phi_{2,2}) = \texttt{false} \text{ and } \mathit{eval}^*(\Phi_{2,2})(r) = \Phi_{2,2}]
\end{aligned}$$

The user may now assemble these answers to get the answers ans_1 and ans_2 for his original query sequence:

$$\begin{aligned}
&\mathit{ans}_1 = \mathit{ans}_{1,1} \wedge \mathit{ans}_{1,2} = \texttt{mum} \wedge \neg(\exists X_A)R(X_A,f,e)\\
&\mathit{ans}_2 = \mathit{ans}_{2,1} \wedge \mathit{ans}_{2,2} = (\exists X_B)R(d,X_B,e) \wedge (\exists X_B)(\exists X_C)R(a,X_B,X_C)
\end{aligned}$$

The special value `mum` indicates that the answer to $\Phi_{1,1}$ has been refused. Strictly speaking, `mum` is not a logical formula and thus ans_1 is not a logical formula either. In the context of queries from $\mathscr{L}^{\neg,\wedge}_{\exists R}$, however, the user may consider `mum` as a tautology (`mum` $\equiv \top$) in order to express the "logical neutrality" of a refused query. This leads to $\mathit{ans}_1 \equiv \top \wedge \neg(\exists X_A)R(X_A,f,e) \equiv \neg(\exists X_A)R(X_A,f,e)$. ◊

7.2.2 Limits of the Policy Language

Unlike the query language, the policy language may not be enhanced by negation and conjunction without enabling the user to disclose secrets. We illustrate this with the following two examples.

Example 7.16. Recall the relation schema ACC_S and the instance acc from Example 5.1. Suppose that negation is allowed in the policy and consider $\mathit{pot\_sec} = \{\Psi\}$ with

$$\Psi \equiv \neg \mathit{ACC}(\mathit{bankA},103,\mathit{brown},200).$$

A user who sends the query

$$\Phi \equiv \mathit{ACC}(\mathit{bankA},103,\mathit{smith},100)$$

to the database gets the answer $\mathit{ans} = \mathit{eval}^*(\Phi)(\mathit{acc}) = \mathit{ACC}(\mathit{bankA},103,\mathit{smith},100)$ from a stateless CQE, since $\Phi \not\models \Psi$. With the declared FD $\{\mathit{BANK},\mathit{ACC\_NO}\} \rightarrow$

$\{\mathit{ACC\_HOLDER}, \mathit{BALANCE}\}$, however, from this answer the user can conclude that $\mathit{ACC}(\mathit{bankA},103,\mathit{brown},200)$ must be false in acc, since $\{\mathit{BANK}, \mathit{ACC\_NO}\}$ is key and thus the instantiation $(\mathit{bankA},103)$ of the attributes $\mathit{BANK}, \mathit{ACC\_NO}$ uniquely determines the instantiation $(\mathit{smith},100)$ of the attributes $\mathit{ACC\_HOLDER}, \mathit{BALANCE}$. Therefore, the user knows that $r \models_{\mathsf{M}} \neg \mathit{ACC}(\mathit{bankA},103,\mathit{brown},200)$ and thus discloses a secret. ◊

Example 7.17. With the schema ACC_S and the instance acc from Example 5.1, let a conjunctive policy be given by $\mathit{pot\_sec} = \{\Psi\}$ with

$$\Psi \equiv \mathit{ACC}(\mathit{bankB},101,\mathit{anderson},1500) \wedge \mathit{ACC}(\mathit{bankB},105,\mathit{brown},1000).$$

The user queries

$$\Phi_1 \equiv \mathit{ACC}(\mathit{bankB},101,\mathit{anderson},1500) \quad \text{and}$$
$$\Phi_2 \equiv \mathit{ACC}(\mathit{bankB},105,\mathit{brown},1000)$$

may be answered correctly because

$$\Phi_1 \not\models \mathit{ACC}(\mathit{bankB},101,\mathit{anderson},1500) \wedge \mathit{ACC}(\mathit{bankB},105,\mathit{brown},1000) \quad \text{and}$$
$$\Phi_2 \not\models \mathit{ACC}(\mathit{bankB},101,\mathit{anderson},1500) \wedge \mathit{ACC}(\mathit{bankB},105,\mathit{brown},1000).$$

However, considering both answers,

$$\mathit{ans}_1 = \mathit{eval}^*(\Phi_1)(\mathit{acc}) = \mathit{ACC}(\mathit{bankB},101,\mathit{anderson},1500) \quad \text{and}$$
$$\mathit{ans}_2 = \mathit{eval}^*(\Phi_2)(\mathit{acc}) = \mathit{ACC}(\mathit{bankB},105,\mathit{brown},1000),$$

it holds that

$$\{\mathit{ans}_1, \mathit{ans}_2\} \models \mathit{ACC}(\mathit{bankB},101,\mathit{anderson},1500) \wedge \mathit{ACC}(\mathit{bankB},105,\mathit{brown},1000)$$

and thus the user is able to disclose the secret. ◊

Having shown exemplarily that negation and conjunction should not be allowed in the policy language in general, we now motivate why disjunction in the policy language is not harmful. For a query Φ and a disjunctive secret $\Psi_1 \vee \Psi_2$, the stateless CQE has to check whether or not $\Phi \models \Psi_1 \vee \Psi_2$. Since, following Example 7.11, we refrain from allowing disjunction in the query language, $\Phi \models \Psi_1 \vee \Psi_2$ holds if and only if $\Phi \models \Psi_1$ or $\Phi \models \Psi_2$ holds (if we allowed disjunction in the query language, the query could be given by $\Phi \equiv \Psi_1 \vee \Psi_2$ which would lead to $\Phi \not\models \Psi_1$ and $\Phi \not\models \Psi_2$).[14] Thus, the policy $\{\Psi_1 \vee \Psi_2\}$ is, in some sense, equivalent with the policy

[14]We will formally prove this claim in the context of Theorem 7.22 in subsection 7.2.3.

$\{\Psi_1, \Psi_2\} \subset \mathscr{L}_{\exists R}$ which can be handled by the (confidentiality-preserving) stateless CQE *scqe* as introduced in Definition 6.11.

We now enhance our policy language by disjunction and define a stateless CQE for this enhanced language. In Subsection 7.2.3, we will formally show that this stateless CQE is secure in the sense of Definition 5.4.

Definition 7.18 ($\mathscr{L}_{\exists R}$ with disjunction). *The language $\mathscr{L}_{\exists R}^{\vee}$ is inductively defined as follows:*

- *If $\chi \in \mathscr{L}_{\exists R}$ then $\chi \in \mathscr{L}_{\exists R}^{\vee}$;*
- *if $\chi_1,\chi_2 \in \mathscr{L}_{\exists R}^{\vee}$ then $\chi_1 \vee \chi_2 \in \mathscr{L}_{\exists R}^{\vee}$.*

Definition 7.19 (Stateless CQE for secrets from $\mathscr{L}_{\exists R}^{\vee}$). *The stateless CQE $scqe_{\vee}$ emerges from the stateless CQE scqe, as given by Definition 6.11, by replacing the policy language $\mathscr{L}_{\exists R}$ with $\mathscr{L}_{\exists R}^{\vee}$.*

As will be indicated in the proof of Theorem 7.22 in Subsection 7.2.3, $scqe_{\vee}$ may be implemented by splitting up each disjunctive secret $\Psi_1 \vee \Psi_2 \vee ... \vee \Psi_m$ into atomic secrets $\Psi_1, \Psi_2, ..., \Psi_m$. Consequently, on an operational level, $scqe_{\vee}$ can be reduced to *scqe* and is therefore efficiently implementable as well (see Section 6.4). In contrast to $scqe^{\neg,\wedge}$, we do not propose this reduction to *scqe* already in the definition of $scqe_{\vee}$, since on a declarative level it makes no difference whether or not disjunctive secrets are split.

7.2.3 Preservation of Confidentiality

In the previous subsection, we identified possible enhancements of the query and policy languages on an informal level. It turned out that, regarding the query language, conjunction and a restricted form of negation might be added and, regarding the policy language, only disjunction might be harmless.

In the following, we will formally show that these enhancements are in fact acceptable regarding preservation of confidentiality. We start with a lemma which basically says that each query sequence of negative existential R-sentences can be rewritten as a query sequence without negation that leads to the same answers under stateless CQE.

Lemma 7.20 (Negation in query sequences). *Let r be a database instance of the schema $R_S = \langle R, \mathcal{U}, \Sigma\rangle$, $pot\_sec \subset \mathscr{L}_{\exists R}$ a policy with $select(\Psi) \in bfs(R_S)$ for every $\Psi \in pot\_sec$, $Q = \langle(\neg)\Phi_1, (\neg)\Phi_2, ...\rangle$ a query sequence with possibly negated existential R-sentences $\Phi_1, \Phi_2, ... \in \mathscr{L}_{\exists R}$, and Q^+ the query sequence that emerges from Q by skipping negation, that is, $Q^+ = \langle \Phi_1, \Phi_2, ...\rangle$ with $\Phi_1, \Phi_2, ... \in \mathscr{L}_{\exists R}$. Then $scqe(Q^+)(r,pot\_sec) = scqe(Q)(r,pot\_sec)$ for scqe with the modified censor according to (7.19).*

Proof. Let the answer sets to Q^+ and Q be denoted by $\mathit{scqe}(Q^+)(r,\mathit{pot\_sec}) = \langle \mathit{ans}_1^+, \mathit{ans}_2^+, \ldots\rangle$ and $\mathit{scqe}(Q)(r,\mathit{pot\_sec}) = \langle \mathit{ans}_1, \mathit{ans}_2, \ldots\rangle$, respectively. We now show that $\mathit{ans}_i^+ = \mathit{ans}_i$ for every $i \in \{1, 2, \ldots\}$ by distinguishing two cases:

1. $\mathit{ans}_i = \mathtt{mum}$. According to Definition 6.11 (with the modified censor (7.19)), scqe yields the same censor decisions on Q and Q^+, since the censor only checks the positive form of a formula. Therefore, we get $\mathit{ans}_i^+ = \mathit{ans}_i = \mathtt{mum}$.

2. $\mathit{ans}_i \neq \mathtt{mum}$. In this case, $\mathit{ans}_i = \mathit{eval}^*((\neg)\Phi_i)(r)$ according to Definition 6.11. If the i-th query of Q is positive, then the same query occurs in Q^+ at the i-th position and it trivially holds that $\mathit{ans}_i^+ = \mathit{ans}_i$. Otherwise, if the i-th query of Q is negative, we get $\mathit{ans}_i = \mathit{eval}^*(\neg\Phi_i)(r)$. We distinguish two subcases:
   a) $\mathit{eval}^*(\neg\Phi_i)(r) = \Phi_i$. Then, by Definition 2.2, $r \not\models_{\mathsf{M}} \neg\Phi_i$ and thus $r \models_{\mathsf{M}} \Phi_i$. It follows that $\mathit{ans}_i^+ = \mathit{eval}^*(\Phi_i)(r) = \Phi_i$.
   b) $\mathit{eval}^*(\neg\Phi_i)(r) = \neg\Phi_i$. Then, by Definition 2.2, $r \models_{\mathsf{M}} \neg\Phi_i$ and thus $r \not\models_{\mathsf{M}} \Phi_i$. It follows that $\mathit{ans}_i^+ = \mathit{eval}^*(\Phi_i)(r) = \neg\Phi_i$.

Consequently, since $\mathit{scqe}(Q^+)(r,\mathit{pot\_sec})$ and $\mathit{scqe}(Q)(r,\mathit{pot\_sec})$ have been shown to return the same answers, the proof is complete. □

We make use of this result in the proof of the following theorem which declares that the stateless CQE from Definition 7.14 actually preserves confidentiality.

Theorem 7.21 ($\mathit{scqe}^{\neg,\wedge}$ preserves confidentiality). *Reconsider the assumptions of Theorem 7.10, but let the queries in Q be elements of $\mathscr{L}_{\exists R}^{\neg,\wedge}$. Then the stateless CQE $\mathit{scqe}^{\neg,\wedge}(Q)(r,\mathit{pot\_sec})$ is secure in the sense of Definition 5.4.*

Proof. For simplicity, we show the theorem for a query sequence consisting of a single query. However, the proof can be enhanced straightforwardly for query sequences of arbitrary length.

Let $Q = \langle\Phi\rangle$ with $\Phi \equiv (\neg)\Phi_1 \wedge (\neg)\Phi_2 \wedge \ldots \wedge (\neg)\Phi_m$ a query from $\mathscr{L}_{\exists R}^{\neg,\wedge}$. By Definition 7.14 and Lemma 7.20 it holds that

$$\begin{aligned} \mathit{scqe}^{\neg,\wedge}(\langle\Phi\rangle)(r,\mathit{pot\_sec}) &= \mathit{scqe}(\langle(\neg)\Phi_1, (\neg)\Phi_2, \ldots, (\neg)\Phi_m\rangle)(r,\mathit{pot\_sec}) \qquad (7.20)\\ &= \mathit{scqe}(\langle\Phi_1, \Phi_2, \ldots, \Phi_m\rangle)(r,\mathit{pot\_sec})\\ &= \langle \mathit{ans}_1, \mathit{ans}_2, \ldots, \mathit{ans}_m\rangle \end{aligned}$$

for scqe from Definition 6.11 but with the modified censor (7.19).

In the following, we determine an alternative instance in the sense of Definition 5.4, thereby proving $\mathit{scqe}^{\neg,\wedge}(\langle\Phi\rangle)(r,\mathit{pot\_sec})$ secure. By Theorem 7.10, we know that $\mathit{scqe}(\langle\Phi_1, \Phi_2, \ldots, \Phi_m\rangle)(r,\mathit{pot\_sec})$ is secure in the sense of Definition 5.4 and thus, by Lemma 5.5, for the user knowledge $\mathit{log} = \mathit{log}_0 \cup \{\mathit{ans}_i \mid i \in \{1, \ldots, m\}, \mathit{ans}_i \neq \mathtt{mum}\}$

and a fixed potential secret $\Psi \in \mathit{pot\_sec}$ it holds that $\mathit{log} \not\models \Psi$, that is, there exists an instance r' with

$$r' \models_{\mathsf{M}} \mathit{log} \text{ and } r' \not\models_{\mathsf{M}} \Psi. \tag{7.21}$$

By (7.20), $\mathit{scqe}^{\neg,\wedge}(\langle\Phi\rangle)(r,\mathit{pot\_sec})$ and $\mathit{scqe}(\langle\Phi_1,\Phi_2,\dots,\Phi_m\rangle)(r,\mathit{pot\_sec})$ yield the same answers and thus the same user log. Therefore, r' satisfies the properties (7.21) also for $\mathit{scqe}^{\neg,\wedge}(\langle\Phi\rangle)(r,\mathit{pot\_sec})$, making it a suitable alternative instance in the sense of Definition 5.4 for $\mathit{scqe}^{\neg,\wedge}(\langle\Phi\rangle)(r,\mathit{pot\_sec})$. □

It remains to show that also the stateless CQE for the enhanced policy language according to Definition 7.19 preserves confidentiality. We express this result in the following theorem.

Theorem 7.22 ($\mathit{scqe}_\vee$ preserves confidentiality). *Reconsider the assumptions of Theorem 7.10, but let the potential secrets in* $\mathit{pot\_sec}$ *be elements of* $\mathscr{L}^{\vee}_{\exists R}$ *such that* $\mathit{select}(\Psi_{i,j}) \in \mathit{bfs}(R_S)$ *for every* $\Psi_{i,j}$ *occurring in* $\Psi_i \equiv \Psi_{i,1}\vee\Psi_{i,2}\vee\ldots\vee\Psi_{i,m_1} \in \mathit{pot\_sec}$. *Then the stateless CQE* $\mathit{scqe}_\vee(Q)(r,\mathit{pot\_sec})$ *is secure in the sense of Definition 5.4.*

Proof. For simplicity, we show the theorem for a policy consisting of a single potential secret. However, the proof can be enhanced straightforwardly for arbitrary policies.

We consider the policy $\mathit{pot\_sec} = \{\Psi\}$ with $\Psi \equiv \Psi_1 \vee \Psi_2 \vee \ldots \vee \Psi_m$ and show that $\mathit{scqe}_\vee$ yields the same results on $\mathit{pot\_sec}$ as scqe on $\mathit{pot\_sec}' = \{\Psi_1,\Psi_2,\dots,\Psi_m\}$. Thus, $\mathit{scqe}_\vee$ can be reduced to scqe by splitting up the policy into atomic, that is, non-disjunctive, secrets. Since, according to Theorem 7.10, scqe is secure in the sense of Definition 5.4, it then immediately follows that $\mathit{scqe}_\vee$ is secure as well.

More precisely, we show that for a query $\Phi \in \mathscr{L}_{\exists R}$ it holds that

$$\begin{aligned} &\Phi \models \Psi_1 \vee \Psi_2 \vee \ldots \vee \Psi_m \quad \text{if and only if} \\ &\Phi \models \Psi_1 \text{ or } \Phi \models \Psi_2 \text{ or } \ldots \text{ or } \Phi \models \Psi_m, \end{aligned} \tag{7.22}$$

which yields the same censor decisions and thus the same answers under $\mathit{scqe}_\vee$ and scqe.

First, we show the "$\Rightarrow$"-part of (7.22): Assume that $\Phi \models \Psi_1 \vee \Psi_2 \vee \ldots \vee \Psi_m$, which is, by the definition of logical implication, equivalent to

$$\text{for all instances } r: \text{ if } r \models_{\mathsf{M}} \Phi \text{ then } r \models_{\mathsf{M}} \Psi_1 \vee \Psi_2 \vee \ldots \vee \Psi_m. \tag{7.23}$$

With the semantics of the $\vee$-operator it follows

$$\text{for all instances } r: \text{ if } r \models_{\mathsf{M}} \Phi \text{ then } r \models_{\mathsf{M}} \Psi_1 \text{ or } \ldots \text{ or } r \models_{\mathsf{M}} \Psi_m. \tag{7.24}$$

Now consider a ground atom $\chi_g \in \mathscr{L}_R$ with the following properties:

- For all $A \in \mathit{select}(\Phi)$: $\chi_g[A] = \Phi[A]$;
- for all $A \notin \mathit{select}(\Phi)$: $\chi_g[A] \in \mathit{Const}'$ with Const' denoting a set of constants that do not occur in $\Phi, \Psi_1, \ldots, \Psi_m$.[15]

By Definition 6.4, Φ is relevant for χ_g and thus, by Lemma 6.5,

$$\chi_g \models \Phi. \tag{7.25}$$

We then consider an instance $r' = \{\chi_g\}$, that is,

$$r' \models_{\mathsf{M}} \chi_g \tag{7.26}$$

and $r' \not\models_{\mathsf{M}} \chi$ for every ground atom $\chi \in \mathscr{L}_R$ with $\chi \not\equiv \chi_g$. From (7.25) and (7.26), we conclude with Lemma 6.7 that $r' \models_{\mathsf{M}} \Phi$ and thus, by (7.24), $r' \models_{\mathsf{M}} \Psi_1$ or ... or $r' \models_{\mathsf{M}} \Psi_m$. We assume that $r' \models_{\mathsf{M}} \Psi_i$ for a fixed $\Psi_i \in \{\Psi_1, \Psi_2, \ldots, \Psi_m\}$ and have to show that $\Phi \models \Psi_i$ holds in order to complete the proof of the "$\Rightarrow$"-part of (7.22). By Lemma 6.7, there exists a ground atom $\chi'_g \in \mathscr{L}_R$ with

$$r' \models_{\mathsf{M}} \chi'_g \text{ and } \chi'_g \models \Psi_i. \tag{7.27}$$

Since, by construction, χ_g is the only ground atom being true in r', we know that $\chi'_g \equiv \chi_g$. Consequently, with (7.27) it holds that

$$\chi_g \models \Psi_i, \tag{7.28}$$

meaning that Ψ_i is relevant for χ_g according to Lemma 6.5. From Definition 6.4 it then follows that

$$\text{for all } A \in \mathit{select}(\Psi_i) : \chi_g[A] = \Psi_i[A]. \tag{7.29}$$

By definition of Const', $\Psi_i[A] \in \mathit{Const} \backslash \mathit{Const}'$ for all attributes $A \in \mathit{select}(\Psi_i)$ and thus, according to (7.29), $\chi_g[A] \in \mathit{Const} \backslash \mathit{Const}'$ for all attributes $A \in \mathit{select}(\Psi_i)$. Then, by construction of χ_g, it holds that

$$\text{for all } A \in \mathit{select}(\Psi_i) : \Phi[A] = \Psi_i[A]. \tag{7.30}$$

Therefore, by Definition 6.4, Ψ_i is relevant for Φ and, by Lemma 6.5, $\Phi \models \Psi_i$.

Second, we show the "$\Leftarrow$"-part of (7.22): Assume that $\Phi \models \Psi_i$ for a fixed $\Psi_i \in \{\Psi_1, \Psi_2, \ldots, \Psi_m\}$ and consider an instance r with

$$r \models_{\mathsf{M}} \Phi. \tag{7.31}$$

[15]Such a non-empty set Const' always exists, since we assume an infinite set Const of constants and $\Phi, \Psi_1, \Psi_2, \ldots, \Psi_m$ contain only finitely many constants.

Then, by the definition of logical implication, it follows that $r \models_{\mathsf{M}} \Psi_i$ and thus also

$$r \models_{\mathsf{M}} \Psi_1 \vee \ldots \vee \Psi_i \vee \ldots \vee \Psi_m. \tag{7.32}$$

From (7.31), (7.32) and the definition of logical implication, we conclude that $\Phi \models \Psi_1 \vee \Psi_2 \ldots \vee \Psi_m$. □

In the proof of Theorem 7.21, queries have been split up into the single conjuncts in order to reduce $\mathit{scqe}^{\neg,\wedge}$ to scqe; moreover, in the proof of Theorem 7.22, policy elements have been split up into the single disjuncts to reduce $\mathit{scqe}_{\vee}$ to scqe. These two approaches can easily be combined to allow queries from $\mathscr{L}_{\exists R}^{\neg,\wedge}$ and potential secrets from $\mathscr{L}_{\exists R}^{\vee}$ at the same time.

The following definition captures this insight and Corollary 7.24, which follows from Theorems 7.21 and 7.22, states that the combined mechanism preserves confidentiality.

Definition 7.23 (Stateless CQE for Boolean expressions). *Consider*

- *a query sequence* $Q = \langle \Phi_1, \Phi_2, \ldots \rangle$ *with* $\Phi_1, \Phi_2, \ldots \in \mathscr{L}_{\exists R}^{\neg,\wedge}$*, that is,* $\Phi_i \equiv (\neg)\Phi_{i,1} \wedge (\neg)\Phi_{i,2} \wedge \ldots \wedge (\neg)\Phi_{i,m_i}$ *for all* $i \in \{1, 2, \ldots\}$ *such that* $\Phi_{i,j} \in \mathscr{L}_{\exists R}$ *for all* $\Phi_{i,j}$*,*
- *a policy* $\mathit{pot\_sec} \subset \mathscr{L}_{\exists R}^{\vee}$ *and*
- *an instance* r*.*

Then the stateless CQE $\mathit{scqe}_{\vee}^{\neg,\wedge}$ *is defined as follows:*

$$\mathit{scqe}_{\vee}^{\neg,\wedge}(Q)(r,\mathit{pot\_sec}) := \mathit{scqe}_{\vee}(Q')(r,\mathit{pot\_sec}) \quad \textit{with}$$
$$Q' := \langle (\neg)\Phi_{1,1}, (\neg)\Phi_{1,2}, \ldots, (\neg)\Phi_{1,m_1}, (\neg)\Phi_{2,1}, \ldots \rangle$$

Corollary 7.24 ($\mathit{scqe}_{\vee}^{\neg,\wedge}$ preserves confidentiality). *Reconsider the assumptions of Theorem 7.10, but let the queries in* Q *be elements of* $\mathscr{L}_{\exists R}^{\neg,\wedge}$ *and the potential secrets in* $\mathit{pot\_sec}$ *elements of* $\mathscr{L}_{\exists R}^{\vee}$ *such that* $\mathsf{select}(\Psi_{i,j}) \in \mathit{bfs}(R_S)$ *for every* $\Psi_{i,j}$ *occurring in* $\Psi_i \equiv \Psi_{i,1} \vee \Psi_{i,2} \vee \ldots \vee \Psi_{i,m_1} \in \mathit{pot\_sec}$*. Then the stateless CQE* $\mathit{scqe}_{\vee}^{\neg,\wedge}(Q)(r,\mathit{pot\_sec})$ *is secure in the sense of Definition 5.4.*

In Figure 7.3 on the next page, the schematic architecture of the combined mechanism is depicted. The lower right part captures the core of stateless CQE (scqe) as already shown in Figure 5.1. The enhancement $\mathit{scqe}^{\neg,\wedge}$ to queries from the language $\mathscr{L}_{\exists R}^{\neg,\wedge}$ and the enhancement $\mathit{scqe}_{\vee}$ to policies from the language $\mathscr{L}_{\exists R}^{\vee}$ is depicted in the upper right and in the lower left part, respectively, as a kind of preprocessing. Both enhancements together form the stateless CQE for Boolean expressions, $\mathit{scqe}_{\vee}^{\neg,\wedge}$. Queries as well as policies have to be split up suitably before they are forwarded to the core. More precisely, each query that contains conjunction

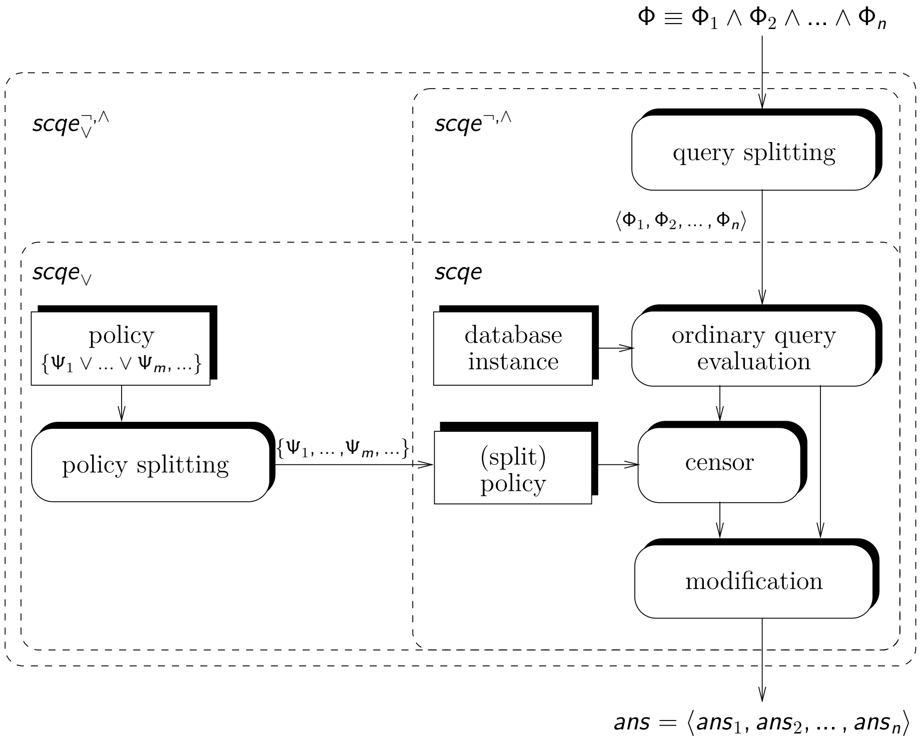
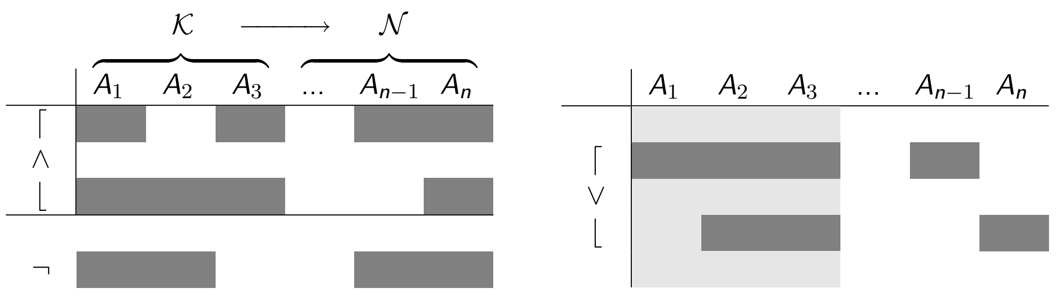

$\Phi \equiv \Phi_1 \wedge \Phi_2 \wedge \ldots \wedge \Phi_n$
$scqe_{\vee}^{\neg,\wedge}$
$scqe^{\neg,\wedge}$
query splitting
$\langle \Phi_1, \Phi_2, \ldots, \Phi_n \rangle$
$scqe_{\vee}$
$scqe$
policy
$\{\Psi_1 \vee \ldots \vee \Psi_m, \ldots\}$
database instance
ordinary query evaluation
policy splitting
$\{\Psi_1, \ldots, \Psi_m, \ldots\}$
(split) policy
censor
modification
$ans = \langle ans_1, ans_2, \ldots, ans_n \rangle$

Figure 7.3: Schematic architecture of $scqe_{\vee}^{\neg,\wedge}$.



$\mathcal{K} \longrightarrow \mathcal{N}$
A_1 A_2 A_3 ... A_{n-1} A_n
$\wedge$
$\neg$
A_1 A_2 A_3 ... A_{n-1} A_n
$\vee$

(a) schema and query restrictions

(b) policy restrictions

Figure 7.4: Visualization of the restrictions for Boolean expressions.

is considered as a set of atomic queries, and each potential secret that contains disjunction is considered as a a set of atomic potential secrets. Consequently, the actual controlled evaluation of queries can still be performed by *scqe*.

In Figure 7.4, the restrictions for confidentiality-preserving query answering under Boolean expressions in query and policy languages are visualized. Basically, the same elements as in Figure 7.2 are depicted, that is, ONF as schema restriction, existential R-sentences as queries and as potential secrets, and the "basic facts restriction" for potential secrets. Moreover, Figure 7.4 indicates which types of Boolean operations are allowed according to the investigations of this section: Figure 7.4(a) shows that queries may contain conjunction and negation (but neither negated conjunction nor disjunction), and Figure 7.4(b) indicates that potential secrets may contain disjunction (but neither negation nor conjunction).

7.3 Open Queries

With the query languages investigated so far, the database user is only able to express closed queries, that is, queries that are either true or false in the database instance under consideration. Most practical queries, however, are open queries, as already motivated in the introduction of Chapter 7. In this section, we enhance our query language in order to enable the database user to express open queries.

In Subsection 7.3.1, we introduce our enhanced query language and a suitable stateless CQE mechanism for answering open queries in a controlled way. As we shall see in Subsection 7.3.2, however, we will run into problems when still assuming that the database user is aware of the declared confidentiality policy (as demanded in Section 5.1). In Subsection 7.3.3, we therefore drop the assumption that the user is aware of the policy and show that it is then possible to preserve confidentiality again.

Stateless CQE for open queries has originally been investigated in the work of Biskup, Hartmann, Link and Lochner [BHLL10b] which has been taken as a basis for this section.

7.3.1 Query Language and Completeness Information

We take the language $\mathscr{L}_{\exists R}$ of existential R-sentences as a basis for our open query language. In Section 7.5, we will investigate how the resulting language can be enhanced by Boolean operations.

Open queries are simply existential R-sentences with free variables, as formally described by the following Definition.

Definition 7.25 (Open existential R-sentences). *The language $\mathscr{L}_{\exists R,fv}$ is defined by*

$$\begin{aligned}\mathscr{L}_{\exists R,fv} := \{\ & (\exists X_1)\dots(\exists X_m)R(v_1,\dots,v_n) \mid 0 \leq m \leq n,\\ & X_i \in Var,\\ & v_i \in Var \cup Const,\\ & \{X_1,\dots,X_m\} \subseteq \{v_1,\dots,v_n\},\\ & v_i, v_j \in Var \Rightarrow v_i \neq v_j\ \}.\end{aligned}$$

Of course, we also have to adapt the stateless CQE mechanism to open queries. As already indicated by the ordinary query evaluation (Definitions 2.2 and 2.3), for open queries we need an essentially different approach than for closed queries. This is due to the fact that closed queries can be answered by `true` or `false`, whereas for open queries the answers are sets of variable assignments making the query true in the considered database instance.

Regarding the refusal approach of CQE, the mechanisms introduced so far either give an honest answer to a query or refuse to answer a query at all (except for $scqe^{\neg,\wedge}$, as introduced in Definition 7.14, which considers a conjunctive query as a sequence of atomic sentences and answers each conjunct separately). We do not consider this approach appropriate for open queries but suggest to inspect the ordinary evaluation result and discard the harmful variable assignments. Observe that a variable assignment which is not part of the answer set to an open query may then be interpreted in two different ways:

- It may make the query false in the considered database instance.
- It may be harmful, that is, it possibly leads to the disclosure of a potential secret.

Because the suggested mechanism first determines the honest and complete answer to an open query and then “cleans up” the result set, we rather use the term *filtering* than refusal in the following. We now provide a formal definition for this mechanism.

Definition 7.26 (Stateless CQE for queries from $\mathscr{L}_{\exists \mathsf{R,fv}}$). *The stateless CQE scqe^{oq} is a specific modified query evaluation according to Definition 5.2. Given a policy $\mathit{pot\_sec} \subset \mathscr{L}_{\exists R}$, each of the answers $\mathit{ans}_1, \mathit{ans}_2, \ldots$ to a query sequence $Q = \langle \Phi_1, \Phi_2, \ldots \rangle$ with $\Phi_1, \Phi_2, \ldots \in \mathscr{L}_{\exists R,fv}$ is defined subject to the $\mathit{refused}$-set*

$$\mathit{refused}(\Phi_i(\vec{V_i}),\mathit{pot\_sec}) := \{\ \Phi(\vec{c_i}) \mid \vec{c_i} \in \mathit{Const} \times \ldots \times \mathit{Const} \textit{ and } \textit{exists } \Psi \in \mathit{pot\_sec} : \Phi(\vec{c_i}) \models \Psi\ \}.$$

More precisely, the stateless CQE scqe^{oq} is defined as follows:

$$\mathit{scqe}^{oq}(Q)(r,\mathit{pot\_sec}) := \langle \mathit{ans}_1, \mathit{ans}_2, \ldots \rangle \quad \textit{with}$$
$$\mathit{ans}_i := \mathit{eval}^*(\Phi_i(\vec{V_i}))(r) \setminus \mathit{refused}(\Phi_i(\vec{V_i}),\mathit{pot\_sec})$$

Observe that, regarding open queries, the database user might be aware of a "completeness information". In the case of ordinary query evaluation, this completeness information reads as follows:

> Each assignment of the free variables in a query Φ that does not make Φ true in the actual database instance makes Φ false in this instance.

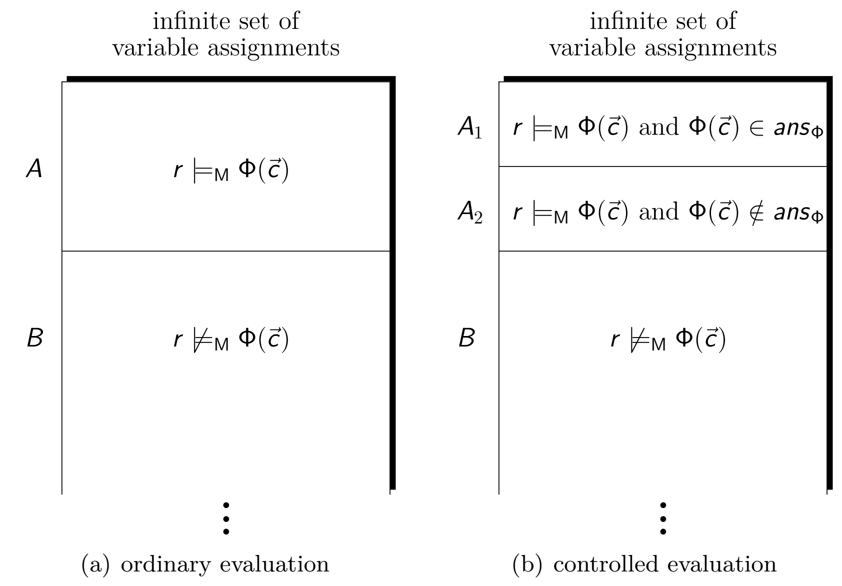
This information is valid, since, according to Section 5.1, we employ a closed world assumption. In the case of CQE, however, the completeness information reads as follows:

> Each assignment of the free variables in a query Φ that is not part of the (controlled) answer to Φ makes Φ false in the database instance or leads to the disclosure of a potential secret.

This description of the completeness information seems to be the natural one, but for the following investigations we need an alternative characterization. Note, however, that this characterization is only a syntactic alternative but semantically equivalent to the former one:

> Each assignment of the free variables in a query Φ either makes Φ false in the database instance, or it makes Φ true and is part of the controlled answer, or it makes Φ true and is not part of the controlled answer.

In Figure 7.5 on the following page, the completeness information regarding an open query $\Phi(\vec{V})$ is visualized as a partition of the (infinite) set of possible variable assignments of $\vec{V}$: In Figure 7.5(a), this set is divided into the finite set A of assignments that make $\Phi(\vec{V})$ true in the database instance and the infinite set B of assignments that make $\Phi(\vec{V})$ false. In Figure 7.5(b), A is further divided into the assignments that make $\Phi(\vec{V})$ true and that are returned as part of the controlled



infinite set of variable assignments
A
$r \models_M \Phi(\vec{c})$
B
$r \not\models_M \Phi(\vec{c})$
(a) ordinary evaluation
infinite set of variable assignments
A_1
$r \models_M \Phi(\vec{c})$ and $\Phi(\vec{c}) \in ans_\Phi$
A_2
$r \models_M \Phi(\vec{c})$ and $\Phi(\vec{c}) \notin ans_\Phi$
B
$r \not\models_M \Phi(\vec{c})$
(b) controlled evaluation

Figure 7.5: Partition of variable assignments for open queries.

answer (A_1) and the assignments that make $\Phi(\vec{V})$ true, but that are not returned because they are harmful (A_2).

According to Figure 7.5(b), we now express the completeness information by the following *completeness sentence*; it is dependent of the query Φ and the positive part of the controlled answer according to Definition 7.26, hereafter denoted by ans_Φ:[16]

$$complete(\Phi(\vec{V}), ans_\Phi) :\equiv \tag{7.33}$$

$$(\forall \vec{V}) \left[\underbrace{\neg\Phi(\vec{V})}_{B} \vee \underbrace{\left(\Phi(\vec{V}) \wedge \bigvee_{\Phi(\vec{c})\in ans_\Phi} \vec{V} = \vec{c}\right)}_{A_1} \vee \underbrace{\left(\Phi(\vec{V}) \wedge \bigwedge_{\Phi(\vec{c})\in ans_\Phi} \vec{V} \neq \vec{c}\right)}_{A_2} \right]$$

In the following, we use the abbreviations A_1, A_2 and B (which correspond to the partitions of the set of variable assignments in Figure 7.5) for the disjuncts of the completeness sentence when appropriate.

Note that the completeness sentence is actually a tautology, since

$$\bigwedge_{\Phi(\vec{c})\in ans_\Phi} \vec{V} \neq \vec{c} \quad \equiv \quad \neg \bigvee_{\Phi(\vec{c})\in ans_\Phi} \vec{V} = \vec{c},$$

[16]We slightly abuse notation, since universal quantification and equality are meant component-wise in this sentence.

and thus (7.33) reduces to $(\forall \vec{V})[\neg\Phi(\vec{V}) \vee \Phi(\vec{V})]$. Due to the universal validity we do not add this completeness sentence to the user log explicitly. However, we keep in mind that the user is aware of this information; as we shall see in the following subsection, it may assist him in disclosing secrets.

7.3.2 Problems with Known Policies

So far, we assumed that the database user who sends queries to the database is aware of the confidentiality policy which has been defined by the security administrator. The following example shows that this assumption should not be kept up in the context of open queries.

Example 7.27. Reconsider the Schema ACC_S and the instance acc from Example 5.1. Moreover, let a policy $\mathit{pot\_sec} = \{\Psi\}$ be given by

$$\Psi \equiv (\exists X_{BANK})(\exists X_{ACC\_NO})\mathit{ACC}(X_{BANK}, X_{ACC\_NO}, \mathit{brown}, 1000).$$

The queries $\Phi_1, \Phi_2 \in \mathscr{L}_{\exists R,fv}$ are given by

$$\Phi_1(X_{ACC\_NO}, X_{ACC\_H}, X_{BAL}) \equiv \mathit{ACC}(\mathit{bankB}, X_{ACC\_NO}, X_{ACC\_H}, X_{BAL}) \quad \text{and}$$
$$\Phi_2 \equiv (\exists X_{BAL})\mathit{ACC}(\mathit{bankB}, 105, \mathit{brown}, X_{BAL}).$$

Observe that Φ_1 is an open query that asks for all account information (account number, account holder and balance) of bank B in the instance acc, and Φ_2 is a closed query that asks if client Brown has, according to acc, an account at bank B with the account number 105.

As per Definition 7.26, the answers to Φ_1 and Φ_2 are determined by means of the respective eval^*- and $\mathit{refused}$-sets:

$$\begin{aligned} \mathit{eval}^*(\Phi_1)(\mathit{acc}) = \{\ & \mathit{ACC}(\mathit{bankB}, 101, \mathit{anderson}, 1500), \\ & \mathit{ACC}(\mathit{bankB}, 105, \mathit{brown}, 1000)\ \} \end{aligned}$$
$$\mathit{refused}(\Phi_1, \mathit{pot\_sec}) = \{\mathit{ACC}(\mathit{bankB}, 105, \mathit{brown}, 1000)\}$$

$$\mathit{eval}^*(\Phi_2)(\mathit{acc}) = \{(\exists X_{BAL})\mathit{ACC}(\mathit{bankB}, 105, \mathit{brown}, X_{BAL})\}$$
$$\mathit{refused}(\Phi_2, \mathit{pot\_sec}) = \emptyset$$

Consequently, we get $\mathit{scqe}^{oq}(\langle\Phi_1, \Phi_2\rangle)(\mathit{acc}, \mathit{pot\_sec}) = \langle \mathit{ans}_1, \mathit{ans}_2\rangle$ with

$$\mathit{ans}_1 = \{\mathit{ACC}(\mathit{bankB}, 101, \mathit{anderson}, 1500)\} \quad \text{and}$$
$$\mathit{ans}_2 = \{(\exists X_{BAL})\mathit{ACC}(\mathit{bankB}, 105, \mathit{brown}, X_{BAL})\}.$$

From ans_2 the user knows (by the semantics of the existential quantifier) that $\mathit{acc} \models_M \mathit{ACC}(\mathit{bankB}, 105, \mathit{brown}, c)$ for a suitable constant $c \in \mathit{Const}$. Note that, regarding query Φ_1, the variable assignment vector $(105,\mathit{brown},c)$ belongs to part A_2 of Figure 7.5(b), since it makes Φ_1 true in acc but has not been returned as part of the answer to Φ_1; thus, it discloses a secret. Since $\Psi \equiv (\exists X_{BANK})(\exists X_{ACC\_NO})ACC(X_{BANK},X_{ACC\_NO},\mathit{brown},1000)$ is the only declared secret, it must hold that

$$\begin{aligned}&ACC(\mathit{bankB},105,\mathit{brown},c) \models\\&(\exists X_{BANK})(\exists X_{ACC\_NO})ACC(X_{BANK},X_{ACC\_NO},\mathit{brown},1000),\end{aligned}$$

which means, according to Definition 6.4 and Lemma 6.5, that $c = 1000$. Summing up, the user knows that

$$\begin{aligned}&\mathit{acc} \models_{\mathrm{M}} ACC(\mathit{bankB},105,\mathit{brown},1000) \quad \text{and}\\&ACC(\mathit{bankB},105,\mathit{brown},1000) \models \Psi,\end{aligned}$$

and as a result, by the definition of $\models$, he also knows that $\mathit{acc} \models_{\mathrm{M}} \Psi$, that is, he discloses that the potential secret Ψ is true in the database instance acc. ◊

A crucial point in this example is the user awareness concerning the policy. The potential secret Ψ can only be inferred because the user knows the policy. In the following, we justify this claim and start with an informal description of the information the user gets from being aware of the confidentiality policy:

> If an assignment $\vec{c}$ of the free variables in an open query $\Phi(\vec{V})$ makes $\Phi(\vec{V})$ true in the database instance and does not occur in the controlled answer to $\Phi(\vec{V})$, then there exists a potential secret that is relevant for $\Phi(\vec{c})$.

We express this description more formally in the following *awareness sentence*, which is dependent of the query Φ, the policy $\mathit{pot\_sec} = \{\Psi_1, \Psi_2, \ldots, \Psi_m\}$ and the positive part of the controlled answer ans_Φ (which is not given as parameter of the awareness sentence explicitly, since it can be computed from the query and the policy):

$$\mathit{aware}(\Phi(\vec{V}),\mathit{pot\_sec}) :\equiv \tag{7.34}$$
$$(\forall \vec{V}) \left[\underbrace{\left(\Phi(\vec{V}) \wedge \bigwedge_{\Phi(\vec{c}) \in \mathit{ans}_\Phi} \vec{V} \neq \vec{c} \right)}_{A_2} \Rightarrow \bigvee_{i=1}^{m} \left(\bigwedge_{A \in \mathit{select}(\Psi_i)} \Phi(\vec{V})[A] = \Psi_i[A] \right) \right]$$

Note that the left-hand side of the implication corresponds to the A_2-disjunct of the completeness sentence (7.33).

When modeling the assumed user knowledge in the context of open queries, the awareness sentence (7.34) must clearly be taken into account. Considering an open query $\Phi_i(\vec{V_i})$ and the user knowledge log_{i-1} before answering $\Phi_i(\vec{V_i})$, the user knowledge log_i after having answered the query is determined as follows:

$$log_i = log_{i-1} \cup ans_i \cup complete(\Phi_i(\vec{V_i}), ans_i) \cup aware(\Phi_i(\vec{V_i}), pot\_sec) \qquad (7.35)$$

The disclosure of a potential secret, as exemplarily sketched in Example 7.27, exploits this user knowledge by suitably combining the completeness sentence (7.33) and the awareness sentence (7.34). Note that the completeness sentence can be rearranged to $(\forall \vec{V})[(\neg B \wedge \neg A_1) \Rightarrow A_2]$, that is, a user who knows that B and A_1 do not hold for a variable assignment $\vec{c}$ can infer that A_2 holds for $\vec{c}$. From the awareness sentence it then follows that

$$\bigvee_{i=1}^{m} \left(\bigwedge_{A \in select(\Psi_i)} \Phi(\vec{V})[A] = \Psi_i[A] \right),$$

that is, a $\Psi_i \in pot\_sec$ is relevant for $\Phi(\vec{c})$, leading to $\Phi(\vec{c}) \models \Psi_i$ (by Definition 6.4 and Lemma 6.5). We supposed that $B \equiv \neg\Phi(\vec{V})$ does not hold for $\vec{V} = \vec{c}$, thus $r \models_M \Phi(\vec{c})$ and consequently, by the definition of logical implication, $r \models_M \Psi_i$. If there is a unique $\Psi_i \in pot\_sec$ such that Ψ_i is relevant for $\Phi(\vec{c})$, then the user knows that this particular potential secret is true in the database instance and thus confidentiality is violated.

We point out that *not* the combination of an open query and a closed query is responsible for the sketched disclosure of secrets; reconsidering Example 7.27, the following open queries would lead to the same result sets as Φ_1 and Φ_2 and thus they would cause the same problem:

$$\Phi'_1(X_{ACC\_NO}, X_{ACC\_H}, X_{BAL}) \equiv ACC(bankB, X_{ACC\_NO}, X_{ACC\_H}, X_{BAL}) \quad \text{and}$$
$$\Phi'_2(X_{ACC\_NO}) \equiv (\exists X_{BAL}) ACC(bankB, X_{ACC\_NO}, brown, X_{BAL}).$$

It is rather the combination of an open query and an existential R-sentence that makes it possible to disclose a secret: On the one hand, the controlled evaluation of the open query suggests that there is no information in the database regarding a certain aspect—in Example 7.27, ans_1 suggests that there is no information about an account holder Brown with an account at bank B in the database; on the other hand, the controlled evaluation of the existential R-sentence shows that the uncontrolled answer of the open query would have contained information regarding this aspect—in Example 7.27, ans_2 shows that there *is* an account holder Brown with an account at bank B. Consequently, the user knows that a secret has been filtered from the answer set of the open query.

Preventing the database user from expressing queries with existentially quantified variables would possibly solve the sketched problem. However, we refrain from investigating this approach because further language restrictions are counterproductive when aiming at developing an expressive query language. In the following subsection, we therefore propose an alternative solution to the confidentiality problem in the context of open queries.

7.3.3 Preservation of Confidentiality

As justified in the preceding subsection, there is no straightforward transition from stateless CQE for closed queries to stateless CQE for open queries when aiming at guaranteeing preservation of confidentiality. Therefore, we have to suitably modify our approach, preferably without further restricting our query and policy languages.

Since the user awareness regarding the policy turned out to be a crucial point in Example 7.27, we claim that hiding the confidentiality policy from the user solves the problem. This is justified as follows: From (7.35) we know that, after having answered a query, the new user log consists of the former user log, the answer to the current query, the completeness sentence, and the awareness sentence. In the preceding subsection, we showed that the completeness sentence in combination with the awareness sentence may cause the disclosure of a potential secret. The completeness sentence is a tautology, so we cannot prevent the user from exploiting it for drawing inferences. The awareness sentence, however, is only known to the user if the policy is known, since the disjunction $\bigvee_{i=1}^{m}(\ldots)$ ranges over the declared potential secrets. Consequently, when hiding the policy, the awareness sentence can no longer be exploited by the user. Thus, when arguing about open queries we hereafter assume the policy to be unknown to the user.

It still remains to be shown formally that the answers to open queries do not enable the user to disclose a potential secret, that is, that scqe^{oq} is secure in the sense of Definition 5.4. We will show this in Theorem 7.35, but previously we introduce the chase of existential R-sentences—a technique that follows the chase algorithm of relational database theory as described in, for example, [AHV95]. Basically, our chase mechanism takes a set of existential R-sentences and a set of functional dependencies as input and generates a database instance as output that satisfies the functional dependencies and makes the existential R-sentences true. We will need this algorithm in the proof of Theorem 7.35 to construct an alternative database instance as demanded by Definition 5.4. The chase is formally introduced in the following Definition 7.28 and illustrated later in Example 7.29.

Definition 7.28 (Chase of existential R-sentences). *Consider a set $\mathcal{S} \subset \mathscr{L}_{\exists R}$ of existential R-sentences (which are "variable-disjoint" according to the remark to Definition 6.2) and a set Σ of FDs such that there exists an instance r with $r \models_{\mathsf{M}} \mathcal{S}$*

and $r \models_{\mathsf{M}} \Sigma$*. Moreover, let* $\mathcal{S}^*$ *denote the set of matrices (that is, quantifier-free parts) of the existential* R*-sentences in* $\mathcal{S}$*. We then call the constants* distinguished variables *and the formerly existentially quantified variables* non-distinguished variables *and assume the non-distinguished variables to be linearly ordered with ordering relation* $<_V$*. The application of an FD* σ *to formulas* $\chi_1^*, \chi_2^* \in \mathcal{S}^*$ *proceeds similar to the FD application according to Definition 7.1, but it replaces one of the original formulas instead of adding a new formula to* $\mathcal{S}^*$*:*

> *If* $\chi_1^*[K] = \chi_2^*[K]$ *for all* $K \in lhs(\sigma)$*,*
> *then, for all* $N \in rhs(\sigma)$*, set* $\chi_1^*[N] := \chi_2^*[N]$ *if one of the following holds:*[17]
>
> - $\chi_1^*[N]$ *is non-distinguished and* $\chi_2^*[N]$ *is distinguished or*
> - $\chi_1^*[N]$ *and* $\chi_2^*[N]$ *are both non-distinguished and* $\chi_1^*[N] <_V \chi_2^*[N]$*.*

The chase *of* $\mathcal{S}$ *with* Σ *is the set* $\mathcal{S}_\Sigma \subset \mathscr{L}_{\exists R}$ *that results from repeatedly applying the FDs of* Σ *to formulas of* $\mathcal{S}^*$ *until a fixpoint is reached (that is, none of the FDs is applicable or* $\mathcal{S}^*$ *is not changed by any further FD applications), and afterwards adding existential quantifiers for the remaining non-distinguished variables.*

Remarks. The precondition of Definition 7.28 (existence of an instance r with $r \models_{\mathsf{M}} \mathcal{S}$ and $r \models_{\mathsf{M}} \Sigma$) ensures that the sentences in $\mathcal{S}$ do not contradict the FDs Σ.

The original chase as introduced in, for example, [AHV95] is defined for a tableau query[18] and a set of functional and join dependencies. Interpreting the set $\mathcal{S}^*$ as tableau query, it can be seen that the original chase is a generalized form of our chase and therefore our chase inherits the properties of the original chase.

The application of FDs according to Definition 7.1 is correlated to a chase step according to Definition 7.28 as follows: Considering $\chi_1^*, \chi_2^* \in \mathcal{S}^*$ and $\sigma \in \Sigma$ (with $\chi_1^*[K] = \chi_2^*[K]$ for all $K \in lhs(\sigma)$), we can construct existential R-sentences χ_1 and χ_2 such that

- σ is applicable to χ_1 and χ_2 in the sense of Definition 7.1 and
- the result of the FD application corresponds to the result of the chase step.

[17] If both $\chi_1^*[N]$ and $\chi_2^*[N]$ are distinguished, it already holds that $\chi_1^*[N] = \chi_2^*[N]$; otherwise, χ_1^* and χ_2^* would contradict the FD $\sigma \in \Sigma$, which is, however, impossible due to the precondition of Definition 7.28.

[18] A tableau query $(\mathcal{T},\mu)$ consists of a tableau $\mathcal{T}$ (which is basically a set of tuples that may contain both variables and constants) and a tuple μ of variables such that each of these variables occurs in $\mathcal{T}$. The evaluation of $(\mathcal{T},\mu)$ is the set of tuples with the same structure as μ for which the pattern described by $\mathcal{T}$ occurs in the database instance. Please refer to [AHV95] for details.

Consequently, all results for FD application (in particular Lemma 7.3) also apply for the chase.

More precisely, the construction of χ_1 and χ_2 is performed in the following way (for $i \in \{1,2\}$):

- If $\chi_i^*[A]$ is distinguished then let $\chi_i[A] := \chi_i^*[A]$;
- If $\chi_i^*[A]$ is non-distinguished and $A \notin lhs(\sigma)$ then let $\chi_i[A] \in Var$ and add an existential quantifier for this variable;
- If $\chi_i^*[A]$ is non-distinguished and $A \in lhs(\sigma)$ then let $\chi_i[A] \in Const$ such that $\chi_i[A]$ is a "new" constant (that does not occur in χ_1^*, χ_2^*) and $\chi_1[A] = \chi_2[A]$ if and only if $\chi_1^*[A] = \chi_2^*[A]$.

For example, if $\chi_1^* \equiv R(a,X_1,c,X_2)$, $\chi_2^* \equiv R(a,X_1,X_3,d)$ (with a, c, d denoting distinguished variables and X_1, X_2, X_3 denoting non-distinguished variables) and $\sigma \equiv AB \rightarrow CD$, the corresponding existential R-sentences (in the sense of the above construction) are $\chi_1 \equiv (\exists X_2)R(a,c',c,X_2)$ and $\chi_2 \equiv (\exists X_3)R(a,c',X_3,d)$ with c' denoting a "new" constant. The result of applying σ to χ_1 and χ_2 is $\chi_\sigma \equiv R(a,c',c,d)$, and the chase step yields $\chi_1^* \equiv R(a,X_1,c,d)$ which obviously corresponds to χ_σ when re-replacing c' with X_1.

The following example together with Figure 7.6 on page 100 illustrate the chase procedure.

Example 7.29. Consider the relation schema $\langle R, ABC, \Sigma\rangle$ with $\Sigma = \{A \rightarrow B, A \rightarrow C\}$ and let the set $\mathcal{S}$ of existential R-sentences be given by

$$\mathcal{S} = \{(\exists X_1)R(a,b,X_1), (\exists X_2)R(a,b,X_2), (\exists X_3)R(a,X_3,c)\}.$$

For the instance r that consists of the single tuple $R(a,b,c)$ (which will also turn out to be the result of chasing $\mathcal{S}$ with Σ), it obviously holds that $r \models_{\mathsf{M}} \mathcal{S}$ and $r \models_{\mathsf{M}} \Sigma$; therefore, the precondition of Definition 7.28 is satisfied. The set $\mathcal{S}^*$ is then given by

$$\mathcal{S}^* = \{R(a,b,X_1), R(a,b,X_2), R(a,X_3,c)\}$$

with a, b, c denoting distinguished variables and X_1, X_2, X_3 denoting non-distinguished variables. The transition from $\mathcal{S}$ to $\mathcal{S}^*$ is depicted in Figure 7.6(a). We further assume that $X_1 <_V X_2 <_V X_3$.

The chase may now be performed in an arbitrary order on the elements of $\mathcal{S}^*$. We exemplarily describe one of these executions.

1. Apply $A \rightarrow C$ to $\chi_1^* \equiv R(a,b,X_1)$ and $\chi_2^* \equiv R(a,b,X_2)$. Since $X_1 <_V X_2$, we replace X_1 in χ_1^* with X_2. The set $\mathcal{S}^*$ is updated to $\mathcal{S}^* = \{R(a,b,X_2), R(a,X_3,c)\}$ (because of the set semantics, the duplicate $R(a,b,X_2)$ occurs only once in $\mathcal{S}^*$). This step is depicted in Figure 7.6(b).

2. Apply $A \to B$ to $\chi_1^* \equiv R(a,X_3,c)$ and $\chi_2^* \equiv R(a,b,X_2)$. Since X_3 is non-distinguished and b is distinguished, we replace X_3 in χ_1^* with b. The set $\mathcal{S}^*$ is updated to $\mathcal{S}^* = \{R(a,b,X_2), R(a,b,c)\}$.
   This step is depicted in Figure 7.6(c).

3. Apply $A \to C$ to $\chi_1^* \equiv R(a,b,X_2)$ and $\chi_2^* \equiv R(a,b,c)$. Since X_2 is non-distinguished and c is distinguished, we replace X_2 in χ_1^* with c. The set $\mathcal{S}^*$ is updated to $\mathcal{S}^* = \{R(a,b,c)\}$.
   This step is depicted in Figure 7.6(d).

Since now $\mathcal{S}^*$ is a singleton, no more applications of FDs are possible. Moreover, $\mathcal{S}^*$ does not contain any non-distinguished variables and thus we need not to add existential quantifiers. Therefore, the result of the chase is $\mathcal{S}_\Sigma = \{R(a,b,c)\}$. The transition from $\mathcal{S}^*$ to $\mathcal{S}_\Sigma$ is depicted in Figure 7.6(e). $\Diamond$

As previously mentioned, we apply the chase in the proof of Theorem 7.35. It serves as a mechanism that constructs an alternative database instance (as demanded by Definition 5.4), starting from the set of answers to a given query sequence. We call such an instance that emerges from the chase mechanism a chased instance. More precisely, it is defined as follows.

Definition 7.30 (Chased instance). *Let $\mathcal{S} \subset \mathscr{L}_{\exists R}$ and Σ a set of FDs such that the precondition of Definition 7.28 is satisfied. Moreover, $\mathcal{S}_\Sigma$ denotes the chase of $\mathcal{S}$ with Σ, Const_{old} is the set of constants occurring in $\mathcal{S}$, and Const_{new} is a set of "new" constants, that is, $\mathsf{Const}_{old} \cap \mathsf{Const}_{new} = \emptyset$. Now consider the set $\mathcal{S}_g := \{g(\chi_\Sigma) \mid \chi_\Sigma \in \mathcal{S}_\Sigma\}$ of ground atoms with*

$$g(\chi_\Sigma)[A] := \begin{cases} \chi_\Sigma[A] & \text{if } A \in \mathit{select}(\chi_\Sigma) \\ c \in \mathsf{Const}_{new} & \text{if } A \notin \mathit{select}(\chi_\Sigma) \end{cases},$$

such that equal variables in elements of $\mathcal{S}_\Sigma$ are replaced by equal constants from Const_{new} and distinct variables are replaced by distinct constants. A chased instance *$r_{\mathcal{S}_\Sigma}$ is a witness instance for $\mathcal{S}_g$, that is, for every ground atom χ_g it holds that*

$r_{\mathcal{S}_\Sigma} \models_{\mathsf{M}} \chi_g$ *if* $\chi_g \in \mathcal{S}_g$;
$r_{\mathcal{S}_\Sigma} \not\models_{\mathsf{M}} \chi_g$ *if* $\chi_g \notin \mathcal{S}_g$.

In other words, a chased instance is constructed by replacing all (existentially quantified) variables in the result of a chase by new constants. We illustrate the concept of a chased instance with the following example.

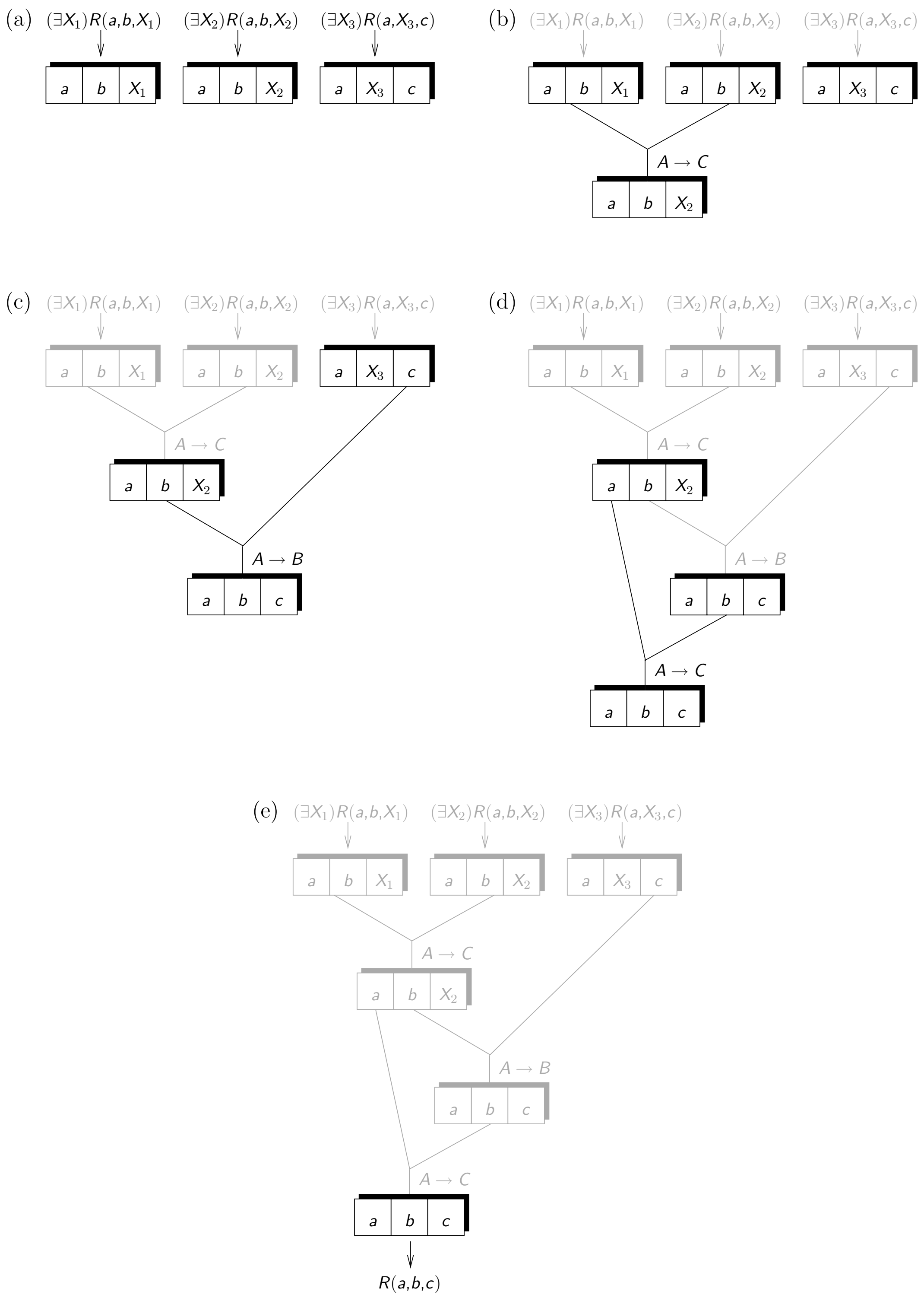

(a)
(b)
(c)
(d)
(e)
$(\exists X_1)R(a,b,X_1)$
$(\exists X_2)R(a,b,X_2)$
$(\exists X_3)R(a,X_3,c)$
$A \to C$
$A \to B$
$R(a,b,c)$

Figure 7.6: Illustration of the chase.

Example 7.31. Consider the relation schema $\langle R, ABC, \Sigma\rangle$ with the functional dependencies $\Sigma = \{A \to B, A \to C, C \to B\}$ and let the set $\mathcal{S}$ be given by

$$\mathcal{S} = \{\ (\exists X_1)R(a,b,X_1),\ (\exists X_2)(\exists X_3)R(X_2,X_3,c),\ (\exists X_4)R(X_4,d,e),$$
$$(\exists X_5)(\exists X_6)R(X_5,X_6,e),\ (\exists X_7)R(a,X_7,f)\ \}.$$

We assume that $X_1 <_V X_2 <_V ... <_V X_7$ and chase $\mathcal{S}$ with Σ. The precondition for the chase is satisfied, for example, by the instance $r = \{R(a,b,f), R(b,a,c), R(c,d,e)\}$ ($r \models_M \mathcal{S}$ and $r \models_M \Sigma$), and the chase result is

$$\mathcal{S}_\Sigma = \{\ \chi^1_\Sigma,\ \chi^2_\Sigma,\ \chi^3_\Sigma\ \} \text{ with}$$
$$\chi^1_\Sigma \equiv R(a,b,f),\ \chi^2_\Sigma \equiv (\exists X_2)(\exists X_3)R(X_2,X_3,c) \text{ and } \chi^3_\Sigma \equiv (\exists X_5)R(X_5,d,e).$$

For constructing a chased instance, we consider the set $Const_{new} = \{n_1, n_2, ...\}$ of new constants. Since $Const_{old} = \{a, b, c, d, e, f\}$, the condition $Const_{old} \cap Const_{new} = \emptyset$ is satisfied. The set $\mathcal{S}_g$ is then determined by replacing the variables in $\mathcal{S}_\Sigma$ with constants from $Const_{new}$:

$$\mathcal{S}_g = \{\ g(\chi^1_\Sigma),\ g(\chi^2_\Sigma),\ g(\chi^3_\Sigma)\ \} \text{ with}$$
$$g(\chi^1_\Sigma) \equiv R(a,b,f),\ g(\chi^2_\Sigma) \equiv R(n_1,n_2,c) \text{ and } g(\chi^3_\Sigma) \equiv R(n_3,d,e).$$

The chased instance $r_{\mathcal{S}_\Sigma}$ is the instance that makes exactly the ground atoms of $\mathcal{S}_g$ true and all other ground atoms false. We may represent $r_{\mathcal{S}_\Sigma}$ as the following table:

$r_{\mathcal{S}_\Sigma}$	A	B	C
	a	b	f
	n_1	n_2	c
	n_3	d	e

Observe that $r_{\mathcal{S}_\Sigma}$ is indeed an instance of the schema $\langle R, ABC, \Sigma\rangle$ because the FDs Σ are satisfied. $\Diamond$

We require the chase to be a "proper" algorithm, that is, it should terminate and be, in some sense, sound and complete. More precisely, regarding soundness, the chase should (syntactically) produce only existential R-sentences that are logically implied by the input, that is, by the set $\mathcal{S}$ and the functional dependencies Σ. Regarding completeness, we consider a chased instance $r_{\mathcal{S}_\Sigma}$ and demand that it is a model for all formulas being logically implied by $\mathcal{S} \cup \Sigma$; with the definition of logical implication in mind, this requirement is equivalent to $r_{\mathcal{S}_\Sigma}$ being a model of $\mathcal{S} \cup \Sigma$.

The original chase (see the remark to Definition 7.28) is known to satisfy the properties of termination, soundness and completeness. In the following Lemma 7.32, however, we re-elaborate these properties for our special form of the chase and show that they are actually satisfied.

Lemma 7.32 (Properties of the chase). *The chase according to Definition 7.28 has the following properties:*

1. *It always terminates and yields a unique*[19] *result (independent of the order of the FD applications).*
2. *It is sound in the sense that $\mathcal{S} \cup \Sigma \models \mathcal{S}_\Sigma$.*
3. *It is complete in the sense that $r_{\mathcal{S}_\Sigma} \models_{\mathsf{M}} \mathcal{S} \cup \Sigma$ for every chased instance $r_{\mathcal{S}_\Sigma}$ according to Definition 7.30.*

Proof. As explained above, each chase step can be simulated by an FD application in the sense of Definition 7.1. Therefore, we may argue about FD applications in $\mathcal{S}$ rather than about chase steps in $\mathcal{S}^*$ in the following.

Termination and *uniqueness* follow from the Church-Rosser property of the general chase, refer to [AHV95].

Regarding *soundness*, by the definition of logical implication, we have to show that for every instance r it holds that

$$\text{if } r \models_{\mathsf{M}} \mathcal{S} \cup \Sigma \text{ then } r \models_{\mathsf{M}} \mathcal{S}_\Sigma. \tag{7.36}$$

We choose an instance r such that

$$r \models_{\mathsf{M}} \mathcal{S} \cup \Sigma. \tag{7.37}$$

By the termination property of the chase, each $\chi_\Sigma \in \mathcal{S}_\Sigma$ results from a finite sequence of FD applications. In the first step, a χ_σ is produced by applying an FD $\sigma \in \Sigma$ to $\chi_1, \chi_2 \in \mathcal{S}$. From Lemma 7.3, we know that

$$\{\chi_1, \chi_2, \sigma\} \models \chi_\sigma. \tag{7.38}$$

Since $\mathcal{S} \models \{\chi_1, \chi_2\}$ and $\Sigma \models \sigma$, we get $\mathcal{S} \cup \Sigma \models \{\chi_1, \chi_2, \sigma\}$ which, together with (7.38) and the transitivity of logical implication, leads to $\mathcal{S} \cup \Sigma \models \chi_\sigma$. According to the chase definition, one of χ_1, χ_2 is replaced by χ_σ and then the next step is performed. We may therefore inductively apply the above argument and finally get $\mathcal{S} \cup \Sigma \models \chi_\Sigma$ for each $\chi_\Sigma \in \mathcal{S}_\Sigma$. With (7.37) and the definition of logical implication, it follows that $r \models_{\mathsf{M}} \chi_\Sigma$ for each $\chi_\Sigma \in \mathcal{S}_\Sigma$ and thus $r \models_{\mathsf{M}} \mathcal{S}_\Sigma$. Consequently, $\mathcal{S} \cup \Sigma \models \mathcal{S}_\Sigma$ which completes the soundness proof.

Regarding *completeness*, we have to show that for every chased instance $r_{\mathcal{S}_\Sigma}$ it holds that $r_{\mathcal{S}_\Sigma} \models_{\mathsf{M}} \mathcal{S}$ and $r_{\mathcal{S}_\Sigma} \models_{\mathsf{M}} \Sigma$. We choose an arbitrary chased instance $r_{\mathcal{S}_\Sigma}$ for the remainder of the proof.

[19]The result is unique up to variable identifiers.

First, we show "$r_{\mathcal{S}_\Sigma} \models_\mathsf{M} \mathcal{S}$": By Definition 7.1, for the result χ_σ of applying an FD $\sigma \in \Sigma$ to the existential R-sentences χ_1 and χ_2, it holds for $i \in \{1, 2\}$ that

$$\chi_\sigma[A] = \chi_i[A] \text{ for all attributes } A \in \mathit{select}(\chi_i). \tag{7.39}$$

By Definition 6.4 and Lemma 6.5, it follows from (7.39) that $\chi_\sigma \models \chi_i$ for $i \in \{1, 2\}$. Inductively applying this argument and the transitivity of logical implication, for each $\chi \in \mathcal{S}$ there exists a $\chi_\Sigma \in \mathcal{S}_\Sigma$ with

$$\chi_\Sigma \models \chi. \tag{7.40}$$

(If no FD is applied to a $\chi \in \mathcal{S}$ during the chase process, then $\chi \in \mathcal{S}_\Sigma$ and thus (7.40) holds with $\chi_\Sigma :\equiv \chi$.)
According to Definition 7.30, $r_{\mathcal{S}_\Sigma}$ is a witness instance for $\mathcal{S}_g$ which emerges from $\mathcal{S}_\Sigma$ by "grounding" the elements of $\mathcal{S}_\Sigma$. Consequently, by construction of $\mathcal{S}_g$, Definition 6.4 and Lemma 6.5, for each $\chi_\Sigma \in \mathcal{S}_\Sigma$ there exists a ground atom $\chi_g \in \mathcal{S}_g$ such that

$$\chi_g \models \chi_\Sigma. \tag{7.41}$$

Since $r_{\mathcal{S}_\Sigma} \models_\mathsf{M} \mathcal{S}_g$, it follows from (7.41) and (7.40) that $r_{\mathcal{S}_\Sigma} \models_\mathsf{M} \mathcal{S}$.

Second, we show "$r_{\mathcal{S}_\Sigma} \models_\mathsf{M} \Sigma$": We suppose the contrary, that is, there exists a $\sigma \in \Sigma$ such that

$$r_{\mathcal{S}_\Sigma} \not\models_\mathsf{M} \sigma. \tag{7.42}$$

By the definition of FDs and the construction of $r_{\mathcal{S}_\Sigma}$, (7.42) means: There exist ground atoms $\chi_{g_1}, \chi_{g_2} \in \mathcal{S}_g$ with $r_{\mathcal{S}_\Sigma} \models_\mathsf{M} \chi_{g_1}$ and $r_{\mathcal{S}_\Sigma} \models_\mathsf{M} \chi_{g_2}$, such that

$$\chi_{g_1}[A] = \chi_{g_2}[A] \text{ for all } A \in \mathit{lhs}(\sigma) \quad \text{but} \tag{7.43}$$

$$\chi_{g_1}[B] \neq \chi_{g_2}[B] \text{ for a } B \in \mathit{rhs}(\sigma). \tag{7.44}$$

Let $\chi_{\Sigma_1}, \chi_{\Sigma_2} \in \mathcal{S}_\Sigma$ be the existential R-sentences from which χ_{g_1} and χ_{g_2} emerged during the construction of $\mathcal{S}_g$. Due to this construction, (7.43) and (7.44) already hold for χ_{Σ_1} and χ_{Σ_2}, since equal attribute values in χ_{g_1} and χ_{g_2} result from equal attribute values in χ_{Σ_1} and χ_{Σ_2} and distinct attribute values in χ_{g_1} and χ_{g_2} result from distinct attribute values in χ_{Σ_1} and χ_{Σ_2}. Consequently, $\chi_{\Sigma_1}[A] = \chi_{\Sigma_2}[A]$ for all $A \in \mathit{lhs}(\sigma)$ which is, however, a contradiction to the exhaustive application of FDs as demanded by the chase definition. As a result, (7.42) is falsified and we get $r_{\mathcal{S}_\Sigma} \models_\mathsf{M} \Sigma$ which completes the proof. □

In the proof of Theorem 7.35, we will have to show that answers regarding the alternative database instance are also answers regarding the original database instance. A more general form of this property is captured by the following lemma. (Again, the original chase is already known to satisfy this property.)

Lemma 7.33 (Implications of chased instances). *Consider the chase $\mathcal{S}_\Sigma$ of a set $\mathcal{S} \subset \mathscr{L}_{\exists R}$ with a set Σ of FDs, and a chased instance $r_{\mathcal{S}_\Sigma}$. Then for a formula $\chi \in \mathscr{L}_{\exists R}$ with $r_{\mathcal{S}_\Sigma} \models_\mathsf{M} \chi$ and $\chi[A] \in \mathit{Const}_{old}$ for every attribute $A \in \mathit{select}(\chi)$ it holds that $\mathcal{S} \cup \Sigma \models \chi$.*

Proof. According to Lemma 6.7, from $r_{\mathcal{S}_\Sigma} \models_\mathsf{M} \chi$ it follows that there exists a ground atom χ_g with

$$r_{\mathcal{S}_\Sigma} \models_\mathsf{M} \chi_g \text{ and } \chi_g \models \chi. \tag{7.45}$$

The chase with FDs according to Definition 7.28 either replaces a non-distinguished variable with another non-distinguished variable or a non-distinguished variable with a distinguished variable from another formula in each step. Consequently, no constants from Const_{new} are introduced during the chase of $\mathcal{S}$ with Σ. Moreover, during the construction of $r_{\mathcal{S}_\Sigma}$ according to Definition 7.30, when $\mathcal{S}_g$ is generated from $\mathcal{S}_\Sigma$, only constants from Const_{new} are used to replace the variables. Thus, for the ground atom χ_g from (7.45) there exists a $\chi_\Sigma \in \mathcal{S}_\Sigma$ such that

$$\chi_\Sigma[A] = \chi_g[A] \text{ for all attributes } A \text{ with } \chi_g[A] \in \mathit{Const}_{old}. \tag{7.46}$$

Because of $\chi_g \models \chi$ (see (7.45)) it holds by Definition 6.4 and Lemma 6.5 that $\chi_g[A] = \chi[A]$ for every attribute $A \in \mathit{select}(\chi)$. Since we assume χ to contain only constants from Const_{old}, with (7.46) we conclude that

$$\chi_\Sigma[A] = \chi[A] \text{ for all attributes } A \in \mathit{select}(\chi). \tag{7.47}$$

Again, we apply Definition 6.4 and Lemma 6.5 and find that (7.47) is equivalent with $\chi_\Sigma \models \chi$. From the soundness of the chase according to Lemma 7.32, we know that $\mathcal{S} \cup \Sigma \models \chi_\Sigma$. Thus, applying the transitivity of the logical implication, we get $\mathcal{S} \cup \Sigma \models \chi$. □

Finally, we need a variant of Lemma 6.9 that is adapted to sets of (positive) existential R-sentences. It states that a potential secret is implied by a set of existential R-sentences and a set of FDs if and only if it is already implied by a single element of the set of existential R-sentences.

Lemma 7.34 (Implications of existential R-sentences and FDs). *Let $R_S = \langle R, \mathcal{U}, \Sigma \rangle$ be a relation schema in ONF, $\mathcal{S} \subset \mathscr{L}_{\exists R}$ a set of existential R-sentences, and $\Psi \in \mathscr{L}_{\exists R}$ a potential secret with $\mathit{select}(\Psi) \in \mathit{bfs}(R_S)$. Then the following properties are equivalent:*

1. $\mathcal{S} \cup \Sigma \models \Psi$.

2. $\chi \models \Psi$ *for a* $\chi \in \mathcal{S}$.

Proof. First, we show "1. ⇒ 2." and therefore assume that

$$\mathcal{S} \cup \Sigma \models \Psi. \tag{7.48}$$

Since the chase has been proven to be complete in Lemma 7.32, we know that

$$r_{\mathcal{S}_\Sigma} \models_{\mathsf{M}} \mathcal{S} \cup \Sigma \tag{7.49}$$

for all chased instances $r_{\mathcal{S}_\Sigma}$. In the following, we choose an arbitrary chased instance $r_{\mathcal{S}_\Sigma}$ and consider the set Const_{old} of constants occurring in $\mathcal{S} \cup \{\Psi\}$ and the set Const_{new} of constants that are newly introduced in the construction of $r_{\mathcal{S}_\Sigma}$. Observe that (7.49) together with (7.48) leads to

$$r_{\mathcal{S}_\Sigma} \models_{\mathsf{M}} \Psi \tag{7.50}$$

by definition of logical implication. Thus, by Lemma 6.7, there exists a ground atom Ψ_g with $r_{\mathcal{S}_\Sigma} \models_{\mathsf{M}} \Psi_g$ and $\Psi_g \models \Psi$. By Definition 6.4 and Lemma 6.5, it follows that

$$\Psi_g[A] = \Psi[A] \text{ for all attributes } A \in \mathit{select}(\Psi). \tag{7.51}$$

Since we assume that $\mathit{select}(\Psi) \subset \mathit{Const}_{old}$, we conclude from the construction of $r_{\mathcal{S}_\Sigma}$ that there exists a $\Psi_\Sigma \in \mathcal{S}_\Sigma$ such that $\Psi_\Sigma \models \Psi$. Since $\mathcal{S}_\Sigma$ emerges from a sequence of FD applications to formulas from $\mathcal{S}$, we can inductively apply Lemma 7.8 to Ψ_Σ and finally get $\chi \models \Psi$ for a $\chi \in \mathcal{S}$. (Note that Lemma 7.8 is applicable in this context, since we assume that $\mathit{select}(\Psi) \in \mathit{bfs}(R_S)$.)

Second, the direction "2. ⇒ 1." immediately holds by the monotonicity of first-order logic. □

Having defined the notions of chase and chased instance and having presented some results regarding the chase, we now show that hiding the policy from the database user suffices to ensure preservation of confidentiality.

Theorem 7.35 (scqe^{oq} preserves confidentiality). *Reconsider the assumptions from Theorem 7.10, but let the queries in Q be elements of $\mathscr{L}_{\exists R,fv}$ and suppose that $\mathit{pot\_sec}$ is unknown to the user. Then scqe^{oq} is secure in the sense of Definition 5.4.*

Proof. In this proof, we use the following abbreviations:

$$\begin{array}{lcllcl}
\mathit{eval}_i^* & := & \mathit{eval}^*(\Phi_i(\vec{V}_i))(r) & \mathit{eval}_i^{*\prime} & := & \mathit{eval}^*(\Phi_i(\vec{V}_i))(r') \\
\mathit{refused}_i & := & \mathit{refused}(\Phi_i(\vec{V}_i),\mathit{pot\_sec}) & \mathit{refused}_i' & := & \mathit{refused}(\Phi_i(\vec{V}_i),\mathit{pot\_sec}') \\
\mathit{ans}_i & := & \mathit{eval}_i^* \backslash \mathit{refused}_i & \mathit{ans}_i' & := & \mathit{eval}_i^{*\prime} \backslash \mathit{refused}_i'
\end{array}$$

Moreover, with $\langle ans_1, ans_2, ...\rangle$ we denote the answer sequence to Q produced by $scqe^{oq}$.

Consider the instance r and the policy pot_sec (with (r, pot_sec) being admissible with respect to the a priori user knowledge log_0 according to Definition 5.3), a fixed potential secret $\Psi \in pot\_sec$, and a finite prefix $Q' = \langle \Phi_1(\vec{V}_1), \Phi_2(\vec{V}_2), \dots, \Phi_n(\vec{V}_n)\rangle$ of the query sequence Q. According to Definition 5.4, we have to find an alternative instance r' and an alternative policy pot_sec' satisfying the following properties:

$$r' \models_{\mathsf{M}} \Sigma \tag{7.52}$$
[r' is an instance of the underlying schema]

$$\text{for all } \Psi' \in pot\_sec' : \Psi' \in \mathscr{L}_{\exists R} \text{ and } select(\Psi') \in bfs(R_S) \tag{7.53}$$
[pot_sec' is a valid policy]

$$(r', pot\_sec') \text{ is admissible with respect to } log_0 \tag{7.54}$$
[admissibility in the sense of Definition 5.3]

$$ans_i = ans'_i \text{ for } i \in \{1, \dots, n\} \tag{7.55}$$
[$scqe^{oq}$ yields the same answers on (r, pot_sec) and (r', pot_sec')]

$$eval^*(\Psi)(r') = \neg\Psi \tag{7.56}$$
[Ψ is false in r']

Observe that, unlike in previous confidentiality proofs, it does not need to hold that $pot\_sec' = pot\_sec$. This is due to the assumption that the confidentiality policy is hidden from the user, see Definition 5.4.

In the following, we first construct the alternative instance r' and the alternative policy pot_sec' such that properties (7.52) and (7.53) are satisfied. After that, we prove that r' and pot_sec' also satisfy properties (7.54)–(7.56).

Construction of r' and pot_sec': Consider the answer sets $ans_1, ans_2, \dots, ans_n$ to the queries of Q'. We construct r' as a chased instance, see Definition 7.30, that emerges from the chase of $\bigcup_{i=1}^{n} ans_i$ with Σ: By the definition of $scqe^{oq}$, it holds that $r \models_{\mathsf{M}} \bigcup_{i=1}^{n} ans_i$ and, moreover, $r \models_{\mathsf{M}} \Sigma$, since r is a proper instance; thus, r satisfies the precondition of Definition 7.28 which justifies the applicability of Definition 7.30. We denote the result of the chase with Ans_Σ and the set of ground atoms that is constructed from Ans_Σ in order to obtain the chased instance with Ans_g. Furthermore, we denote the set of constants that occur in $ans_1, \dots, ans_n$ and Ψ with $Const_{old}$ and the set of constants that are newly introduced during the construction of Ans_g with $Const_{new}$ such that $Const_{old} \cap Const_{new} = \emptyset$. By Lemma 7.32 (completeness of the chase), it holds that

$$r' \models_{\mathsf{M}} \Sigma, \tag{7.57}$$

that is, r' is actually an instance as demanded by (7.52).

For the alternative policy pot_sec', we generate a potential secret from the language $\mathscr{L}_{\exists R}$ for each new constant occurring in Ans_g:

$$\begin{aligned} pot\_sec' := \{\Psi' \mid\ & \text{exists } \chi_g \in Ans_g \text{ such that for an attribute } A \\ & \chi_g[A] \in Const_{new} \text{ and } \Psi'[A] = \chi_g[A] \text{ and} \\ & \Psi'[A'] \in Var \text{ for all } A' \in \mathcal{U}\backslash\{A\}\ \} \end{aligned} \tag{7.58}$$

Since each $\Psi' \in pot\_sec'$ has the form

$$\begin{aligned} & (\exists X_1)(\exists X_2) \ldots (\exists X_{n-1}) R(v_1, v_2, \ldots, v_n) \\ & \quad \text{with } v_i \in Const \text{ for exactly one } i \in \{1, \ldots, n\} \\ & \quad \text{and } \{v_1, \ldots, v_{i-1}, v_{i+1}, \ldots, v_n\} = \{X_1, \ldots, X_{n-1}\}, \end{aligned}$$

it holds that $select(\Psi') \in bfs(R_S)$ (by the remark to Definition 7.5) as demanded by (7.53).

Property (7.54): According to Definition 5.3, we have to show that

$$r' \models_M log_0; \text{ and} \tag{7.59}$$

$$\text{for every } \Psi' \in pot\_sec' : \text{ if } r' \models_M \Psi' \text{ then } log_0 \not\models \Psi'. \tag{7.60}$$

Since $log_0 = \Sigma$, (7.59) is satisfied by (7.52). Moreover, $log_0 \not\models \chi$ for all formulas $\chi \in \mathscr{L}_{\exists R}$ by the structure of log_0; thus, (7.60) is trivially satisfied.

Property (7.55): We choose an arbitrary i and show that for all formulas $\chi \in \mathscr{L}_{\exists R}$ which represent elements of answer sets it holds that

$$\chi \in ans_i \quad \text{if and only if} \quad \chi \in ans'_i. \tag{7.61}$$

For the "⇒"-part, consider a $\chi \in ans_i$. We show that $\chi \in eval_i^{*\prime}$ and $\chi \notin refused'_i$ (which is equivalent to $\chi \in ans'_i$):

First, by Definition 7.26 and Definition 2.3,

$$\chi \in eval_i^* = \{\Phi_i(\vec{c}_i) \mid \vec{c}_i \in Const \times \ldots \times Const \text{ and } r \models_M \Phi_i(\vec{c}_i)\}. \tag{7.62}$$

Moreover, by the completeness of the chase according to Lemma 7.32, we know that $r' \models_M \bigcup_{i=1}^n ans_i$ and therefore in particular

$$r' \models_M \chi. \tag{7.63}$$

Together with (7.62), we get

$$\chi \in \{\Phi_i(\vec{c}_i) \mid \vec{c}_i \in Const \times \ldots \times Const \text{ and } r' \models_M \Phi_i(\vec{c}_i)\} = eval_i^{*\prime}. \tag{7.64}$$

Second, we indirectly show that $\chi \notin \mathit{refused}'_i$ and therefore assume that $\chi \in \mathit{refused}'_i$. Then, by Definition 7.26, there exists a $\Psi' \in \mathit{pot\_sec}'$ such that $\chi \models \Psi'$. By Definition 6.4 and Lemma 6.5 it follows that

$$\chi[A] = \Psi'[A] \text{ for every } A \in \mathit{select}(\Psi'). \tag{7.65}$$

By construction of $\mathit{pot\_sec}'$ according to (7.58), we know that

$$\Psi'[A] \in \mathit{Const}_{\mathit{new}} \text{ for every } A \in \mathit{select}(\Psi'), \tag{7.66}$$

and thus, with (7.65),

$$\chi[A] \in \mathit{Const}_{\mathit{new}} \text{ for every } A \in \mathit{select}(\Psi'), \tag{7.67}$$

which is a contradiction to the definition of new constants. Therefore, $\chi \notin \mathit{refused}'_i$.

For the "$\Leftarrow$"-part, consider a $\chi \in \mathit{ans}'_i$. We show that $\chi \in \mathit{eval}^*_i$ and $\chi \notin \mathit{refused}_i$ (which is equivalent to $\chi \in \mathit{ans}_i$):

First, by the same argument as in the "$\Rightarrow$"-direction, $\chi \in \mathit{eval}^{*\prime}_i$ and thus $r' \models_{\mathsf{M}} \chi$. We now show indirectly that χ cannot contain constants from $\mathit{Const}_{\mathit{new}}$ and therefore assume the contrary:

$$\chi[A] \in \mathit{Const}_{\mathit{new}} \text{ for an attribute } A \in \mathit{select}(\chi). \tag{7.68}$$

By construction of r' and Lemma 6.7, there exists a $\chi_g \in \mathit{Ans}_g$ with $r' \models_{\mathsf{M}} \chi_g$ and $\chi_g \models \chi$. It follows by Definition 6.4 and Lemma 6.5 that

$$\chi_g[A] = \chi[A] \text{ for every } A \in \mathit{select}(\chi). \tag{7.69}$$

With (7.68) we get

$$\chi_g[A] \in \mathit{Const}_{\mathit{new}} \text{ for an attribute } A \in \mathit{select}(\chi) \tag{7.70}$$

and thus, by construction of $\mathit{pot\_sec}'$, there exists a $\Psi' \in \mathit{pot\_sec}'$ such that

$$\Psi'[A] = \chi_g[A] \text{ and } \Psi'[B] \in \mathit{Var} \text{ for all attributes } B \in \mathcal{U} \backslash \{A\}. \tag{7.71}$$

From (7.70) and (7.71) we conclude that $\chi[A] = \Psi'[A]$ for each attribute $A \in \mathit{select}(\Psi')$ which, by Definition 6.4 and Lemma 6.5, leads to $\chi \models \Psi'$ and thus $\chi \in \mathit{refused}'_i$, a contradiction to $\chi \in \mathit{ans}'_i$. It follows that $\chi[A] \in \mathit{Const}_{\mathit{old}}$ for every $A \in \mathit{select}(\chi)$. Since $\mathcal{S} := \bigcup_{i=1}^{n} \mathit{ans}_i \subset \mathscr{L}_{\exists R}$, r' is chased instance to $\mathcal{S}$

and Σ with $r' \models_{\mathsf{M}} \chi$, and $\chi[A] \in \mathit{Const}_{old}$ for every $A \in \mathit{select}(\chi)$, we can apply Lemma 7.33 and get

$$\bigcup_{i=1}^{n} \mathit{ans}_i \cup \Sigma \models \chi. \tag{7.72}$$

By Definition 7.26 and Definition 2.3, we know that

$$r \models_{\mathsf{M}} \bigcup_{i=1}^{n} \mathit{ans}_i, \tag{7.73}$$

and, since $(r, \mathit{pot\_sec})$ is assumed to be admissible with respect to $\mathit{log}_0 = \Sigma$, it holds that

$$r \models_{\mathsf{M}} \Sigma. \tag{7.74}$$

From (7.73), (7.74), (7.72) and the definition of logical implication, we conclude that $r \models_{\mathsf{M}} \chi$ and thus

$$\chi \in \{\Phi_i(\vec{c}_i) \mid \vec{c}_i \in \mathit{Const} \times \ldots \times \mathit{Const} \text{ and } r \models_{\mathsf{M}} \Phi_i(\vec{c}_i)\}, \tag{7.75}$$

that is, $\chi \in \mathit{eval}_i^*$.

Second, we show that $\chi \notin \mathit{refused}_i$, again indirectly by assuming $\chi \in \mathit{refused}_i$, which means that there exists a $\Psi' \in \mathit{pot\_sec}$ such that $\chi \models \Psi'$ (Definition 7.26). From (7.72), it then follows by transitivity of logical implication that

$$\bigcup_{i=1}^{n} \mathit{ans}_i \cup \Sigma \models \Psi'. \tag{7.76}$$

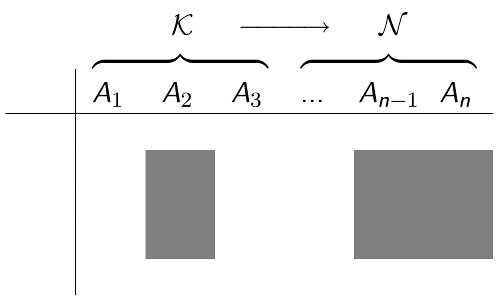
Since $\bigcup_{i=1}^{n} \mathit{ans}_i \subset \mathscr{L}_{\exists R}$ and $\Psi' \in \mathscr{L}_{\exists R}$ with $\mathit{select}(\Psi') \in \mathit{bfs}(R_S)$ by assumption, we can apply Lemma 7.34 which leads to

$$\chi' \models \Psi' \text{ for a } \chi' \in \bigcup_{i=1}^{n} \mathit{ans}_i \tag{7.77}$$

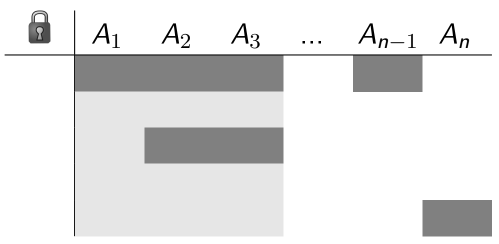
which means that

$$\chi' \in \bigcup_{i=1}^{n} \mathit{refused}_i \tag{7.78}$$

by Definition 7.26. This is obviously a contradiction to $\chi' \in \bigcup_{i=1}^{n} \mathit{ans}_i$ and thus $\chi \notin \mathit{refused}_i$.


$\mathcal{K} \longrightarrow \mathcal{N}$
A_1 A_2 A_3 ... A_{n-1} A_n

(a) schema and query restrictions



(b) policy restrictions

Figure 7.7: Visualization of the restrictions for open queries.

Property (7.56): We show this property by an indirect proof and therefore assume that $eval^*(\Psi)(r') = \Psi$ which, by Definition 2.2, means that

$$r' \models_M \Psi. \tag{7.79}$$

By (7.79) and the assumption that $\Psi[A] \in Const_{old}$ for every $A \in select(\Psi)$, we can apply Lemma 7.33 to $\mathcal{S} := \bigcup_{i=1}^{n} ans_i$, Σ and Ψ which yields

$$\bigcup_{i=1}^{n} ans_i \cup \Sigma \models \Psi. \tag{7.80}$$

By Lemma 7.34, (7.80) is equivalent to

$$\chi' \models \Psi \text{ for a } \chi' \in \bigcup_{i=1}^{n} ans_i \tag{7.81}$$

which again means that $\chi' \in \bigcup_{i=1}^{n} refused_i$ and is therefore a contradiction to $\chi' \in \bigcup_{i=1}^{n} ans_i$. It follows that $eval^*(\Psi)(r') = \neg\Psi$. □

As in the preceding sections, we again provide a visualization of the restrictions which are assumed in this section in Figure 7.7. This figure differs from Figure 7.2 as follows:

- With open queries, the user is not only able to ask for (parts of) single tuples but also for sets of (parts of) tuples by using free variables; this is indicated by the gray blocks in Figure 7.7(a).
- The policy is supposed to be unknown to the user; this is indicated by the padlock icon in Figure 7.7(b).

7.4 Open Confidentiality Policies

In the preceding section, we enhanced the query language by free variables. The resulting language $\mathscr{L}_{\exists R,fv}$ is suitable for expressing open queries—a class of queries that is highly relevant from the perspective of the database user. It might, however, also be desirable for the security administrator to have free variables at his disposal to express "secret schemas", as already sketched in the introduction to Chapter 7. Suppose, the database administrator aims at protecting all instantiations of a subset of attributes in a database instance. If we take $\mathscr{L}_{\exists R}$ (or $\mathscr{L}_{\exists R}^{\vee}$) as policy language, a problem arises: The secrets are declared independently of the actual database instance and thus the security administrator must add a secret to the policy for each possible instantiation of the attribute set under consideration. Since we assume an infinite set of constants, however, the resulting confidentiality policy would also be infinite and thus not representable in a closed form. If the security administrator avoids this problem by declaring only those instantiations as potential secrets that are actually true in the database instance, the database user would be able to disclose these secrets by simply reading the policy. As with open queries, we could hide the policy from the user, but this is not necessary as we will see in the following and would be in contrast to our "smart user assumption" from Section 5.1. Besides, it is desirable that the policy can be declared independently of the actual database instance in order to confirm the functional separation of database administrator and security administrator as proposed in Section 5.1. We illustrate the sketched problems with an example.

Example 7.36. Consider relation schema ACC_S and instance acc from Example 5.1 and suppose that the security administrator wants to protect each account holder of bank A, that is, each instantiation of ACC_HOLDER that occurs with $BANK = bankA$. With a policy from the language $\mathscr{L}_{\exists R}$, this can be expressed by enumerating all elements from $Const$ as instantiations for ACC_HOLDER:

$$\begin{aligned} pot\_sec_1 = \{\ & (\exists X_{ACC\_NO})(\exists X_{BAL})ACC(bankA, X_{ACC\_NO}, anderson, X_{BAL}),\\ & (\exists X_{ACC\_NO})(\exists X_{BAL})ACC(bankA, X_{ACC\_NO}, brown, X_{BAL}),\\ & (\exists X_{ACC\_NO})(\exists X_{BAL})ACC(bankA, X_{ACC\_NO}, jones, X_{BAL}), \ldots\ \ \} \end{aligned}$$

This is, however, an infinite policy. Alternatively, the security administrator could declare only those potential secrets that are true in acc, that is,

$$\begin{aligned} pot\_sec_2 = \{\ & (\exists X_{ACC\_NO})(\exists X_{BAL})ACC(bankA, X_{ACC\_NO}, smith, X_{BAL}),\\ & (\exists X_{ACC\_NO})(\exists X_{BAL})ACC(bankA, X_{ACC\_NO}, jones, X_{BAL})\ \ \}. \end{aligned}$$

A database user being aware of the construction of pot_sec_2 knows that $acc \models_M pot\_sec_2$. If he additionally knows the actual contents of pot_sec_2 (which is a justified requirement according to Section 5.1), he is able to disclose the declared secrets. $\Diamond$

Motivated by this example, we investigate open confidentiality policies in this section. In Subsection 7.4.1, we start with relating open policies to closed policies, adapting the security definition from Section 5.2 to open policies, and proposing a stateless CQE with a suitable censor. In Subsection 7.4.2, we propose an alternative censor for open policies that can be interpreted algorithmically, prove it equivalent to the censor from Subsection 7.4.1, and show that the stateless CQE for open policies preserves confidentiality.

This section is based on the work of Biskup, Lochner and Sonntag [BLS09] who originally investigated open confidentiality policies.

7.4.1 Policy Language and Adapted Stateless CQE

We take $\mathscr{L}_{\exists R,fv}$ as policy language, that is, the security administrator might use free variables to express "secret schemas". For simplicity, we assume $\mathscr{L}_{\exists R}$ as query language and do not consider disjunctive secrets in this section. In Section 7.5, we will investigate how these languages may be enhanced by Boolean operations.

Regarding semantics, a policy with secrets from $\mathscr{L}_{\exists R,fv}$ can be transformed into an (infinite) policy with secrets from $\mathscr{L}_{\exists R}$ via *expansion*.

Definition 7.37 (Expansion of open policies). *Let $\mathit{pot\_sec} \subset \mathscr{L}_{\exists R,fv}$ be an open confidentiality policy. The* expansion of an open potential secret $\Psi(\vec{V}) \in \mathit{pot\_sec}$ *is defined by*

$$\mathit{ex}(\Psi(\vec{V})) := \{\Psi(\vec{c}) \mid \vec{c} \in \mathit{Const} \times \ldots \times \mathit{Const}\}.$$

The expansion of $\mathit{pot\_sec}$ *is defined by*

$$\mathit{ex}(\mathit{pot\_sec}) := \bigcup_{\Psi(\vec{V}) \in \mathit{pot\_sec}} \mathit{ex}(\Psi(\vec{V})).$$

An open policy is supposed to have the same semantics as its expansion, which is basically expressed by the following Definitions 7.38–7.40.

First, we suitably adapt the notions of admissibility and secure query evaluation, as introduced in Section 5.2, to open policies. Considering an open policy as short representation of its expansion, it is easy to adapt these Definitions 5.3 and 5.4.

Definition 7.38 (Admissibility (for open policies)). *Given the a priori user knowledge log_0, a pair $(r,\mathit{pot\_sec})$ of a database instance r and a confidentiality policy $\mathit{pot\_sec} \subset \mathscr{L}_{\exists R,fv}$ is called* admissible with respect to log_0 *if the following conditions hold:*

1. $r \models_{\mathsf{M}} \mathit{log}_0$;

2. *a) if* $\mathsf{pot\_sec}$ *is known to the user:* $\mathsf{log}_0 \not\models \Psi(\vec{c})$ *for every* $\Psi(\vec{c}) \in \mathsf{ex}(\mathsf{pot\_sec})$*,*

   b) if $\mathsf{pot\_sec}$ *is unknown to the user:*
   $\mathsf{log}_0 \not\models \Psi(\vec{c})$ *for every* $\Psi(\vec{c}) \in \mathsf{ex}(\mathsf{pot\_sec})$ *with* $r \models_{\mathsf{M}} \Psi(\vec{c})$*.*

Definition 7.39 (Secure query evaluation for open policies). *Given*

- *a (possibly infinite) query sequence* $Q = \langle \Phi_1, \Phi_2, \ldots \rangle$ *(with* $\Phi_1, \Phi_2, \ldots$ *being queries expressed in a suitable query language) and*
- *a confidentiality policy* $\mathsf{pot\_sec} = \{\Psi_1(\vec{V_1}), \Psi_2(\vec{V_2}), \ldots, \Psi_m(\vec{V_m})\} \subset \mathscr{L}_{\exists R, fv}$*,*

the modified query evaluation $\mathsf{m\_eval}$ *is* secure with respect to $\mathsf{pot\_sec}$ *if*

for every finite prefix Q' *of* Q
for every $\Psi(\vec{c}) \in \mathsf{ex}(\mathsf{pot\_sec})$
for every database instance r
  such that $(r, \mathsf{pot\_sec})$ *is admissible*
  with respect to the a priori knowledge log_0
there exists an alternative instance r' *and*
there exists an alternative policy $\mathsf{pot\_sec}'$
  such that $(r', \mathsf{pot\_sec}')$ *is admissible*
  with respect to the a priori knowledge log_0
satisfying the following properties:

1. $\mathsf{m\_eval}(Q')(r, \mathsf{pot\_sec}) = \mathsf{m\_eval}(Q')(r', \mathsf{pot\_sec}')$*;*
2. $\mathsf{eval}^*(\Psi(\vec{c}))(r') = \neg\Psi(\vec{c})$*;*
3. *if* $\mathsf{pot\_sec}$ *is known to the database user, then* $\mathsf{pot\_sec}' = \mathsf{pot\_sec}$*.*

The modified query evaluation $\mathsf{m\_eval}$ *is* secure *if it is secure with respect to every possible policy* $\mathsf{pot\_sec}$*.*

Finally, we adapt the stateless CQE *scqe*, as introduced in Definition 6.11, to open policies. It suffices to slightly adjust the censor definition. The actual CQE mechanism remains the same but uses the adjusted censor.

Definition 7.40 (Stateless CQE for secrets from $\mathscr{L}_{\exists \mathsf{R}, \mathsf{fv}}$). *The stateless CQE* scqe_{op} *is a specific modified query evaluation according to Definition 5.2. Given a policy* $\mathsf{pot\_sec} \subset \mathscr{L}_{\exists R, fv}$*, the answers* $\mathsf{ans}_1, \mathsf{ans}_2, \ldots$ *to a query sequence* $Q = \langle \Phi_1, \Phi_2, \ldots \rangle$*, with* $\Phi_1, \Phi_2, \ldots \in \mathscr{L}_{\exists R}$*, are determined subject to the censor function*

$$\mathsf{censor}^{op}_{\exists R}(\mathsf{pot\_sec}, \Phi) := (\textit{exists } \Psi(\vec{c}))[\Psi(\vec{c}) \in \mathsf{ex}(\mathsf{pot\_sec}) \textit{ and } \Phi \models \Psi(\vec{c})].$$

The stateless CQE is then defined as follows:

$$scqe_{op}(Q)(r, pot\_sec) = \langle ans_1, ans_2, \ldots\rangle \quad \textit{with}$$

$$\begin{array}{rcl} ans_i & := & \texttt{if}\ censor^{op}_{\exists R}(pot\_sec, \Phi_i) \\ & & \texttt{then}\ mum \\ & & \texttt{else}\ eval^{*}(\Phi_i)(r) \end{array}$$

Unfortunately, the adjusted censor according to this definition has a major drawback: It has to scan the infinite expansion of an open policy and is thus not algorithmically implementable in this form. We take account of this problem in the following subsection.

7.4.2 Alternative Censor and Preservation of Confidentiality

The censor $censor^{op}_{\exists R}$, as introduced in Definition 7.40, has been defined in a "natural way" on the basis of the original censor for select-queries, see Definition 6.11. In order to construct a censor that is effectively implementable, however, it is not appropriate to define the censor via the infinite expansion of the confidentiality policy. Rather, it should be formulated in terms of the finite policy.

Observe that a potential secret $\Psi(\vec{V}) \in \mathscr{L}_{\exists R, fv}$ should be relevant for a query $\Phi \in \mathscr{L}_{\exists R}$ (in the sense of an adapted form of Definition 6.4) if

- each constant from $\Psi(\vec{V})$ occurs in Φ at the same position (that is, attribute) and
- each attribute instantiated with a free variable in $\Psi(\vec{V})$ is instantiated with a constant in Φ.

We formalize this notion in an alternative censor definition for $scqe_{op}$. In the following, we denote the set of variables that occur freely in a formula χ with $fv(\chi)$.

Definition 7.41 (Alternative censor for open policies). *Consider an open policy $pot\_sec \subset \mathscr{L}_{\exists R, fv}$, a query $\Phi \in \mathscr{L}_{\exists R}$, and the underlying relation schema $\langle R, \mathcal{U}, \Sigma\rangle$. The alternative censor for open confidentiality policies, $\widetilde{censor}^{op}_{\exists R}$, is then defined as follows:*

$$\begin{array}{rl} \widetilde{censor}^{op}_{\exists R}(pot\_sec, \Phi) := & (\textit{exists}\ \Psi(\vec{V}))[\Psi(\vec{V}) \in pot\_sec\ \textit{and for all}\ A \in \mathcal{U} : \\ & \textit{if}\ \Psi(\vec{V})[A] \in Const\ \textit{then}\ \Phi[A] = \Psi(\vec{V})[A]\ \textit{and} \\ & \textit{if}\ \Psi(\vec{V})[A] \in fv(\Psi(\vec{V}))\ \textit{then}\ \Phi[A] \in Const] \end{array}$$

Remark. Instead of proposing an alternative censor, we could also redefine the notion of relevance (see Definition 6.4) and investigate the relation between this new notion of relevance and logical implication (in the sense of Lemma 6.5). We prefer the alternative censor, however, so as not to get confused with the notion of relevance which is widely used throughout this thesis.

This alternative censor is in fact equivalent to $\mathit{censor}^{op}_{\exists R}$ as we show in the following lemma. We therefore assume hereafter that scqe_{op} uses $\widetilde{\mathit{censor}}^{op}_{\exists R}$ instead of $\mathit{censor}^{op}_{\exists R}$.

Lemma 7.42 (Equivalence of censors for open policies). *For an arbitrary query $\Phi \in \mathscr{L}_{\exists R}$ and an arbitrary policy $\mathit{pot\_sec} \subset \mathscr{L}_{\exists R,\mathit{fv}}$, it holds that $\mathit{censor}^{op}_{\exists R}(\mathit{pot\_sec},\Phi) = \widetilde{\mathit{censor}}^{op}_{\exists R}(\mathit{pot\_sec},\Phi)$.*

Proof. Assuming $\langle R,\mathcal{U},\Sigma\rangle$ as underlying relation schema, we show that for every $\Psi(\vec{V}) \in \mathit{pot\_sec}$ the following properties are equivalent:

1. [$\mathit{censor}^{op}_{\exists R}$] There exists a constant vector $\vec{c}$ with $\Psi(\vec{c}) \in \mathit{ex}(\Psi(\vec{V}))$ and $\Phi \models \Psi(\vec{c})$.
2. [$\widetilde{\mathit{censor}}^{op}_{\exists R}$] For all $A \in \mathcal{U}$ it holds that
   a) if $\Psi(\vec{V})[A] \in \mathit{Const}$ then $\Phi[A] = \Psi(\vec{V})[A]$;
   b) if $\Psi(\vec{V})[A] \in \mathit{fv}(\Psi(\vec{V}))$ then $\Phi[A] \in \mathit{Const}$.

First, we show "1. ⇒ 2.": Consider a constant vector $\vec{c}$ such that $\Psi(\vec{c}) \in \mathit{ex}(\Psi(\vec{V}))$ and

$$\Phi \models \Psi(\vec{c}). \tag{7.82}$$

We now consider properties 2 a) and 2 b) separately.

Regarding 2 a), by Definition 7.37, if $\Psi(\vec{V})[A] \in \mathit{Const}$ for an attribute $A \in \mathcal{U}$, then $\Psi(\vec{c})$ agrees with $\Psi(\vec{V})$ on this attribute, that is,

$$\Psi(\vec{c})[A] = \Psi(\vec{V})[A] \text{ for every } A \in \mathit{select}(\Psi(\vec{V})). \tag{7.83}$$

Moreover, by Definition 6.4 and Lemma 6.5, we conclude from (7.82) that

$$\Phi[A] = \Psi(\vec{c})[A] \text{ for every } A \in \mathit{select}(\Psi(\vec{c})). \tag{7.84}$$

Combining (7.83) and (7.84) then yields

$$\Phi[A] = \Psi(\vec{V})[A] \text{ for every } A \in \mathit{select}(\Psi(\vec{V})), \tag{7.85}$$

which is equivalent to 2 a).

Regarding 2 b), we suppose that $\Psi(\vec{V})[A] \in \mathit{fv}(\Psi(\vec{V}))$ for an attribute A. By Definition 7.37, this means that $\Psi(\vec{c})[A] \in \mathit{Const}$. From (7.84) it then follows that $\Phi[A] \in \mathit{Const}$.

Second, we show "2. $\Rightarrow$ 1.": Consider a secret $\Psi(\vec{V}) \in \mathit{pot\_sec}$ that satisfies property 2. Let $A_1, \dots, A_k$ be the attributes with $\Psi(\vec{V})[A_i] \in \mathit{fv}(\Psi(\vec{V}))$ for $i \in \{1, \dots, k\}$, and

$$\vec{c}_\Phi = (c_{A_1}, \dots, c_{A_k}) \text{ with } c_{A_i} := \Phi[A_i] \text{ for every } i \in \{1, \dots, k\} \tag{7.86}$$

a constant vector such that c_{A_i} is an instantiation of A_i for every $i \in \{1, \dots, k\}$. It is obviously satisfied that $\Psi(\vec{c}_\Phi) \in \mathit{ex}(\Psi(\vec{V}))$, so we still have to show that $\Phi \models \Psi(\vec{c}_\Phi)$ which is, according to Definition 6.4 and Lemma 6.5, equivalent to

$$\Phi[A] = \Psi(\vec{c}_\Phi)[A] \text{ for every } A \in \mathit{select}(\Psi(\vec{c}_\Phi)). \tag{7.87}$$

Consider an attribute A such that $A \in \mathit{select}(\Psi(\vec{c}_\Phi))$. By construction, this holds only if A is instantiated by a constant or by a free variable in $\Psi(\vec{V})$. We distinguish these two cases:

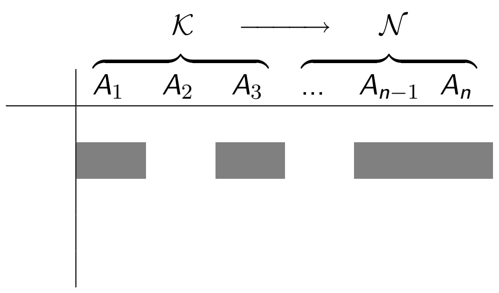
1. $\Psi(\vec{V})[A] \in \mathit{Const}$. In this case, by property 2 a), $\Phi[A] = \Psi(\vec{V})[A]$ and thus also $\Phi[A] = \Psi(\vec{c}_\Phi)[A]$ by (7.83).
2. $\Psi(\vec{V})[A] \in \mathit{fv}(\Psi(\vec{V}))$. Then, by construction of $\vec{c}_\Phi$, $\Phi[A] = c_A = \Psi(\vec{c}_\Phi)[A]$.

Consequently, (7.87) holds and thus property 1 is satisfied. □

Having introduced a "natural" censor and an equivalent censor that is algorithmically implementable, in the following we justify that the induced stateless CQE scqe_{op} (according to Definition 7.40) preserves confidentiality.

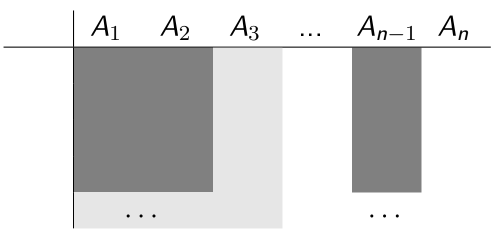
Observe that Definitions 5.3, 5.4 and 6.11 differ from Definitions 7.38–7.40 only in that for closed confidentiality policies the elements of the policy are considered and for open policies the elements of the expansion of the policy are considered. Moreover, by Definition 7.37, the expansion of an open policy yields an infinite set of existential R-sentences, that is, an infinite closed policy. Consequently, $\mathit{censor}^{op}_{\exists R}$ works actually the same as $\mathit{censor}_{\exists R}$ from Definition 6.11, only that it checks elements from an infinite policy rather than from a finite policy. In order to prove scqe_{op} confidentiality-preserving, it therefore suffices to check whether Theorem 7.10 (which considers closed queries and closed secrets from the language $\mathscr{L}_{\exists R}$) also holds if we assume an infinite policy.

The proof of Theorem 7.10, however, does not make use of the finiteness property of the policy and thus applies for infinite sets of existential R-sentences as well. More precisely, already the definition of security (Definition 5.4) indicates that it makes no difference whether a finite or an infinite policy $\mathit{pot\_sec}$ is considered: An alternative instance and an alternative policy must exist for every $\Psi \in \mathit{pot\_sec}$, but for every $\Psi \in \mathit{pot\_sec}$ this instance and policy might be different. Consequently, in the proof of Theorem 7.10, an arbitrary element of the policy is fixed when showing



$\mathcal{K} \longrightarrow \mathcal{N}$
A_1 A_2 A_3 ... A_{n-1} A_n

(a) schema and query restrictions



(b) policy restrictions

Figure 7.8: Visualization of the restrictions for open policies.

"(7.13) ⇒ (7.14)" and thus the argumentation is also valid when assuming an infinite policy.

Summing up, scqe_{op} is secure, since it essentially works the same as scqe. We formally express this insight in the following proposition.

Proposition 7.43 (scqe_{op} preserves confidentiality). *Let $R_S = \langle R, \mathcal{U}, \Sigma \rangle$ be a relation schema in ONF, r an instance of R_S, $\mathit{pot\_sec} \subset \mathscr{L}_{\exists R,fv}$ a confidentiality policy with $\mathit{select}(\Psi(\vec{c})) \in \mathit{bfs}(R_S)$ for every $\Psi(\vec{c}) \in \mathit{ex}(\mathit{pot\_sec})$, and $Q = \langle \Phi_1, \Phi_2, ... \rangle$ a query sequence with $\Phi_1, \Phi_2, ... \in \mathscr{L}_{\exists R}$. Moreover, let $\mathit{log}_0 = \Sigma$ be the a priori user knowledge and assume that $(r, \mathit{pot\_sec})$ is admissible with respect to log_0. Then the stateless CQE $\mathit{scqe}_{op}(Q)(r, \mathit{pot\_sec})$ is secure in the sense of Definition 7.39.*

In Figure 7.8, the restrictions that are demanded by Proposition 7.43 are visualized. The schema restrictions and the query restrictions are the same as in Section 7.1, thus Figure 7.8(a) corresponds to Figure 7.2(a). Regarding the policy restrictions, Proposition 7.43 allows for free variables in potential secrets in order to express (infinite) sets of potential secrets with a single formula. This is indicated by sets of (parts of) tuples in Figure 7.8(b). The dots signify that the expansion of a potential secret with free variables yields an infinite set of potential secrets without free variables.

7.5 Filling the Gaps

In Chapter 6 and Sections 7.1–7.4, we investigated for several combinations of query and policy languages under which restrictions they are suitable for stateless CQE. Although the crucial language combinations have been considered, regarding a systematic investigation, some combinations are still missing. In this section, we identify these gaps and elaborate on how to fill them.

In Subsection 7.5.1, we merge the contributions of the preceding sections and present comprehensive query and policy languages. After that, in Subsection 7.5.2, we investigate the newly introduced comprehensive query language and present a secure stateless CQE for this language. Finally, in Subsection 7.5.3, we consider the policy languages that have been proposed so far and show that they are reducible to existential R-sentences.

7.5.1 Combining Boolean Operations with Free Variables

Having considered Boolean expressions and free variables isolatedly in the preceding sections, we now combine the aspects from Sections 7.2 and 7.3 and the aspects from Sections 7.2 and 7.4, respectively, to obtain open query and policy languages with Boolean operations.

For the policy language, we combine $\mathscr{L}^{\vee}_{\exists R}$ and $\mathscr{L}_{\exists R,fv}$, that is, disjunction and free variables. This combination is a straightforward adaption of Definition 7.18.

Definition 7.44 (Open existential R-sentences with disjunction). *The language $\mathscr{L}^{\vee}_{\exists R,fv}$ is inductively defined as follows:*

- *If* $\chi \in \mathscr{L}_{\exists R,fv}$ *then* $\chi \in \mathscr{L}^{\vee}_{\exists R,fv}$*;*
- *if* $\chi_1, \chi_2 \in \mathscr{L}^{\vee}_{\exists R,fv}$ *then* $\chi_1 \vee \chi_2 \in \mathscr{L}^{\vee}_{\exists R,fv}$*.*

Regarding the query language, the combination of $\mathscr{L}^{\neg,\wedge}_{\exists R}$ and $\mathscr{L}_{\exists R,fv}$ is ambiguous. More precisely, negation together with free variables may lead to problems which is illustrated by the following example.

Example 7.45. Let $R_S = \langle R, A, \emptyset \rangle$ be a relation schema and $r = \{R(a)\}$ an instance of R_S. Moreover, the infinite set of constants is given by $Const = \{a, b, ...\}$. Now consider the query $\Phi(X) \equiv \neg R(X)$ that contains negation on the one hand and is open on the other hand. The evaluation of $\Phi(X)$ according to Definition 2.3 is the infinite set $eval^*(\Phi(X))(r) = \{\neg R(b), \neg R(c), ...\}$. $\Diamond$

Following the notion of [AHV95], Example 7.45 addresses the problem of *unsafe queries*, that is, queries whose evaluation is infinite or, more general, depends on the underlying domain. The authors of [AHV95] propose a modified semantics, the *active domain interpretation*, for handling unsafe queries. Basically, this semantics evaluates a query relative to the active domain $Const' \subset Const$ with $Const'$ denoting the finite set of constants actually occurring in the database instance or in the query. Consequently, the evaluation of a query is always finite. However, the authors admit that the active domain semantics is not feasible from a practical point of view and therefore introduce the notion of *domain independence*: A query is called domain independent if the evaluation is the same on all domains.

As a result, we should aim at admitting only domain independent queries because they can be evaluated in the "natural way" as introduced in Section 2.3. It turns out, however, that the restricted form of negation that may be used according to $\mathscr{L}_{\exists R}^{\neg,\wedge}$ might yield domain dependent queries when being combined with free variables. This is due to the fact that negation may only be used in the form $\neg(\exists X_1)\dots(\exists X_m)R(\dots)$, that is, in combination with atomic existential R-sentences as in Example 7.45. Since a database instance is finite by definition, such a query has potentially an infinite evaluation when assuming an infinite set of constants as domain.

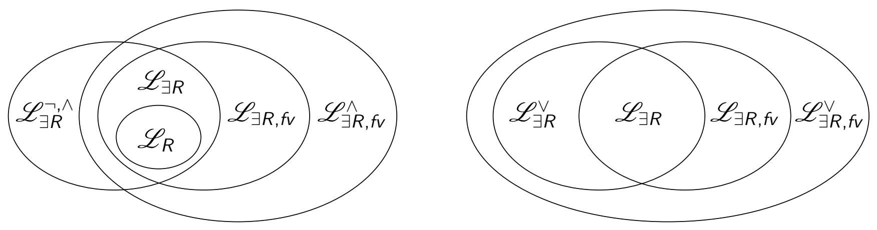
To solve this problem, we could further restrict the use of negation when combining $\mathscr{L}_{\exists R}^{\neg,\wedge}$ with free variables such that $\neg\Phi$ is allowed only if Φ is a positive existential R-sentence without free variables. But in the context of open queries another problem arises: In Subsection 7.2.1, we reduced $\mathit{scqe}^{\neg,\wedge}$ to scqe by splitting up conjunctive queries and answering each conjunct separately. Due to Definition 6.11 and the adapted censor (7.19), scqe behaves the same on a positive query Φ and its negative form $\neg\Phi$. More precisely, the decision whether or not a query is to be refused as well as the ordinary evaluation eval^* is independent of the sign of the query. Considering open queries, however, scqe must be replaced by scqe^{oq}, and now the ordinary evaluation is no longer independent of the sign of the query: If $r \models_M \Phi$ for a (positive) existential R-sentence Φ, then Φ would be answered by $\{\Phi\}$ whereas $\neg\Phi$ would be answered by $\emptyset$. Due to this discrepancy between answering negative existential R-sentences in the context of closed queries and in the context of open queries, the idea of Subsection 7.2.1 cannot be simply adapted for open queries. We therefore refrain from considering negation in the context of open queries but only combine conjunction and free variables in the following.

Definition 7.46 (Open existential R-sentences with conjunction). *The language $\mathscr{L}_{\exists R,fv}^{\wedge}$ is inductively defined as follows:*

- *If* $\chi \in \mathscr{L}_{\exists R,fv}$ *then* $\chi \in \mathscr{L}_{\exists R,fv}^{\wedge}$*;*
- *if* $\chi_1, \chi_2 \in \mathscr{L}_{\exists R,fv}^{\wedge}$ *then* $\chi_1 \wedge \chi_2 \in \mathscr{L}_{\exists R,fv}^{\wedge}$*.*

Remark. In Section 11.3, we will at least sketch how the query language $\mathscr{L}_{\exists R,fv}^{\wedge}$ can be enhanced by a restricted form of negation.

To get an overview of all proposed languages, refer to Figure 7.9 on the following page; it depicts the languages and their inclusion relations in form of Venn diagrams. We started with the query language $\mathscr{L}_R$ (R-sentences) and the policy language $\mathscr{L}_{\exists R}$ (existential R-sentences) in Chapter 6. In Chapter 7, we enhanced these languages step by step until we reached $\mathscr{L}_{\exists R,fv}^{\wedge}$ and $\mathscr{L}_{\exists R,fv}^{\vee}$, respectively. As illustrated by Figure 7.9, the most expressive policy language $\mathscr{L}_{\exists R,fv}^{\vee}$ comprises all previously proposed policy languages (see Figure 7.9(b)). Regarding query languages, however, we decided not to combine negation with free variables; thus, the query language $\mathscr{L}_{\exists R,fv}^{\wedge}$ does not comprise all previously proposed query languages (see Figure 7.9(a)).


$\mathscr{L}^{\neg,\wedge}_{\exists R}$
$\mathscr{L}_{\exists R}$
$\mathscr{L}_{R}$
$\mathscr{L}_{\exists R,fv}$
$\mathscr{L}^{\wedge}_{\exists R,fv}$
$\mathscr{L}^{\vee}_{\exists R}$
$\mathscr{L}_{\exists R}$
$\mathscr{L}_{\exists R,fv}$
$\mathscr{L}^{\vee}_{\exists R,fv}$

(a) query languages (b) policy languages

Figure 7.9: Relations between the investigated languages.

7.5.2 A Comprehensive Query Language

In Subsection 7.2.1, we proposed to split up queries from $\mathscr{L}^{\neg,\wedge}_{\exists R}$ into single conjuncts and to answer these conjuncts separately. As a consequence, if one of the conjuncts implies a potential secret, only this conjunct has to be refused whereas the other conjuncts can still be answered. We now adopt this concept for $\mathscr{L}^{\wedge}_{\exists R,fv}$ and define the corresponding stateless CQE $\mathit{scqe}^{oq,\wedge}$, which is an adaption of scqe^{oq}, as given by Definition 7.26, for open query sequences containing conjunction.

Definition 7.47 (Stateless CQE for queries from $\mathscr{L}^{\wedge}_{\exists \mathbf{R},\mathbf{fv}}$). *Consider*

- *a query sequence $Q = \langle \Phi_1, \Phi_2, ...\rangle$ with $\Phi_1, \Phi_2, ... \in \mathscr{L}^{\wedge}_{\exists R,fv}$, that is,*
  $\Phi_i \equiv \Phi_{i,1}(\vec{V}_{i,1}) \wedge \Phi_{i,2}(\vec{V}_{i,2}) \wedge ... \wedge \Phi_{i,m_i}(\vec{V}_{i,m_i})$
  such that $\Phi_{i,j}(\vec{V}_{i,j}) \in \mathscr{L}_{\exists R,fv}$ for all $\Phi_{i,j}(\vec{V}_{i,j})$,
- *a policy $\mathit{pot\_sec}$ and*
- *an instance r.*

Then, the stateless CQE $\mathit{scqe}^{oq,\wedge}$ is defined as follows:

$$\mathit{scqe}^{oq,\wedge}(Q)(r,\mathit{pot\_sec}) := \mathit{scqe}^{oq}(Q')(r,\mathit{pot\_sec}) \quad \textit{with}$$
$$Q' := \langle \Phi_{1,1}(\vec{V}_{1,1}), \Phi_{1,2}(\vec{V}_{1,2}), \dots, \Phi_{1,m_1}(\vec{V}_{1,m_1}), \Phi_{2,1}(\vec{V}_{2,1}), ...\rangle$$

We now show that $\mathit{scqe}^{oq,\wedge}$ is secure when assuming $\mathscr{L}_{\exists R}$ as policy language (please refer to Subsection 7.5.3 for the investigation of more expressive policy languages). Observe that we have to hide the policy from the user when dealing with open queries, as elaborated in Section 7.3.

Theorem 7.48 ($\mathit{scqe}^{oq,\wedge}$ preserves confidentiality). *Reconsider the assumptions from Theorem 7.10, but let the queries in Q be elements of $\mathscr{L}^{\wedge}_{\exists R,fv}$ and suppose that $\mathit{pot\_sec}$ is unknown to the user. Then $\mathit{scqe}^{oq,\wedge}$ is secure in the sense of Definition 5.4.*

Proof. The claim directly follows from the definition of $scqe^{oq,\wedge}$ (Definition 7.47) and Theorem 7.35. □

7.5.3 Reducing Policy Languages to Existential *R*-Sentences

So far, we mostly assumed $\mathscr{L}_{\exists R}$ or $\mathscr{L}^{\vee}_{\exists R}$ as policy language. Thus, it remains arguable what happens when exchanging this language by a more expressive one. In the following, we roughly outline that in the context of stateless CQE all proposed policy languages are reducible to $\mathscr{L}_{\exists R}$ and therefore the results for $\mathscr{L}_{\exists R}$ also apply for the other languages.

Regarding open confidentiality policies, observe that a potential secret that contains free variables is only a short notation for its expansion, as already defined in Subsection 7.4.1. More precisely, an open policy $pot\_sec \subset \mathscr{L}_{\exists R,fv}$ is supposed to have the same semantics as its expansion $ex(pot\_sec) \subset \mathscr{L}_{\exists R}$. Therefore, when reasoning about open potential secrets, it is obligatory to previously expand them. Consequently, each open policy may (at least conceptually) be transformed into an equivalent closed policy.

Regarding disjunctive policies, in the context of Theorem 7.22 it has been proven that for a query $\Phi \in \mathscr{L}_{\exists R}$ and a disjunctive secret $\Psi \equiv \Psi_1 \vee \Psi_2 \vee ... \vee \Psi_m \in \mathscr{L}^{\vee}_{\exists R}$ it holds that

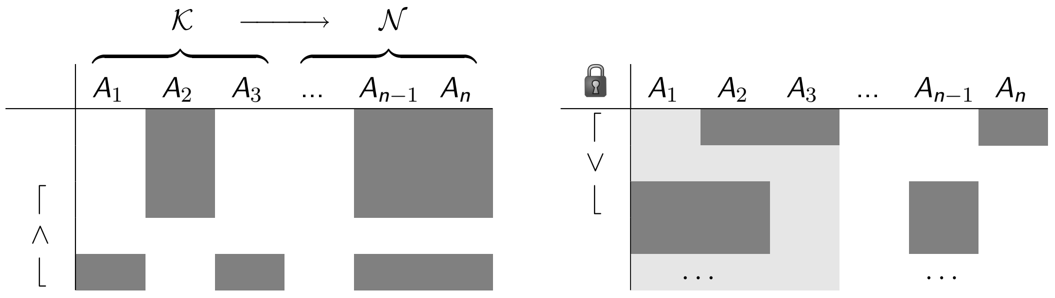
$$\begin{aligned}&\Phi \models \Psi_1 \vee \Psi_2 \vee ... \vee \Psi_m \quad \text{if and only if} \\ &\Phi \models \Psi_1 \text{ or } \Phi \models \Psi_2 \text{ or } ... \text{ or } \Phi \models \Psi_m.\end{aligned} \tag{7.88}$$

Note that all yet defined censors base their decision on whether or not an existential R-sentence Φ logically implies at least one of the declared (non-disjunctive) potential secrets, see Definitions 6.11, 7.26, 7.40, and 7.41 and the definition of $censor^{\neg}_{\exists R}$ in (7.19). According to (7.88), for potential secrets from $\mathscr{L}_{\exists R}$, this is equivalent to Φ logically implying the disjunction of the potential secrets. It follows that each disjunctive policy from $\mathscr{L}^{\vee}_{\exists R}$ may be transformed into an equivalent policy from $\mathscr{L}_{\exists R}$ by splitting up each disjunctive secret into atomic secrets.

In the case of $\mathscr{L}^{\vee}_{\exists R,fv}$, which allows to declare potential secrets that contain free variables as well as disjunction, the above argumentation can be applied as follows: An open disjunctive potential secret of the form $\Psi_1(\vec{V}_1) \vee ... \vee \Psi_m(\vec{V}_m) \in \mathscr{L}^{\vee}_{\exists R,fv}$ is first transformed into $\{\Psi_1(\vec{V}_1), ..., \Psi_m(\vec{V}_m)\}$ in order to eliminate disjunction. After that, this set is expanded to $\{ex(\Psi_1(\vec{V}_1)), ..., ex(\Psi_m(\vec{V}_m))\}$, a set that is (in the context of stateless CQE) equivalent to $\Psi_1(\vec{V}_1) \vee ... \vee \Psi_m(\vec{V}_m)$.

Summing up, all results that we found for the policy language $\mathscr{L}_{\exists R}$ also apply for the policy languages $\mathscr{L}^{\vee}_{\exists R}$, $\mathscr{L}_{\exists R,fv}$ and $\mathscr{L}^{\vee}_{\exists R,fv}$ that emerge from $\mathscr{L}_{\exists R}$ by adding disjunction and/or free variables.

Figure 7.10 on the next page visualizes schema restrictions, query language restrictions and policy language restrictions when assuming $\mathscr{L}^{\wedge}_{\exists R,fv}$ as query language


$\mathcal{K} \longrightarrow \mathcal{N}$
A_1 A_2 A_3 ... A_{n-1} A_n
$\wedge$
$\vee$
...
...

(a) schema and query restrictions (b) policy restrictions

Figure 7.10: Visualization of the restrictions for comprehensive languages.

and $\mathscr{L}^{\vee}_{\exists R,fv}$ as policy language. Regarding Figure 7.10(a), the "FD pattern" $\mathcal{K} \rightarrow \mathcal{N}$ represents the ONF restriction, the interrupted table row represents an existential R-sentence, and the gray blocks in the upper part of the table represents open queries, that is, queries that are answered by sets of (parts of) tuples. Moreover, queries may be connected by conjunction which is indicated by the $\wedge$-symbol. Regarding Figure 7.10(b), the policy elements are restricted to existential R-sentences (represented by the interrupted table row), existential R-sentences enhanced by free variables (represented by the gray blocks in the lower part of the table), and disjunctions of these elements (represented by the $\vee$-symbol). Furthermore, as indicated by the padlock icon, the policy is supposed to be unknown to the user.

7.6 Summary

In this chapter, we investigated several language enhancements of the "basic case" from Chapter 6. We started with existential R-sentences as query language, which enabled us to express select-project-queries. It turned out that, regarding preservation of confidentiality, the transition from select- to select-project-queries is non-trivial under the presence of functional dependencies. More precisely, we have to restrict the confidentiality policy to basic facts in order to keep up confidentiality.

Moreover, we investigated Boolean operations in query and policy languages. Since stateless CQE requires that no "sophisticated" inferences can be drawn, we have to restrict the use of Boolean operations in order to prevent implicative structures. In fact, queries may be conjunctive but not disjunctive, whereas potential secrets may be disjunctive but not conjunctive.

Another enhancement that we considered is the introduction of free variables. Regarding the query language, free variables allow for open queries, that is, queries that are answered by a set of tuples rather than by a truth value. We observed that the assumption that the user knows the confidentiality policy cannot be kept up when he is allowed to use free variables in his queries. Regarding the policy language, free variables allow for the declaration of “secret schemas”. More precisely, a free variable in a potential secret is (at least conceptually) replaced with every possible constant when checking if a query implies this secret. In this way, multiple secrets can be declared with a single formula.

Finally, we combined our results in order to develop the comprehensive query language $\mathscr{L}^{\wedge}_{\exists R,fv}$ and the comprehensive policy language $\mathscr{L}^{\vee}_{\exists R,fv}$. Regarding the query language, it turned out that the combination of free variables with negation cannot be treated straightforwardly. This is, on the one hand, due to the problem of unsafe queries, and, on the other hand, due to the different evaluation of closed and open queries. We therefore refrained from integrating negation into the comprehensive query language. In Section 11.3, however, we sketch some ideas regarding negation in open queries.

Figure 7.11 on the following page summarizes our achievements with a tabular overview of all query/policy language combinations. For each of these combinations, the identified restrictions and the reference within this thesis are given.

It has been shown in Chapter 6 and Sections 7.1–7.5 that the restrictions which are mentioned in Figure 7.11 are sufficient to guarantee the preservation of confidentiality in the sense of Definitions 5.4 and 7.39. Although we did not explicitly show that these restrictions are also necessary for the preservation of confidentiality, we provided some examples which indicate that relaxations of these constraints might easily compromise confidentiality. We shortly recall the consequences of these examples:

- Example 7.4 justifies the basic facts restriction when using existential R-sentences in the query language.
- Example 7.11 shows that disjunction in queries possibly allows for harmful inferences; consequently, the use of conjunction and negation must also be restricted in order to prevent the simulation of disjunction.
- Example 7.16 illustrates that negation in potential secrets can be harmful.
- Example 7.17 illustrates that conjunction in potential secrets can be harmful.
- Example 7.27 justifies why the policy must be unknown to the user under open queries.
- Example 7.45 together with the following elaborations show that the use of negation together with open queries may lead to unsafe queries.

Certainly, parts of our achievements might still be enhanceable for some special cases of query sequences or policies. We believe, however, that the table in Figure 7.11 basically covers the space of possibilities for stateless CQE in relational databases.

query language	**policy language**			
	$\mathscr{L}_{\exists R}$	$\mathscr{L}_{\exists R}^{\vee}$	$\mathscr{L}_{\exists R,fv}$	$\mathscr{L}_{\exists R,fv}^{\vee}$
$\mathscr{L}_{R}$	no restrictions (Chapter 6)	no restrictions (Section 7.5)	no restrictions (Section 7.5)	no restrictions (Section 7.5)
$\mathscr{L}_{\exists R}$	ONF, basic facts (Section 7.1)	ONF, basic facts (Section 7.2)	ONF, basic facts (Section 7.4)	ONF, basic facts (Section 7.5)
$\mathscr{L}_{\exists R}^{\neg,\wedge}$	ONF, basic facts (Section 7.2)	ONF, basic facts (Section 7.2)	ONF, basic facts (Section 7.5)	ONF, basic facts (Section 7.5)
$\mathscr{L}_{\exists R,fv}$	ONF, basic facts unknown policy (Section 7.3)	ONF, basic facts unknown policy (Section 7.5)	ONF, basic facts unknown policy (Section 7.5)	ONF, basic facts unknown policy (Section 7.5)
$\mathscr{L}_{\exists R,fv}^{\wedge}$	ONF, basic facts unknown policy (Section 7.5)	ONF, basic facts unknown policy (Section 7.5)	ONF, basic facts unknown policy (Section 7.5)	ONF, basic facts unknown policy (Section 7.5)

Figure 7.11: Investigated combinations of query and policy languages.

Part III

A Stateless Inference Control System

Chapter 8

SQL Representation of Queries and Policies

In this and the subsequent chapter, we aim at developing a computationally efficient algorithmic implementation of the concepts that have been investigated in Part II. In doing so, we show that stateless CQE could actually be installed as a mediating layer between a database management system and the users of this system with the objective to preserve confidentiality in terms of the declared confidentiality policies. Of course, for stateless CQE to be successfully deployed in an environment with sensitive data, a more thorough investigation and protection of this environment would be necessary; for example, it must be guaranteed that the stateless CQE mechanism cannot be bypassed when accessing the database and that there are no data leaks due to covert channels. In this thesis, however, we abstract from these issues and focus on the implementation of the core of stateless CQE.

In the following Sections 8.1 and 8.2, we prepare the algorithmic implementation of stateless CQE by elaborating the representation of the elements of query and policy languages from Part II by means of the database language SQL [SQL08]. More precisely, we take the SQL implementation of the Oracle database system [LR$^+$10] as a basis for our investigations.

8.1 Queries

When translating queries of the relational calculus into SQL, we must pay attention to the different semantics: While relational calculus queries are evaluated under set semantics, that is, duplicates are removed from the answers, SQL queries are usually evaluated using bag semantics, that is, the answers may contain duplicates. SQL, however, provides the keyword `DISTINCT` in order to remove duplicates from answer sets. Consequently, we simulate set semantics in the following by confining ourselves to `SELECT DISTINCT` queries.

In Figure 8.1 on the next page, an SQL fragment which is able to express the elements of the query languages from Part II is depicted in form of syntax diagrams. These diagrams, which follow the notation of [LR$^+$10], represent a formal grammar

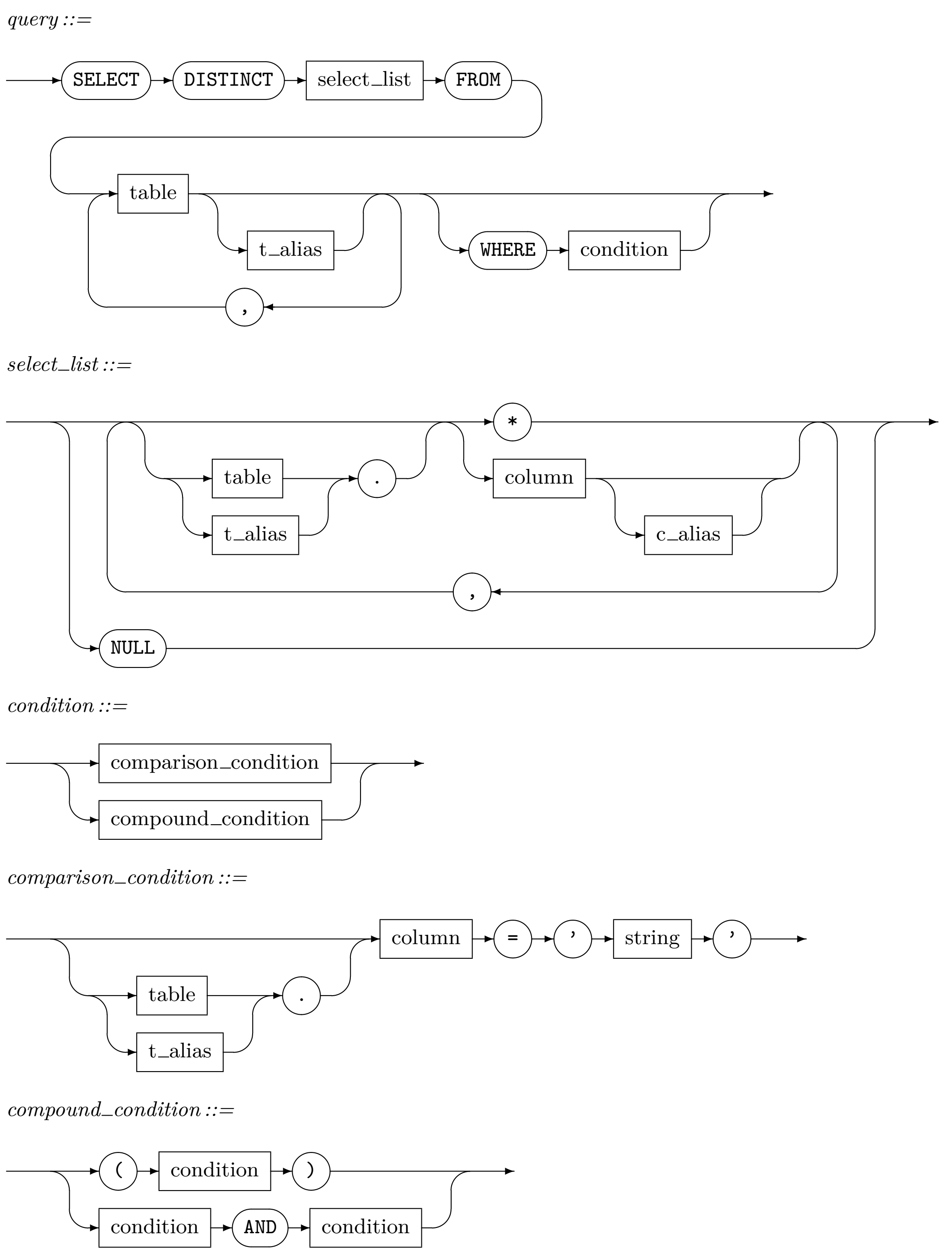


Figure 8.1: Considered SQL fragment.

with the start symbol *query*. For conciseness, the rules for replacing nonterminals with strings are omitted (this concerns the nonterminals *table, t_alias, column, c_alias* and *string*). Basically, the described fragment comprises simple SQL queries without joins[20], subqueries and aggregate functions. More precisely, the fragment allows to select the values of one or more attributes (which are called *columns* in Figure 8.1) from the tuples of a relation (which is called *table* in Figure 8.1) that satisfy a conjunctive combination of comparison conditions. These comparison conditions are equality checks between attribute values and strings. Moreover, it is possible to specify alias names for attributes (*c_alias*) and relations (*t_alias*), and to select the special value `NULL`, see Subsection 8.1.2.

In the following subsections, we show how elements of the languages from Part II can be translated into elements of this SQL fragment. We illustrate these translations with examples assuming the underlying relation schema $R_S = \langle R, \mathcal{U}, \emptyset \rangle$ with the attribute set $\mathcal{U} = \{A, B, C\}$. Moreover, we evaluate the example queries with respect to the following instance r_{ex}:

r_{ex}	A	B	C
	a	b	c
	d	e	f
	a	c	d
	c	b	a

8.1.1 R-Sentences

Interpreted as query, an R-sentence $\chi_R \in \mathscr{L}_R$ asks whether or not the whole tuple represented by χ_R is part of the database instance. To express such a query in SQL, the following constraints must apply regarding the SQL fragment of Figure 8.1:

- The *select_list* is $*$ or a list of all attributes in $\mathcal{U}$.
- The *condition* is a conjunction of comparison conditions such that for each attribute $A \in \mathcal{U}$ it is checked whether A is instantiated with $\chi_R[A]$.

Since we assume that queries always begin with `SELECT DISTINCT`, the answer to such a query is either the single tuple that has been asked for or the empty relation; in terms of the ordinary query evaluation $eval$ according to Definition 2.2, these answers can be interpreted as `true` or `false`, respectively.[21]

[20] Although the *select_list* allows to specify more than one relation, we will not use this feature to express general joins (which are outside the scope of this thesis; see Section 11.2 for an outlook on how to deal with join queries) but only to multiply specify the same relation for a query which technically corresponds to a cross join of the relation with itself. See Subsection 8.1.3 for details.

[21] We refer to $eval$ rather than to $eval^*$, since from the perspective of the database user it seems more natural to answer a closed query Φ with `true` or `false` rather than with Φ or $\neg\Phi$, respectively.

Example 8.1. The query $\Phi \equiv R(a,b,c)$ ($\in \mathscr{L}_R$) is expressed by the following SQL statement:

```
SELECT DISTINCT * FROM R
WHERE A='a' AND B='b' AND C='c'
```

The evaluation of this query with respect to r_{ex} yields the following result:

A	B	C
a	b	c

Since this result is not the empty relation, the query Φ is true in r_{ex}. ◊

8.1.2 Existential R-Sentences

Interpreted as query, an existential R-sentence $\chi_{\exists R} \in \mathscr{L}_{\exists R}$ asks whether or not the partial tuple represented by $\chi_{\exists R}$ is part of the database instance. To express such a query in SQL, the following constraints must apply regarding the SQL fragment of Figure 8.1:

- The *select_list* is a subset of the attributes in $\mathcal{U}$; more precisely, it contains exactly those attributes that are instantiated with a constant in $\chi_{\exists R}$.
- The *condition* is a conjunction of comparison conditions such that for each attribute A occurring in the *select_list* it is checked whether A is instantiated with $\chi_{\exists R}[A]$.

As with R-sentences, the answer to such a query is either the single partial tuple that has been asked for (corresponding to `true` in the relational calculus) or the empty relation (corresponding to `false` in the relational calculus).

Example 8.2. The query $\Phi \equiv (\exists X_B)R(a,X_B,b)$ ($\in \mathscr{L}_{\exists R}$) is expressed by the following SQL statement:

```
SELECT DISTINCT A, C FROM R
WHERE A='a' AND C='b'
```

The evaluation of this query with respect to r_{ex} yields the following result:

A	C

This result is the empty relation; thus, Φ is false in r_{ex}. ◊

A special case of an existential R-sentence is the query that asks for the existence of an arbitrary tuple, for example, $\Phi_\exists \equiv (\exists X_A)(\exists X_B)(\exists X_C)R(X_A,X_B,X_C)$ for the ternary relation R. This query would cause an empty *select_list* which is not allowed according to Figure 8.1. Therefore, the special value `NULL` may be specified as *select_list* which leads to the following translation of the query $\Phi_\exists$:

```
SELECT DISTINCT NULL FROM R
```

This query is evaluated either to the empty tuple which signifies that there exists a tuple in R and thus corresponds to `true` in the relational calculus, or to the empty relation which signifies that R is empty and thus corresponds to `false` in the relational calculus.

8.1.3 Existential R-Sentences with Conjunction

Negation, as used in the languages $\mathscr{L}^{\neg}_{\exists R}$ and $\mathscr{L}^{\neg,\wedge}_{\exists R}$ (see Section 7.2), cannot be straightforwardly translated to SQL. This is due to the fact that general negative queries may be unsafe, that is, their evaluation may be an infinite set. We therefore only consider conjunctive queries without negation hereafter and denote the pertinent fragment of $\mathscr{L}^{\neg,\wedge}_{\exists R}$ with $\mathscr{L}^{\wedge}_{\exists R}$. This restriction, however, does not decrease the expressiveness of the query language because from the definition of the ordinary query evaluation (Definition 2.2) it follows that $eval(\neg\Phi)(r) \equiv \neg eval(\Phi)(r)$. Consequently, the answer to $\neg\Phi$ can also be obtained by asking Φ and then negating the answer.

Interpreted as query, a conjunction of existential R-sentences $\chi^{\wedge}_{\exists R} \in \mathscr{L}^{\wedge}_{\exists R}$ with $\chi^{\wedge}_{\exists R} \equiv \chi^{\wedge}_{\exists R,1} \wedge \chi^{\wedge}_{\exists R,2} \wedge \dots \wedge \chi^{\wedge}_{\exists R,m}$ (such that $\chi^{\wedge}_{\exists R,i} \in \mathscr{L}_{\exists R}$ for all i) asks whether or not each conjunct $\chi^{\wedge}_{\exists R,i}$ of $\chi^{\wedge}_{\exists R}$ (interpreted as partial tuple) is part of the database instance. To express such a query in SQL, the following constraints must apply regarding the SQL fragment of Figure 8.1:

- The *select_list* contains for every conjunct $\chi^{\wedge}_{\exists R,i}$ of the query $\chi^{\wedge}_{\exists R}$ the attributes that are instantiated with constants in $\chi^{\wedge}_{\exists R,i}$; moreover, each attribute is qualified with the relation symbol and the number of the conjunct it occurs in (for example, $Ri.A$ if A is instantiated with a constant in $\chi^{\wedge}_{\exists R,i}$).

- The *condition* is a conjunction of comparison conditions such that for each attribute $Ri.A$ occurring in the *select_list*, qualified with relation symbol and number of the conjunct, it is checked whether $Ri.A$ is instantiated with $\chi^{\wedge}_{\exists R,i}[A]$.

The answer to such a query is either a single tuple which equals to the concatenation of all (partial) tuples being asked for in the conjuncts of $\chi^{\wedge}_{\exists R}$, or it is the empty relation. The former case corresponds to the answer `true` in the relational calculus, and the latter case corresponds to the answer `false` in the relational calculus.

Example 8.3. The query $\Phi \equiv R(a,b,c) \wedge (\exists X_C)R(d,e,X_C)$ ($\in \mathscr{L}^{\wedge}_{\exists R}$) is expressed by the following SQL statement:

```
SELECT DISTINCT R1.A, R1.B, R1.C, R2.A, R2.B
FROM R R1, R R2
WHERE R1.A='a' AND R1.B='b' AND R1.C='c' AND
      R2.A='d' AND R2.B='e'
```

The evaluation of this query with respect to r_{ex} yields the following result:

A	B	C	A	B
a	b	c	d	e

Consequently, Φ is true in r_{ex}. ◊

If a conjunct $\chi_{\exists}$ of a relational calculus query $\Phi \in \mathscr{L}^{\wedge}_{\exists R}$ asks for the existence of an arbitrary tuple (like the query $\Phi_{\exists}$ in Subsection 8.1.2), this conjunct does not need to be explicitly represented in the SQL translation of the query. This is justified as follows: On the one hand, if the underlying relation R is empty, $\chi_{\exists}$ evaluates to `false` and consequently also Φ evaluates to `false` in the relational calculus. The evaluation of the SQL representation of Φ, however, is the empty relation even if $\chi_{\exists}$ is not represented, since R is empty and consequently every conjunct $\chi \not\equiv \chi_{\exists}$ of Φ yields the empty relation. On the other hand, if there is at least one tuple in R, $\chi_{\exists}$ evaluates to `true` in the relational calculus and consequently the answer to Φ only depends on the other conjuncts $\chi \not\equiv \chi_{\exists}$.

8.1.4 Open Queries

Syntactically, a formula $\chi_{\exists R,fv} \in \mathscr{L}_{\exists R,fv}$ is an existential R-sentence that additionally contains (one or more) free variables. The answer to $\chi_{\exists R,fv}$ (interpreted as query) is the set of all assignments of these free variables that make $\chi_{\exists R,fv}$ part of the database instance. To express such a query in SQL, the following constraints must apply regarding the SQL fragment of Figure 8.1:

- The *select_list* is a subset of the attributes in $\mathcal{U}$; more precisely, it contains exactly those attributes that are instantiated with a constant or with a free variable in $\chi_{\exists R,fv}$.
- The *condition* is a conjunction of comparison conditions such that for each attribute A occurring in the *select_list* and with $\chi_{\exists R,fv}[A]$ being a constant it is checked whether A is instantiated with $\chi_{\exists R,fv}[A]$.

Observe that these constraints are similar to those for existential R-sentences (Subsection 8.1.2). However, for an open query $\chi_{\exists R,fv}$, the *select_list* additionally contains those attributes that are instantiated with free variables in $\chi_{\exists R,fv}$. These attributes do not occur in the *condition* and consequently, the answer to the SQL query is the set of all value combinations for these attributes such that the resulting tuples are part of the database instance.

We also include those attributes in the *select_list* that are instantiated with a constant in $\chi_{\exists R,fv}$; this is for the following reason: Consider the special case of an open query with zero free variables. If only those attributes were included in the *select_list* that are instantiated with a free variable, the *select_list* would be empty which is not allowed according to the syntax diagrams in Figure 8.1. Analogous to existential R-sentences (see Subsection 8.1.2), if the query contains neither free variables nor constants, the *select_list* is the special value `NULL`.

In order to easily distinguish between attributes instantiated with a constant and attributes instantiated with a free variable in $\chi_{\exists R,fv}$, we suggest to rename the attribute identifiers in the SQL query appropriately: Attribute identifiers that represent a free variable are extended by the prefix `F_`, as illustrated by the following example.

Example 8.4. The query $\Phi(X_B) \equiv (\exists X_C) R(a,X_B,X_C)$ $(\in \mathscr{L}_{\exists R,fv})$ is expressed by the following SQL statement:

```
SELECT DISTINCT A, B F_B FROM R
WHERE A='a'
```

The evaluation of this query with respect to r_{ex} yields the following result:

A	F_B
a	b
a	c

Consequently, $\Phi(X_B)$ is true in r_{ex} for $X_B = b$ and for $X_B = c$. ♢

8.1.5 Open Queries with Conjunction

The language $\mathscr{L}^{\wedge}_{\exists R,fv}$ combines free variables with conjunction. Therefore, the SQL representation of a formula $\chi^{\wedge}_{\exists R,fv} \in \mathscr{L}^{\wedge}_{\exists R,fv}$, interpreted as a query, is a straightforward combination of the representation of queries from $\mathscr{L}^{\wedge}_{\exists R}$ and $\mathscr{L}_{\exists R,fv}$. The following constraints must apply regarding the SQL fragment of Figure 8.1:

- The *select_list* contains for every conjunct $\chi^{\wedge}_{\exists R,fv,i}$ of the query $\chi^{\wedge}_{\exists R,fv}$ the attributes that are instantiated with a constant or with a free variable in $\chi^{\wedge}_{\exists R,fv,i}$; moreover, each attribute in the *select_list* is qualified with the relation symbol and the number of the conjunct it occurs in.

- The *condition* is a conjunction of comparison conditions such that for each attribute $Ri.A$ occurring in the *select_list* with $\chi^{\wedge}_{\exists R,fv,i}[A]$ being a constant it is checked whether A is instantiated with $\chi^{\wedge}_{\exists R,fv,i}[A]$.

Each conjunct of $\chi^{\wedge}_{\exists R,fv}$ could be answered isolatedly as described in Subsection 8.1.4, yielding a set of tuples for each conjunct. The answer to the SQL representation of $\chi^{\wedge}_{\exists R,fv}$ is the Cartesian product of these sets. Consequently, each tuple in this answer set represents one attribute value combination for which each conjunct of $\chi^{\wedge}_{\exists R,fv}$ is part of the database instance.

We again suggest to rename the attribute identifiers that represent a free variable in the SQL query in order to easily distinguish between attributes instantiated with a constant and attributes instantiated with a free variable in $\chi^{\wedge}_{\exists R,fv}$. Consider the following example.

Example 8.5. The query $\Phi(X_{A_1},X_{C_1},X_{B_3},X_{C_3}) \equiv R(X_{A_1},b,X_{C_1}) \wedge R(a,b,c) \wedge (\exists X_{A_3})R(X_{A_3},X_{B_3},X_{C_3})$ $(\in \mathscr{L}^{\wedge}_{\exists R,fv})$ is expressed by the following SQL statement:

```
SELECT DISTINCT R1.A F_A1, R1.B, R1.C F_C1,
                R2.A, R2.B, R2.C,
                R3.B F_B3, R3.C F_C3
FROM R R1, R R2, R R3
WHERE R1.B='b' AND
      R2.A='a' AND R2.B='b' AND R2.C='c'
```

The evaluation of this query with respect to r_{ex} yields the following result:

F_A1	B	F_C1	A	B	C	F_B3	F_C3
a	b	c	a	b	c	b	c
a	b	c	a	b	c	e	f
a	b	c	a	b	c	c	d
a	b	c	a	b	c	b	a
c	b	a	a	b	c	b	c
c	b	a	a	b	c	e	f
c	b	a	a	b	c	c	d
c	b	a	a	b	c	b	a

Consequently, $\Phi(X_{A_1},X_{C_1},X_{B_3},X_{C_3})$ is true in r_{ex} for the following assignments of the free variables: (a,c,b,c), (a,c,e,f), (a,c,c,d), (a,c,b,a), (c,a,b,c), (c,a,e,f), (c,a,c,d), and (c,a,b,a). ◊

8.2 Policies

As already pointed out in Section 6.4, it is reasonable to store the elements of a confidentiality policy in form of a classification instance $r^{pot-sec}$. This classification instance has the (classification) schema $\langle R^{pot-sec}, \mathcal{U}, \emptyset\rangle$, that is, it shares the attribute set with the database relation R and has no semantic constraints. In the following, we show how the notion of classification instance from Definition 6.15 can be adapted to the policy language $\mathscr{L}^{\vee}_{\exists R,fv}$; more precisely, we propose a tuple representation for the elements of $\mathscr{L}^{\vee}_{\exists R,fv}$. For the other policy languages that have been investigated in Part II, it holds that $\mathscr{L}_{\exists R} \subset \mathscr{L}^{\vee}_{\exists R,fv}$, $\mathscr{L}^{\vee}_{\exists R} \subset \mathscr{L}^{\vee}_{\exists R,fv}$ and $\mathscr{L}_{\exists R,fv} \subset \mathscr{L}^{\vee}_{\exists R,fv}$. Consequently, elements from these languages can be seen as special cases of $\mathscr{L}^{\vee}_{\exists R,fv}$ and their representation in $R^{pot-sec}$ does not need to be considered separately.

Given a (general) potential secret $\Psi \in \mathscr{L}^{\vee}_{\exists R,fv}$ with $\Psi \equiv \Psi_1 \vee \Psi_2 \vee \dots \vee \Psi_m$ and $\Psi_i \in \mathscr{L}_{\exists R,fv}$ for all $i \in \{1, \dots, m\}$, we consider each disjunct Ψ_i of Ψ separately as a generalized tuple.[22] Each of these generalized tuples is then transformed into a tuple μ_i according to the following rules:

- If $\Psi_i[A] \in \mathit{Const}$ for an attribute A, then $\mu_i[A] := \Psi_i[A]$.
- If $\Psi_i[A] \in \mathit{Var}$ for an attribute A and $\Psi_i[A]$ occurs freely in Ψ_i, then $\mu_i[A] := \sim$.
- If $\Psi_i[A] \in \mathit{Var}$ for an attribute A and $\Psi_i[A]$ does not occur freely in Ψ_i, then $\mu_i[A] := \#$.

We assume that $\sim$ and $\#$ are "new" constant symbols, that is, $\sim \notin \mathit{Const}$ and $\# \notin \mathit{Const}$.

A policy $\mathit{pot\_sec} \subset \mathscr{L}^{\vee}_{\exists R,fv}$ is then represented as a classification instance by transforming each disjunct of each $\Psi \in \mathit{pot\_sec}$ into a tuple according to the above rules. On the SQL level, each of these tuples is added to the classification instance by means of the `INSERT INTO` command. We illustrate the representation of a policy by a classification instance with the following example.

Example 8.6. Consider the policy $\mathit{pot\_sec} = \{\Psi_1, \Psi_2\} \subset \mathscr{L}^{\vee}_{\exists R,fv}$ with

$$\begin{aligned}\Psi_1 &\equiv (\exists X_B)(\exists X_C)R(a,X_B,X_C) \quad \text{and}\\ \Psi_2 &\equiv R(b,c,d) \vee (\exists X_B)R(X_A,X_B,e).\end{aligned}$$

To represent $\mathit{pot\_sec}$ as classification instance, we first split Ψ_2 into its disjuncts:

$$\begin{aligned}\Psi_{2,1} &\equiv R(b,c,d)\\ \Psi_{2,2} &\equiv (\exists X_B)R(X_A,X_B,e)\end{aligned}$$

Then, Ψ_1, $\Psi_{2,1}$ and $\Psi_{2,2}$ are transformed into the tuples μ_1, $\mu_{2,1}$ and $\mu_{2,2}$ of the classification instance $r^{pot-sec}$ according to the given rules:

$$\begin{aligned}\mu_1 \quad &:\equiv \quad R^{pot-sec}(a,\#,\#)\\ \mu_{2,1} \quad &:\equiv \quad R^{pot-sec}(b,c,d)\\ \mu_{2,2} \quad &:\equiv \quad R^{pot-sec}(\sim,\#,e)\end{aligned}$$

[22] According to the proof of Theorem 7.22, stateless CQE behaves alike for the policies $\mathit{pot\_sec} = \{\Psi_1 \vee \Psi_2 \vee \dots \vee \Psi_m\}$ and $\mathit{pot\_sec} = \{\Psi_1, \Psi_2, \dots, \Psi_m\}$.

Finally, these tuples are inserted into the classification instance with the following SQL commands:

```
INSERT INTO R_pot_sec VALUES ('a', '#', '#')
INSERT INTO R_pot_sec VALUES ('b', 'c', 'd')
INSERT INTO R_pot_sec VALUES ('~', '#', 'e')
```
◊

Chapter 9

Implementation of Stateless CQE

In this chapter, we develop efficient algorithms for the stateless CQE mechanisms from Part II. Relying on the SQL representations of the query and policy languages from Chapter 8, these algorithms lay the foundation for the implementation of effective and efficient inference control in database management systems.

In Section 9.1, we propose implementations for the censor functions $\mathit{censor}_{\exists R}$ and $\widetilde{\mathit{censor}}^{op}_{\exists R}$ which are the core of the investigated stateless CQE mechanisms. Based on these implementations we then sketch two algorithms for stateless CQE in Section 9.2. The first algorithm is designed for use in a closed queries scenario, whereas the second algorithm is able to deal with query sequences that contain both closed and open queries. Finally, we analyze the complexity of the proposed algorithms.

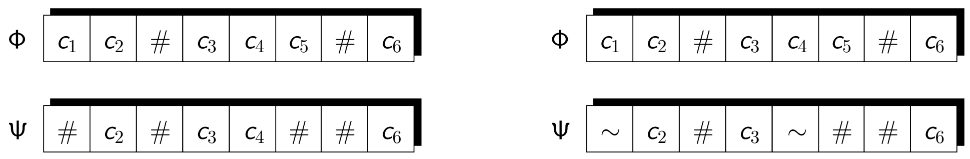
9.1 Implementation of the Censors

The stateless CQE mechanisms for closed queries that have been developed in Part II rely either on $\mathit{censor}_{\exists R}$ or on $\widetilde{\mathit{censor}}^{op}_{\exists R}$ (the censor $\mathit{censor}^{\neg}_{\exists R}$ is not considered in the following, since we do not allow negation in queries, as justified in Subsection 8.1.3). The controlled evaluation of open queries uses a *refused*-set rather than a censor, see Definition 7.26. As we shall see at the end of this section, however, the computation of (the important part of) *refused* can be performed using the censor functions for closed queries. We therefore develop SQL implementations for the censor functions $\mathit{censor}_{\exists R}$ and $\widetilde{\mathit{censor}}^{op}_{\exists R}$ in the following.

We hereafter suppose that a confidentiality policy has been set up by a security administrator in form of a classification instance $r^{pot-sec}$ as described in Section 8.2. Then, the implementation of a censor function has the following general form:

```
SELECT COUNT(*) FROM R_pot_sec
WHERE <condition_const>
AND <condition_var>
```

Using the aggregate function `COUNT`, this SQL statement returns the number of tuples in the classification instance that comply with the conditions `<condition_const>`


Φ
c_1 c_2 # c_3 c_4 c_5 # c_6
Ψ
# c_2 # c_3 c_4 # # c_6
Φ
c_1 c_2 # c_3 c_4 c_5 # c_6
Ψ
~ c_2 # c_3 ~ # # c_6

(a) $\Psi \in \mathscr{L}_{\exists R}$ (b) $\Psi \in \mathscr{L}_{\exists R, fv}$

Figure 9.1: Illustration of relevance.

and `<condition_var>`. These conditions will be designed with respect to the user query under consideration in such a way that

- the return value 0 corresponds to the censor decision `false`, that is, the user query under consideration may be answered honestly; and
- a return value ≥ 1 corresponds to the censor decision `true`, that is, the answer to the user query under consideration must be modified in order to preserve confidentiality.

Firstly, we consider the censor function $censor_{\exists R}$ (see Definition 6.11) which is used for closed queries and potential secrets without free variables. We assume the policy language $\mathscr{L}_{\exists R}$ here, recalling that disjunctive secrets from the language $\mathscr{L}^{\vee}_{\exists R}$ are split into the single disjuncts when inserted into a classification instance (see Section 8.2). The censor $censor_{\exists R}$ checks whether or not $\Phi \models \Psi$ for a user query $\Phi \in \mathscr{L}_{\exists R}$[23] and a potential secret Ψ. According to Definition 6.4 and Lemma 6.5, this condition is equivalent to Ψ being relevant for Φ, that is, $\Phi[A] = \Psi[A]$ for all attributes $A \in select(\Psi)$. We (exemplarily) visualize the concept of relevance in Figure 9.1(a): The attribute values of the query Φ and the potential secret Ψ are represented by boxes, either filled with a constant c_i or the symbol #, representing an existentially quantified variable. Using this representation, Ψ is relevant for Φ if and only if for each attribute A it holds that $\Phi[A] = \Psi[A]$ or $\Psi[A] = \#$.

We now use this characterization of relevance for specifying the conditions for the SQL implementation of $censor_{\exists R}$.

Definition 9.1 (SQL implementation of $censor_{\exists R}$). *Given a query Φ with $A_1, A_2, \ldots, A_{m_1}$ denoting the attributes that are instantiated with constants in Φ such that $\Phi[A_i] = a_i$, and $B_1, \ldots, B_{m_2}$ denoting the attributes that are instantiated with variables in Φ, the conditions `<condition_const>` and `<condition_var>` (regarding the general SQL implementation of censors on page 137) are defined as follows:*

[23]Remember that, according to Definition 7.14, queries with conjunction are split into the single conjuncts which are elements of the language $\mathscr{L}_{\exists R}$ before being forwarded to the censor.

```
<condition_const> := (A_1 = 'a_1' OR A_1 = '#') AND
                     (A_2 = 'a_2' OR A_2 = '#') AND ... AND
                     (A_m1 = 'a_m1' OR A_m1 = '#')
<condition_var> := B_1 = '#' AND B_2 = '#' AND ... AND B_m2 = '#'
```

We illustrate the SQL implementation of the censor function $censor_{\exists R}$ with the following example.

Example 9.2. Let a policy be given by $pot\_sec = \{\Psi_1, \Psi_2\}$ with

$$\Psi_1 \equiv (\exists X_A)(\exists X_B)R(X_A,X_B,c) \quad \text{and}$$
$$\Psi_2 \equiv (\exists X_B)(\exists X_C)R(a,X_B,X_C).$$

The corresponding classification instance $r^{pot-sec}$ then consists of the tuples $R^{pot-sec}(\#,\#,c)$ and $R^{pot-sec}(a,\#,\#)$. Now consider the queries

$$\Phi_1 \equiv (\exists X_C)R(a,b,X_C),$$
$$\Phi_2 \equiv R(a,b,c) \quad \text{and}$$
$$\Phi_3 \equiv (\exists X_A)(\exists X_C)R(X_A,b,X_C).$$

The decision of $censor_{\exists R}$ is computed using the following SQL statements:

Φ_1 :
```
SELECT COUNT(*) FROM R_pot_sec
WHERE (A = 'a' OR A = '#') AND (B = 'b' OR B = '#')
AND C = '#'
```

Φ_2 :
```
SELECT COUNT(*) FROM R_pot_sec
WHERE (A = 'a' OR A = '#') AND (B = 'b' OR B = '#') AND
      (C = 'c' OR C = '#')
```

Φ_3 :
```
SELECT COUNT(*) FROM R_pot_sec
WHERE (B = 'b' OR B = '#')
AND A = '#' AND C = '#'
```

These statements return the values 1, 2 and 0, respectively. Consequently, the censor decision for Φ_1 and Φ_2 is `true` and the censor decision for Φ_3 is `false`. $\Diamond$

Remark. Note that there is potential to optimize the representation of policies. For example, given the tuples $\mu_1 \equiv R^{pot-sec}(a,b,c)$ and $\mu_2 \equiv R^{pot-sec}(a,b,\#)$ of a classification instance, it is easy to show that each query that is refused due to μ_1 would also have been refused due to μ_2, since μ_2 is a generalization of μ_1. Consequently, μ_1 could be removed from the policy. Such optimization issues, however, are not addressed in this thesis.

Secondly, we consider the censor function $\widetilde{censor}^{op}_{\exists R}$ (see Definition 7.41) which is used for closed queries and potential secret with free variables. Similar to $censor_{\exists R}$, we assume the policy language $\mathscr{L}_{\exists R,fv}$ here, since disjunctive secrets from the language $\mathscr{L}^{\vee}_{\exists R,fv}$ are split into the single disjuncts when inserted into a classification instance. Although the notion of relevance has not been defined formally for potential secrets from $\mathscr{L}_{\exists R,fv}$, the definition of $\widetilde{censor}^{op}_{\exists R}$ uses a characterization that is similar to the notion of relevance according to Definition 6.4. An exemplary visualization of this characterization can be found in Figure 9.1(b) on page 138: Compared to Figure 9.1(a), the symbol $\sim$ occurs in the potential secret, signifying a free variable. According to Definition 7.41, for the censor function $\widetilde{censor}^{op}_{\exists R}$ to return `true`, the following must hold: If $\sim$ occurs as instantiation of an attribute in the potential secret, then the same attribute must be instantiated with an arbitrary constant in the query. This induces the construction of `<condition_const>` and `<condition_var>` for $\widetilde{censor}^{op}_{\exists R}$.

Definition 9.3 (SQL implementation of $\widetilde{censor}^{op}_{\exists R}$). *Given a query Φ with $A_1, A_2, \dots, A_{m_1}$ denoting the attributes that are instantiated with constants in Φ such that $\Phi[A_i] = a_i$, and $B_1, \dots, B_{m_2}$ denoting the attributes that are instantiated with variables in Φ, the conditions `<condition_const>` and `<condition_var>` (regarding the general SQL implementation of censors on page 137) are defined as follows:*

```
<condition_const> :=
            (A_1 = 'a_1' OR A_1 = '#' OR A_1 = '~') AND
            (A_2 = 'a_2' OR A_2 = '#' OR A_2 = '~') AND ... AND
            (A_m1 = 'a_m1' OR A_m1 = '#' OR A_m1 = '~')
<condition_var> := B_1 = '#' AND B_2 = '#' AND ... AND B_m2 = '#'
```

We illustrate the SQL implementation of the censor function $\widetilde{censor}^{op}_{\exists R}$ with the following example.

Example 9.4. Let a policy be given by $pot\_sec = \{\Psi\}$ with

$$\Psi \equiv (\exists X_A) R(X_A, b, X_C).$$

The corresponding classification instance $r^{pot-sec}$ then consists of the single tuple $R^{pot-sec}(\#, b, \sim)$. Now consider the queries

$$\Phi_1 \equiv (\exists X_C) R(a, b, X_C) \quad \text{and}$$
$$\Phi_2 \equiv (\exists X_A) R(X_A, b, c).$$

The decision of $\widetilde{censor}^{op}_{\exists R}$ is computed using the following SQL statements:

```
Φ1 :  SELECT COUNT(*) FROM R_pot_sec
      WHERE (A = 'a' OR A = '#' OR A = '~') AND
            (B = 'b' OR B = '#' OR B = '~')
      AND C = '#'

Φ2 :  SELECT COUNT(*) FROM R_pot_sec
      WHERE (B = 'b' OR B = '#' OR B = '~') AND
            (C = 'c' OR C = '#' OR C = '~')
      AND A = '#'
```

These statements return the values 0 and 1, respectively. Consequently, the censor decision for Φ_1 is `false` and the censor decision for Φ_2 is `true`. $\Diamond$

Finally, we show how the controlled evaluation of open queries can be performed using the implementation of $\mathit{censor}_{\exists R}$ or $\widetilde{\mathit{censor}}^{op}_{\exists R}$. According to Definition 7.26, stateless CQE for open queries filters the elements of a *refused*-set from the ordinary evaluation of a query and returns the resulting set as answer. Consequently, we need not to compute the whole *refused*-set but only those elements of *refused* that occur in the ordinary evaluation.

Assume a declared policy with elements from $\mathscr{L}_{\exists R}$, that is, existential R-sentences without free variables, as in Definition 7.26. Observe that the elements of *refused* satisfy the same condition that is checked for in the censor function $\mathit{censor}_{\exists R}$. We therefore suggest to invoke the implementation of $\mathit{censor}_{\exists R}$ for each element of the ordinary query evaluation and remove those elements from the answer for which the censor returns `true`, that is, a value ≥ 1. Analogously, if the declared policy contains elements of $\mathscr{L}_{\exists R,fv}$, that is, existential R-sentences with free variables, the censor function $\widetilde{\mathit{censor}}^{op}_{\exists R}$ must be invoked for each element of the ordinary query evaluation.

9.2 Algorithms for Stateless CQE

Having sketched the SQL implementations of the censors $\mathit{censor}_{\exists R}$ and $\widetilde{\mathit{censor}}^{op}_{\exists R}$ in Section 9.1, we now propose and analyze two algorithms for stateless CQE. Algorithm 1 evaluates closed queries from the language $\mathscr{L}^{\wedge}_{\exists R}$, whereas Algorithm 2 evaluates open queries from the language $\mathscr{L}^{\wedge}_{\exists R,fv}$. Although closed queries can be seen as special case of open queries, we propose two separate algorithms. This is justified by the different properties of "pure" closed query scenarios and "mixed" scenarios with closed and open queries:

- In closed query scenarios, the user is allowed to be aware of the confidentiality policy, whereas in mixed scenarios, the confidentiality policy must be unknown to him (see Section 7.3).

- If a closed query is answered by the empty set in a mixed scenario, the query is either false in the database instance or harmful, but the user cannot distinguish these alternatives. In a closed query scenario, however, this distinction can be made: If the negation of the query is returned as answer, the query is false in the database instance; if `mum` is returned as answer, the query is harmful with respect to the declared confidentiality policy.

Consequently, the security administrator of a database system must decide which algorithm should be used in his system. He thereby also determines the expressive means of the database user (closed queries or open queries).

For both algorithms, we assume that the database user formulates a syntactically correct SQL query as input. The classes of syntactically correct queries for Algorithm 1 and Algorithm 2 have been described in Subsections 8.1.3 and 8.1.5, respectively. Observe that, due to the structure of the suggested SQL implementations, it is easy for a suitable parser to split a query into its conjuncts; this feature is important, since both algorithms actually operate on the single conjuncts rather than on the whole query. Besides the query, the algorithms take the database instance and the classification instance (see Section 8.2) as inputs and return an array *ans* as output which contains the answers to the single conjuncts of the query.[24]

Firstly, consider Algorithm 1 on page 144. For each conjunct Φ_i of the query Φ, the function `censor` is invoked with the conjunct Φ_i and the classification instance $r^{pot-sec}$ as parameters. This function corresponds to the SQL implementation of $censor_{\exists R}$ according to Definition 9.1 or $\widetilde{censor}^{op}_{\exists R}$ according to Definition 9.3, depending on whether the security administrator specified potential secrets from the language $\mathscr{L}^{\vee}_{\exists R}$ or from the language $\mathscr{L}^{\vee}_{\exists R,fv}$, respectively.[25] If the censor decision is `true`, the answer to the conjunct under consideration Φ_i is refused by inserting the special value `mum` into the array *ans*; otherwise, the result of the ordinary evaluation of Φ_i is inserted into *ans* by invoking the function `eval_closed` on Φ_i and the database instance r. This function returns `true` or `false`, depending on the truth value of Φ_i in r (see the SQL representation of existential R-sentences in Subsection 8.1.2 for a detailed description of the implementation). Finally, *ans* is returned to the user.

Since Algorithm 1 loops over a finite set of conjuncts and the functions `censor` and `eval_closed` can basically be implemented by SQL statements, the algorithm is guaranteed to terminate. We now analyze its computational complexity:

- The initialization of *ans* (line 1) takes time $\mathcal{O}(m)$.
- The *for*-loop (lines 2–8) is performed m times.

[24]For simplicity, this evaluation slightly deviates from the one suggested in Section 8.1: The answer sets to the single conjuncts are returned rather than computing their Cartesian product.

[25]We assume that the results of the SQL implementations of the censors are interpreted as truth values, as described in Section 9.1.

- According to Section 9.1, the censor decision (line 3) is basically a select-query on the classification instance $r^{pot-sec}$ with constantly many comparisons. It can be implemented by a relation scan or, if the tuples of $r^{pot-sec}$ are appropriately indexed by a B-tree (as already discussed in Section 6.4), by a tree search in time $\mathcal{O}(log(|r^{pot-sec}|))$ where $|r^{pot-sec}|$ denotes the number of tuples in $r^{pot-sec}$.
- Within the *for*-loop, we make the worst case assumption that the *else*-branch (line 6) is executed (the execution of the *if*-branch (line 4) would take constant time).
- The complexity of the evaluation of Φ_i is estimated as follows: Φ_i is a select-project-query on r. Provided that r is appropriately indexed, the select operation can be performed in time $\mathcal{O}(log(|r|))$ where $|r|$ denotes the number of tuples in r. The projection operation first scans the relation and then, if necessary, removes duplicates from the result. This duplicate removal can be implemented, for example, by sorting and then again scanning the result. The complexity can thus be estimated by $\mathcal{O}(|r| \cdot log(|r|))$ for the sorting which exceeds the complexity of the select operation. As a result, the evaluation of Φ_i has a complexity of $\mathcal{O}(|r| \cdot log(|r|))$.[26]

Since the complexity of the *for*-loop exceeds the complexity of the initialization, Algorithm 1 has an overall complexity of

$$\mathcal{O}(m \cdot (log(|r^{pot-sec}|) + |r| \cdot log(|r|))). \tag{9.1}$$

Secondly, consider Algorithm 2 on the following page for the evaluation of open queries. For each conjunct Φ_i of the query Φ, the sets *controlled_evaluation* and *evaluation* are initialized, the former of which is empty and the latter of which contains the ordinary evaluation of Φ_i with respect to r (which is computed by the function `eval_open`, see Subsection 8.1.4 for a detailed description of the implementation). Then, each element of *evaluation* is checked by the `censor` function which is, as in Algorithm 1, the SQL implementation of $censor_{\exists R}$ or $\widetilde{censor}^{op}_{\exists R}$. If the censor decision is `false`, the element under consideration is "harmless" and thus added to *controlled_evaluation*. In doing so, the algorithm directly computes the set "$eval^{*} \setminus refused$" from Definition 7.26 without explicitly determining the $refused$-set. For each conjunct Φ_i, the set *controlled_evaluation* is then inserted into the array *ans*. Finally, *ans* is returned to the user.

Termination of Algorithm 2 is guaranteed, since it loops over finite sets and the function `eval_open` as well as the function `censor` can basically be implemented with

[26]For the evaluation of closed queries according to Subsection 8.1.2, it is only important whether or not the constructed SQL query returns the empty relation. Therefore, this evaluation might be optimizable in order to achieve an even better complexity. We, however, do not address such optimizations in this thesis.

Algorithm 1 Stateless CQE for closed queries.

```
Input: query $\Phi \in \mathscr{L}^{\wedge}_{\exists R}$, database instance $r$, classification instance $r^{pot-sec}$
Output: controlled answer ans
 1: initialize array ans of length $m$ (number of conjuncts in $\Phi$)
 2: for all conjuncts $\Phi_i$ of $\Phi$ do
 3:     if censor($\Phi_i$,$r^{pot-sec}$) is true then
 4:         ans[i] ← mum
 5:     else
 6:         ans[i] ← eval_closed($\Phi_i$,$r$)
 7:     end if
 8: end for
 9: return ans
```

Algorithm 2 Stateless CQE for open queries.

```
Input: query $\Phi \in \mathscr{L}^{\wedge}_{\exists R,fv}$, database instance $r$, classification instance $r^{pot-sec}$
Output: controlled answer ans
 1: initialize array ans of length $m$ (number of conjuncts in $\Phi$)
 2: for all conjuncts $\Phi_i$ of $\Phi$ do
 3:     controlled_evaluation ← $\emptyset$
 4:     evaluation ← eval_open($\Phi_i$,$r$)
 5:     for all $\mu \in$ evaluation do
 6:         if censor($\mu$,$r^{pot-sec}$) is false then
 7:             controlled_evaluation ← controlled_evaluation $\cup \{\mu\}$
 8:         end if
 9:     end for
10:     ans[i] ← controlled_evaluation
11: end for
12: return ans
```

SQL statements. The computational complexity of the algorithm can be estimated as follows:

- The initialization of *ans* (line 1) takes time $\mathcal{O}(m)$.
- The outer *for*-loop (lines 2–11) is performed m times.
- The assignments in lines 3, 7 and 10 take constant time.
- The evaluation of Φ_i (line 4) has a complexity of $\mathcal{O}(|r| \cdot log(|r|))$ (see the analysis of Algorithm 1).
- The inner loop (lines 5–9) is performed $|eval^*(\Phi_i)(r)|$ times. Since Φ_i is a select-project-query on the single relation r, its evaluation contains at most $|r|$ tuples. Consequently, $|eval^*(\Phi_i)(r)|$ can be estimated by $\mathcal{O}(|r|)$.
- The censor decision can be computed in time $\mathcal{O}(log(|r^{pot-sec}|))$ (see the analysis of Algorithm 1).

Again, the complexity of the initialization of *ans* in line 1 can be neglected and consequently, the overall complexity of Algorithm 2 is

$$\mathcal{O}(m \cdot (|r| \cdot (log(|r|) + |r| \cdot log(|r^{pot-sec}|)))). \tag{9.2}$$

Considering $m + |r| + |r^{pot-sec}|$ as the input length of Algorithm 1 and Algorithm 2, the estimations (9.1) and (9.2) indicate that both algorithms are efficient in terms of computational complexity.

Part IV

Conclusion

Chapter 10

Summary and Evaluation

In this thesis, we proposed the policy-driven technique of stateless Controlled Query Evaluation, or stateless CQE for short, as an approach to efficient inference control in relational databases. Basically, stateless CQE can be seen as a mediating layer between a relational database system and the querying users of this system. In order to preserve confidentiality of sensitive or personal information, stateless CQE inspects each user query and refuses to give an answer to this query if necessary. In particular, stateless CQE never lies to a user. To provide information security rather than only data security, stateless CQE does not attach labels to the contents of the database (like most access control approaches), but relies on a confidentiality policy that has been declared by a security administrator.

We took the inference control mechanism of Controlled Query Evaluation (CQE) as a starting point for our investigations. CQE has been developed for a general logic framework and is thus suitable for relational databases which can be modeled using first-order logic. For each database query, CQE decides by means of a censor function if the honest answer to this query together with the (assumed) user knowledge allows for the disclosure of confidential information, and modifies the answer if necessary. Due to its general applicability, however, CQE involves a major drawback in the context of relational databases: The underlying decision problem of the mechanism is undecidable in first-order logic. Even if the problem is suitably restricted, it remains computationally expensive, since it is basically theorem proving. While original CQE is stateful, that is, it relies on the usage history of the system by keeping track of the knowledge of the user, stateless CQE renders this user log redundant by preventing "sophisticated inferences". More precisely, our approach restricts the query language of the database system so that a user cannot exploit his knowledge in order to disclose confidential information. As a consequence, it suffices to inspect each query isolatedly in order to decide whether or not the answer to this query must be refused.

We investigated stateless CQE for a single user system and assumed a single database relation which is complete (that is, it does not contain null values) and has functional dependencies as semantic constraints. Moreover, we supposed the contents of the database to be static, meaning that we did not consider update operations. We assumed the database user to be capable of (deductive) reasoning in first-order logic and to be aware of the declared confidentiality policy which consists of so-called potential secrets. Due to the logic framework, query languages and policy languages were supposed to be fragments of the relational calculus. We started our investigations with closed select-queries and partial tuples as potential secrets, and showed that this combination is easy to handle with a stateless CQE mechanism: For the censor function of stateless CQE, it suffices to perform a pattern matching between the query and the declared secrets in order to decide whether or not the query must be refused to preserve confidentiality. Unlike the censor functions in (stateful) CQE, this pattern matching can be implemented in logarithmic time.

As an enhancement of the query language, we then investigated select-project-queries. This enhancement seems to be minor, but we showed that it has the potential to compromise confidentiality: Since the user is able to learn parts of tuples when using select-project-queries, he might express several harmless queries and combine the respective answers to a confidential piece of information afterwards by using, for example, a functional dependency. We therefore had to suitably restrict the policy language and the semantic constraints of the database relation in order to prevent such unwanted inferences. Besides functional dependencies, also implicative structures in queries can be exploited to infer confidential information. We elaborated on this issue when examining Boolean combinations of select-project-queries and showed that only conjunction and a restricted form of negation in queries can be handled with stateless CQE. Moreover, we investigated Boolean operations for potential secrets and found that only disjunction should be integrated into the policy language when aiming at guaranteeing preservation of confidentiality. The next step in enhancing the query language was the introduction of free variables, enabling the user to express open queries. The assumption that the database user is aware of the confidentiality policy turned out to be harmful in the context of open queries. We showed, however, that dropping this assumption suffices to regain the property of confidentiality preservation. We also considered free variables for the policy language which enable the security administrator to express “secret schemas”, that is, several potential secrets with a single policy element. For proving such policies confidentiality preserving, we (conceptually) expanded potential secrets with free variables by replacing these variables with all possible constant combinations (which yields an infinite policy).

We then brought together our results and proposed comprehensive query and policy languages that are suitable for our stateless CQE mechanism. Since negation turned out to be problematic in the context of open queries, we refrained from integrating it into the comprehensive query language. For all combinations of query language and policy language, we were able to formally prove that stateless CQE preserves confidentiality. Summarized, our stateless CQE approach works on select-project-queries with conjunction and free variables, and on potential secrets with disjunction and free variables. With several examples, we justified that within the assumed parameters, these query and policy languages are “quite complete”, that is, they basically cover the space of possibilities for stateless CQE in relational databases.

Finally, we proposed an implementation of the stateless CQE mechanism. We described an algorithm for closed queries and an algorithm for open queries, and showed that these algorithms are computationally efficient. Both algorithms basically use SQL statements for computing the censor decisions and thus they are in principle suitable to be integrated into relational database management systems as an inference control layer.

Our stateless CQE mechanism has several advantages over previous approaches to confidentiality preservation in relational databases, the most important of which are sketched in the following.

- Many previous approaches basically enforce access control mechanisms rather than inference control and are thus not able to prevent unwanted information flows, as discussed in Section 1.1. In contrast, stateless CQE not only provides data access control but also information flow control.

- Some inference control approaches are extensions of access control, see, for example, the works of Su and Özsoyoglu [SÖ91], Stickel [Sti94], Dawson, De Capitani di Vimercati and Samarati [DDS99], and Brodsky, Farkas and Jajodia [BFJ00]. Consequently, these approaches assign labels to database entries and base the decision whether or not a user query may be answered honestly on these labels. Stateless CQE, however, is policy-driven, that is, the decision whether or not a query must be refused is based on a confidentiality policy that is independent of the actual data. Therefore, when being integrated into existing databases, stateless CQE mechanisms do not require to elaborately (re-)classify the data.

- Approaches to confidentiality preservation often do not consider the knowledge a user might have about the inner workings of the system. For example, the work of Brodsky, Farkas and Jajodia [BFJ00] neglects that a user might disclose confidential information from the fact that a query is refused. Stateless CQE takes such “meta reasoning” into account. For example, we showed that a user who is aware of the confidentiality policy and the actual functionality of scqe^{oq} might exploit open queries to disclose confidential information. Consequently, we hide the policy from the user in open query scenarios.

- While previous CQE approaches maintain a user log in order to be able to preserve confidentiality, stateless CQE only needs to inspect the query and the confidentiality policy. This results in smaller space requirements and a lower computational complexity.

- Finally, stateless CQE is fairly easy to implement: Provided that the confidentiality policy is given as a database relation, the censor decisions of stateless CQE can be determined using SQL statements. Consequently, the mechanism mainly relies on the functionality of the underlying relational database management system.

Nevertheless, the stateless CQE approach that has been developed in this thesis still has a few shortcomings, approaches to some of which are discussed in Chapter 11. For example, the assumption of a single database user is not tenable for a practical application of the approach. In multi-user scenarios, however, it should be easy to maintain and enforce a separate confidentiality policy for each user of the system. Moreover, database relations are usually dynamic in the sense that their contents may be changed by users performing update operations; stateless CQE does not support updates so far. The exploration in Chapter 8 shows that the query languages of stateless CQE only cover a fragment of SQL. In particular, joins, subqueries and aggregate functions are not supported yet. A related issue is the assumption of a single database relation: While in practice relational databases often consist of several relations, stateless CQE only works on single relations in order to prevent harmful inferences due to inter-relational constraints.

Despite these shortcomings, we believe that stateless CQE is a suitable approach to inference control in relational databases. Our investigations, however, give evidence to the fact that effective and efficient inference control comes at the price of a restricted framework.

Chapter 11

Directions for Future Work

In the final chapter of this thesis, we outline some directions for future research in the field of stateless CQE. Although being an exemplary selection, we believe that an investigation of the following issues is especially important in order to make stateless CQE a convenient inference control mechanism for relational databases.

In Section 11.1, we consider relations that are not in ONF: When subjecting an existing database to inference control, the relations of this database possibly do not adhere to the ONF restriction; consequently an extension of stateless CQE to relations with arbitrary FDs is desirable. In Section 11.2, we address databases with more than one relation, an investigation of which would be an important step towards the treatment of join queries. Finally, in Section 11.3, we roughly outline approaches to more expressive query languages.

11.1 Relaxing the Schema Constraints

In Part II of this thesis, we developed query and policy languages that are as expressive as possible within the given parameters. It turned out that the schema of the underlying database relation should be in ONF in order to guarantee preservation of confidentiality. This assumption, however, might be too restrictive for practical databases. In existing database relations, possibly more semantic constraints than one unique key dependency hold. Note that although such additional semantic constraints are not necessarily represented in the relation schema, the database user might be aware of them due to external information sources or commonsense knowledge. In the following, we sketch how relation schemas with arbitrary FDs (that is, schemas that are not in ONF) could be treated.

Recall that FDs can be exploited to infer potential secrets in several query steps if the policy language is not suitably restricted. This is performed by querying the parts of the potential secret separately and then connecting these parts using FDs. More precisely, given an FD $\mathcal{A} \to \mathcal{B}$, the occurrence of an attribute value combination of two attributes B_1 and B_2 (with $B_1, B_2 \in \mathcal{B}$) can be disclosed by first asking for the attribute value combination of $\mathcal{A}, B_1$ and then asking for the attribute value combination of $\mathcal{A}, B_2$; exploiting the FD then discloses the attribute value combination of $\mathcal{A}, B_1, B_2$ (see also Example 7.4).

Consequently, when allowing relation schemas with arbitrary FDs, potential secrets consisting of two (or more) values of attributes that occur on the right-hand side of an FD cannot be protected with stateless CQE. Therefore, the policy language has to be suitably restricted in order to guarantee preservation of confidentiality. This restriction could be a generalized form of the basic facts restriction from Section 7.1.

11.2 Multiple Relations and Join Queries

So far, stateless CQE only works on single relations. Our investigations can straightforwardly be transferred to databases with multiple relations if there are no inter-relational semantic constraints and if each query is still constrained to refer to only one of these relations. Otherwise, the database user might be able to disclose confidential information as justified in the following.

Firstly, consider inclusion dependencies as an example for inter-relational constraints. An inclusion dependency basically says that all instantiations of a set of attributes occurring in one relation R also occur as instantiations of these attributes in another relation S. If one of these instantiations is declared a potential secret in S but not in R, the user can ask for this instantiation in R and, if the answer is positive, infer that it also occurs in S by exploiting the inclusion dependency. He thereby discloses a secret.

Secondly, it might be desirable for the database user to express join queries, that is, queries that refer to more than one relation. This, however, might be exploited to disclose confidential information in the following way: Consider two instances r and s of the relation schemas $\langle R, AB, A \to B\rangle$ and $\langle S, BC, B \to C\rangle$, respectively, and suppose that the potential secret $\Psi \equiv S(b,c)$ is declared. The natural join $t = r \bowtie s$ of these instances is an instance with the schema $\langle T, ABC, \Sigma\rangle$ such that for each tuple μ in t it holds that the projection of μ to the attributes A, B is in r and the projection of μ to the attributes B, C is in s. For the set Σ of semantic constraints of T it particularly holds that $\Sigma \models A \to BC$. Regarding join queries, T is usually not materialized but constructed as a view relation over R and S. For example, $\Phi_1 \equiv (\exists X_B)(R(a,X_B) \land S(X_B,c))$ and $\Phi_2 \equiv (\exists X_C)(R(a,b) \land S(b,X_C))$ are join queries. If both Φ_1 and Φ_2 are answered with `true` by a stateless CQE mechanism, the user knows that $(\exists X_B)T(a,X_B,c)$ and $(\exists X_C)T(a,b,X_C)$ are true in the join relation T. Then, exploiting the FD $A \to BC$, the user can infer that $T(a,b,c)$ is true, and by the definition of the natural join, it follows that $S(b,c)$ is true as well. This fact, however, discloses the potential secret Ψ.

On the contrary, not every join query is harmful with respect to the declared policy. This is due to the fact that the user is able to compute the join of two instances by himself if he knows the tuples of these instances. Consequently, if he is already allowed to know these tuples, it would not be reasonable to reject the join query. For example, reconsider the instances r and s and the potential secret $\Psi \equiv S(b,c)$ from above: The user can compute the join $t = r \bowtie s$ by sending the queries $\Phi'_1(X_A,X_B) \equiv R(X_A,X_B)$ and $\Phi'_2(X_B,X_C) \equiv S(X_B,X_C)$ to the database and afterwards "manually" joining the answer sets. Consequently, the system could also directly accept the join query $\Phi'_3(X_A,X_B,X_C) \equiv R(X_A,X_B) \wedge S(X_B,X_C)$. In this case, however, the single conjuncts of Φ'_3 must be inspected by the stateless CQE mechanism and as a consequence, the tuple $S(b,c)$ is filtered from the answer set to $S(X_B,X_C)$ and does not occur as part of the join.

From these examples, it can be seen that extensions of stateless CQE to databases with multiple relations and query languages that allow to express join queries require thorough investigations of the possible effects. Regarding inclusion dependencies, a stateless CQE censor should not only consider the relation R the query refers to, but also relations in which R or parts of R are included. The problem that might arise with join queries is related to the problem with relaxed schema constraints in Section 11.1, since the join of two ONF relations is in general not an ONF relation but a relation with arbitrary FDs. Thus, a solution to the former problem might be inspired by a solution to the latter problem.

11.3 Relaxing the Query Language Constraints

Although in Chapter 7 we proposed several enhancements for the basic query language from Chapter 6, the resulting language $\mathscr{L}^{\wedge}_{\exists R,fv}$ is still restricted. In particular, it is not possible to express disjunction and negation. On the one hand, for a database user it would be desirable to have a more expressive language at disposal; on the other hand, however, in Chapter 7 we justified that $\mathscr{L}^{\wedge}_{\exists R,fv}$ is already "maximal", that is, further enhancements would possibly compromise confidentiality. In the following, we sketch some ideas regarding potential enhancements of the query language.

As an approach to disjunctive queries, we suggest to refine the answering strategy for conjunctive queries (see Section 7.2), as outlined for positive closed queries in the following. Consider a query Φ that is a positive Boolean combination of existential R-sentences, that is, it may contain conjunction and disjunction. For simplicity, assume that the elements of Φ are variable-disjoint, that is, each variable occurs at most once in Φ. According to Definition 7.14, a conjunctive query is split into its single conjuncts before being forwarded to the censor function; analogously, Φ can be split into its elements, each of which is then forwarded to the censor function. More precisely, we suggest to bring Φ in disjunctive normal form (DNF) in the first step and get a result of the form $\Phi_{DNF} \equiv (\bigvee_{i=1}^{m}(\bigwedge_{j=1}^{n_i}(\exists^*_{i,j})\Phi_{i,j}))$ with $\Phi_{i,j}$ denoting a quantifier-free formula and $\exists^*_{i,j}$ being a shortcut for the existential quantification of the variables that occur in $\Phi_{i,j}$. Due to the variable disjointness, this transformation from Φ to Φ_{DNF} is always possible and it holds that $\Phi_{DNF} \equiv \Phi$. In the second step, disjunction (which has been shown to be harmful regarding preservation of confidentiality) is transformed into conjunction, and we get $\Phi' \equiv (\bigwedge_{i=1}^{m}(\bigwedge_{j=1}^{n_i}(\exists^*_{i,j})\Phi_{i,j}))$. Observe that $\Phi' \in \mathscr{L}^{\wedge}_{\exists R}$[27], and consequently Φ' can be answered by $scqe^{\neg,\wedge}$ (see Definition 7.14). However, $\Phi' \not\equiv \Phi_{DNF}$ and thus also $\Phi' \not\equiv \Phi$ which means that the original user query is semantically modified before being answered. But note that $\Phi' \models \Phi$ holds, so if Φ' is answered by the stateless CQE mechanism, the user at least gets information that is related to his original query Φ.

In Example 7.45, we learned that queries containing free variables and negation might be unsafe. There are, however, conditions under which negation does not lead to unsafe queries, as illustrated by the following example: Consider an instance r of the relation schema $\langle R, AB, \emptyset\rangle$ and the query $\Phi(X_B) \equiv (\exists X_A)R(X_A,X_B) \wedge \neg R(a,X_B)$. The first part of Φ asks for the values of attribute B that occur in r, whereas the second part of Φ asks for the values of attribute B that do not occur in r in combination with the value a of attribute A. Consequently, the evaluation of Φ can be interpreted as the intersection of the evaluation of $(\exists X_A)R(X_A,X_B)$ and the evaluation of $\neg R(a,X_B)$, which yields a finite set of tuples and makes Φ safe. In general, each conjunction $\chi_1 \wedge \chi_2$ of two elements χ_1, χ_2 of the language $\mathscr{L}_{\exists R,fv}$ can be interpreted as a join (which is a generalized form of intersection); moreover, if all variables that occur freely in χ_2 also occur freely in χ_1, χ_2 may also be negated, as argued above.

Since negation leads to safe queries if suitably "guarded" by a positive query part, it is not reasonable to categorically exclude it from the query language. Moreover, multiple usage of free variables may be allowed in order to express general joins, as already discussed in Section 11.2. In order to integrate these enhancements into our approach, the query language, the stateless CQE mechanism, the SQL translation and the algorithmic implementation must be adapted suitably. We believe, however, that these enhancements, although being a step towards a more user-friendly query language, would not substantially change the core of our stateless CQE approach.

[27]The language $\mathscr{L}^{\wedge}_{\exists R}$ has been introduced in Subsection 8.1.3.

Bibliography

[AHV95] Abiteboul, Serge, Richard Hull, and Victor Vianu: *Foundations of Databases*. Addison-Wesley, 1995.

[BB01] Biskup, Joachim and Piero A. Bonatti: *Lying versus refusal for known potential secrets*. Data & Knowledge Engineering, 38(2):199–222, 2001.

[BB04a] Biskup, Joachim and Piero A. Bonatti: *Controlled query evaluation for enforcing confidentiality in complete information systems*. International Journal of Information Security, 3(1):14–27, 2004.

[BB04b] Biskup, Joachim and Piero A. Bonatti: *Controlled query evaluation for known policies by combining lying and refusal*. Annals of Mathematics and Artificial Intelligence, 40(1–2):37–62, 2004.

[BB07] Biskup, Joachim and Piero A. Bonatti: *Controlled query evaluation with open queries for a decidable relational submodel*. Annals of Mathematics and Artificial Intelligence, 50(1–2):39–77, 2007.

[BBFR00] Bertino, Elisa, Francesco Buccafurri, Elena Ferrari, and Pasquale Rullo: *A logic-based approach for enforcing access control*. Journal of Computer Security, 8(2/3):109–139, 2000.

[BEL08] Biskup, Joachim, David W. Embley, and Jan-Hendrik Lochner: *Reducing inference control to access control for normalized database schemas*. Information Processing Letters, 106(1):8–12, 2008.

[BFJ00] Brodsky, Alexander, Csilla Farkas, and Sushil Jajodia: *Secure databases: constraints, inference channels, and monitoring disclosures*. IEEE Transactions on Knowledge and Data Engineering, 12(6):900–919, 2000.

[BGSW09] Biskup, Joachim, Christian Gogolin, Jens Seiler, and Torben Weibert: *Requirements and protocols for inference-proof interactions in information systems*. In Backes, Michael and Peng Ning (editors): *Computer Security – ESORICS 2009, Proceedings of the 14th European Symposium on Research in Computer Security*, volume 5789 of *Lecture Notes in Computer Science*, pages 285–302. Springer, 2009.

[BHLL10a] Biskup, Joachim, Sven Hartmann, Sebastian Link, and Jan-Hendrik Lochner: *Chasing after secrets in relational databases.* In Laender, Alberto H. F. and Laks V. S. Lakshmanan (editors): *Proceedings of the 4th Alberto Mendelzon International Workshop on Foundations of Data Management, AMW 2010*, volume 619 of *CEUR Workshop Proceedings.* CEUR-WS.org, 2010.

[BHLL10b] Biskup, Joachim, Sven Hartmann, Sebastian Link, and Jan-Hendrik Lochner: *Efficient inference control for open relational queries.* In Foresti, Sara and Sushil Jajodia (editors): *Data and Applications Security and Privacy XXIV, Proceedings of the 24th Annual IFIP WG 11.3 Working Conference, DBSec 2010*, volume 6166 of *Lecture Notes in Computer Science*, pages 162–176. Springer, 2010.

[Bis89] Biskup, Joachim: *Boyce-Codd normal form and object normal forms.* Information Processing Letters, 32(1):29–33, 1989.

[Bis00] Biskup, Joachim: *For unknown secrecies refusal is better than lying.* Data & Knowledge Engineering, 33(1):1–23, 2000.

[Bis09] Biskup, Joachim: *Security in Computing Systems: Challenges, Approaches and Solutions.* Springer, 2009.

[Bis10] Biskup, Joachim: *Usability confinement of server reactions: maintaining inference-proof client views by controlled interaction execution.* In Kikuchi, Shinji, Shelly Sachdeva, and Subhash Bhalla (editors): *Proceedings of the 6th International Workshop on Databases in Networked Information Systems, DNIS 2010*, volume 5999 of *Lecture Notes in Computer Science*, pages 80–106. Springer, 2010.

[BKS95] Bonatti, Piero A., Sarit Kraus, and V. S. Subrahmanian: *Foundations of secure deductive databases.* IEEE Transactions on Knowledge and Data Engineering, 7(3):406–422, 1995.

[BL76] Bell, David E. and Leonard J. LaPadula: *Secure computer system: unified exposition and Multics interpretation.* Technical Report ESD-TR-75-306, The MITRE Corporation, 1976.

[BL07] Biskup, Joachim and Jan-Hendrik Lochner: *Enforcing confidentiality in relational databases by reducing inference control to access control.* In Garay, Juan A., Arjen K. Lenstra, Masahiro Mambo, and René Peralta (editors): *Information Security, Proceedings of the 10th International Conference, ISC 2007*, volume 4779 of *Lecture Notes in Computer Science*, pages 407–422. Springer, 2007.

[BLS09] Biskup, Joachim, Jan-Hendrik Lochner, and Sebastian Sonntag: *Optimization of the controlled evaluation of closed relational queries.* In Gritzalis, Dimitris and Javier Lopez (editors): *Emerging Challenges for Security, Privacy and Trust, Proceedings of the 24th IFIP TC 11 International Information Security Conference, SEC 2009*, volume 297 of *IFIP Advances in Information and Communication Technology*, pages 214–225. Springer, 2009.

[BM72] Bayer, Rudolf and Edward M. McCreight: *Organization and maintenance of large ordered indices.* Acta Informatica, 1(3):173–189, 1972.

[BS05] Bertino, Elisa and Ravi S. Sandhu: *Database security—concepts, approaches, and challenges.* IEEE Transactions on Dependable and Secure Computing, 2(1):2–19, 2005.

[BSW09] Biskup, Joachim, Jens Seiler, and Torben Weibert: *Controlled query evaluation and inference-free view updates.* In Gudes, Ehud and Jaideep Vaidya (editors): *Data and Applications Security XXIII, Proceedings of the 23rd Annual IFIP WG 11.3 Working Conference, DBSec 2009*, volume 5645 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 2009.

[BTW10] Biskup, Joachim, Cornelia Tadros, and Lena Wiese: *Towards controlled query evaluation for incomplete first-order databases.* In Link, Sebastian and Henri Prade (editors): *Foundations of Information and Knowledge Systems, Proceedings of the 6th International Symposium, FoIKS 2010*, volume 5956 of *Lecture Notes in Computer Science*, pages 230–247. Springer, 2010.

[BW08a] Biskup, Joachim and Torben Weibert: *Keeping secrets in incomplete databases.* International Journal of Information Security, 7(3):199–217, 2008.

[BW08b] Biskup, Joachim and Lena Wiese: *Preprocessing for controlled query evaluation with availability policy.* Journal of Computer Security, 16(4):477–494, 2008.

[BW09] Biskup, Joachim and Lena Wiese: *Combining consistency and confidentiality requirements in first-order databases.* In Samarati, Pierangela, Moti Yung, Fabio Martinelli, and Claudio A. Ardagna (editors): *Information Security, Proceedings of the 12th International Conference, ISC 2009*, volume 5735 of *Lecture Notes in Computer Science*, pages 121–134. Springer, 2009.

[CC08] Chen, Yu and Wesley W. Chu: *Protection of database security via collaborative inference detection.* In Chen, Hsinchun and Christopher C. Yang (editors): *Intelligence and Security Informatics, Techniques and Applications*, volume 135 of *Studies in Computational Intelligence*, pages 275–303. Springer, 2008.

[CFMS94] Castano, Silvana, Maria Grazia Fugini, Giancarlo Matella, and Pierangela Samarati: *Database Security.* Addison-Wesley, 1994.

[CG98] Cuppens, Frédéric and Alban Gabillon: *Rules for designing multilevel object-oriented databases.* In Quisquater, Jean Jacques, Yves Deswarte, Catherine Meadows, and Dieter Gollmann (editors): *Computer Security – ESORICS 1998, Proceedings of the 5th European Symposium on Research in Computer Security*, volume 1485 of *Lecture Notes in Computer Science*, pages 159–174. Springer, 1998.

[CG99] Cuppens, Frédéric and Alban Gabillon: *Logical foundations of multilevel databases.* Data & Knowledge Engineering, 29(3):259–291, 1999.

[CG01] Cuppens, Frédéric and Alban Gabillon: *Cover story management.* Data & Knowledge Engineering, 37(2):177–201, 2001.

[CH08] Cuenca Grau, Bernardo and Ian Horrocks: *Privacy-preserving query answering in logic-based information systems.* In Ghallab, Malik, Constantine D. Spyropoulos, Nikos Fakotakis, and Nikolaos M. Avouris (editors): *Proceedings of the 18th European Conference on Artificial Intelligence, ECAI 2008*, volume 178 of *Frontiers in Artificial Intelligence and Applications*, pages 40–44. IOS Press, 2008.

[CM00] Chang, LiWu and Ira S. Moskowitz: *An integrated framework for database privacy protection.* In Thuraisingham, Bhavani M., Reind P. van de Riet, Klaus R. Dittrich, and Zahir Tari (editors): *Data and Application Security, Development and Directions, Proceedings of the 14th Annual IFIP WG 11.3 Working Conference on Database Security, DBSec 2000*, volume 201 of *IFIP Conference Proceedings*, pages 161–172. Kluwer, 2000.

[Cod70] Codd, Edgar F.: *A relational model of data for large shared data banks.* Communications of the ACM, 13(6):377–387, 1970.

[DDS99] Dawson, Steven, Sabrina De Capitani di Vimercati, and Pierangela Samarati: *Specification and enforcement of classification and inference constraints.* In *Proceedings of the 20th IEEE Symposium on Security and Privacy*, pages 181–195. IEEE Computer Society, 1999.

[Den82] Denning, Dorothy E.: *Cryptography and Data Security.* Addison-Wesley, 1982.

[DH96] Delugach, Harry S. and Thomas H. Hinke: *Wizard: a database inference analysis and detection system.* IEEE Transactions on Knowledge and Data Engineering, 8(1):56–66, 1996.

[DP05] Deutsch, Alin and Yannis Papakonstantinou: *Privacy in database publishing.* In Eiter, Thomas and Leonid Libkin (editors): *Database Theory – ICDT 2005, Proceedings of the 10th International Conference*, volume 3363 of *Lecture Notes in Computer Science*, pages 230–245. Springer, 2005.

[DS83] Denning, Dorothy E. and Jan Schlörer: *Inference controls for statistical databases.* IEEE Computer, 16(7):69–82, 1983.

[FJ02] Farkas, Csilla and Sushil Jajodia: *The inference problem: a survey.* SIGKDD Explorations, 4(2):6–11, 2002.

[FP05] Fernández-Medina, Eduardo and Mario Piattini: *Designing secure databases.* Information and Software Technology, 47(7):463–477, 2005.

[FTE01] Farkas, Csilla, Tyrone S. Toland, and Caroline M. Eastman: *The inference problem and updates in relational databases.* In Olivier, Martin S. and David L. Spooner (editors): *Database and Application Security XV, Proceedings of the 15th Annual IFIP WG 11.3 Working Conference on Database and Application Security, DBSec 2001*, volume 215 of *IFIP Conference Proceedings*, pages 181–194. Kluwer, 2001.

[Gol06] Gollmann, Dieter: *Computer Security.* John Wiley & Sons, 2nd edition, 2006.

[GW76] Griffiths, Patricia P. and Bradford W. Wade: *An authorization mechanism for a relational database system.* ACM Transactions on Database Systems, 1(3):242–255, 1976.

[HO08] Halpern, Joseph Y. and Kevin R. O'Neill: *Secrecy in multiagent systems.* ACM Transactions on Information and Systems Security, 12(1), 2008.

[HS96] Hale, John and Sujeet Shenoi: *Analyzing FD inference in relational databases.* Data & Knowledge Engineering, 18(2):167–183, 1996.

[J$^+$09] Jeloka, Sumit *et al.*: *Oracle Label Security Administrator's Guide, 11g Release 2 (11.2).* E10745-02, Oracle Corporation, 2009.

[JS91a] Jajodia, Sushil and Ravi S. Sandhu: *A novel decomposition of multilevel relations into single-level relations.* In *Proceedings of the 12th IEEE Symposium on Security and Privacy*, pages 300–315. IEEE Computer Society, 1991.

[JS91b] Jajodia, Sushil and Ravi S. Sandhu: *Toward a multilevel secure relational data model.* In *Proceedings of the 1991 ACM SIGMOD International Conference on Management of Data*, pages 50–59. ACM, 1991.

[JSSS01] Jajodia, Sushil, Pierangela Samarati, Maria Luisa Sapino, and V. S. Subrahmanian: *Flexible support for multiple access control policies.* ACM Transactions on Database Systems, 26(2):214–260, 2001.

[KMN05] Kenthapadi, Krishnaram, Nina Mishra, and Kobbi Nissim: *Simulatable auditing.* In Li, Chen (editor): *Principles of Database Systems, Proceedings of the 24th ACM Symposium, PODS 2005*, pages 118–127. ACM, 2005.

[KTT89] Keefe, Thomas F., Bhavani M. Thuraisingham, and Wei-Tek Tsai: *Secure query-processing strategies.* IEEE Computer, 22(3):63–70, 1989.

[LDS+90] Lunt, Teresa F., Dorothy E. Denning, Roger R. Schell, Mark Heckman, and William R. Shockley: *The SeaView security model.* IEEE Transactions on Software Engineering, 16(6):593–607, 1990.

[LLV07] Li, Ninghui, Tiancheng Li, and Suresh Venkatasubramanian: *t-closeness: privacy beyond k-anonymity and ℓ-diversity.* In *Proceedings of the 23rd International Conference on Data Engineering, ICDE 2007*, pages 106–115. IEEE Computer Society, 2007.

[LR+10] Lorentz, Diana, Mary Beth Roeser, *et al.*: *Oracle Database SQL Language Reference, 11g Release 2 (11.2).* E17118-03, Oracle Corporation, 2010.

[Mai80] Maier, David: *Minimum covers in the relational database model.* Journal of the ACM, 27(4):664–674, 1980.

[MGKV07] Machanavajjhala, Ashwin, Johannes Gehrke, Daniel Kifer, and Muthuramakrishnan Venkitasubramaniam: *ℓ-diversity: privacy beyond k-anonymity.* ACM Transactions on Knowledge Discovery from Data, 1(1), 2007.

[MS07] Miklau, Gerome and Dan Suciu: *A formal analysis of information disclosure in data exchange.* Journal of Computer and System Sciences, 73(3):507–534, 2007.

[ND96] Nicomette, Vincent and Ives Deswarte: *A multilevel security model for distributed object systems.* In Bertino, Elisa, Helmut Kurth, Giancarlo Martella, and Emilio Montolivo (editors): *Computer Security – ESORICS 1996, Proceedings of the 4th European Symposium on Research in Computer Security*, volume 1146 of *Lecture Notes in Computer Science*, pages 80–98. Springer, 1996.

[OvS94] Olivier, Martin S. and Sebastiaan H. von Solms: *A taxonomy for secure object-oriented databases.* ACM Transactions on Database Systems, 19(1):3–46, 1994.

[PSPS05] Power, David J., Mark Slaymaker, Eugenia A. Politou, and Andrew C. Simpson: *Protecting sensitive patient data via query modification.* In Haddad, Hisham, Lorie M. Liebrock, Andrea Omicini, and Roger L. Wainwright (editors): *Proceedings of the 20th ACM Symposium on Applied Computing, SAC 2005*, pages 224–230. ACM, 2005.

[San93] Sandhu, Ravi S.: *Lattice-based access control models.* IEEE Computer, 26(11):9–19, 1993.

[SD01] Samarati, Pierangela and Sabrina De Capitani di Vimercati: *Access control: policies, models, and mechanisms.* In Focardi, Riccardo and Roberto Gorrieri (editors): *Foundations of Security Analysis and Design, Tutorial Lectures, FOSAD 2000*, volume 2171 of *Lecture Notes in Computer Science*, pages 137–196. Springer, 2001.

[SdJvdR83] Sicherman, George L., Wiebren de Jonge, and Reind P. van de Riet: *Answering queries without revealing secrets.* ACM Transactions on Database Systems, 8(1):41–59, 1983.

[SJ92] Sandhu, Ravi S. and Sushil Jajodia: *Polyinstantiation for cover stories.* In Deswarte, Yves, Gérard Eizenberg, and Jean Jacques Quisquater (editors): *Computer Security – ESORICS 1992, Proceedings of the 2nd European Symposium on Research in Computer Security*, volume 648 of *Lecture Notes in Computer Science*, pages 307–328. Springer, 1992.

[SÖ91] Su, Tzong-An and Gultekin Özsoyoglu: *Controlling FD and MVD inferences in multilevel relational database systems.* IEEE Transactions on Knowledge and Data Engineering, 3(4):474–485, 1991.

[SQL08] *Database Language SQL.* ISO/IEC 9075:2008, International Organization for Standardization, 2008.

[SS07] Stouppa, Phiniki and Thomas Studer: *A formal model of data privacy.* In Virbitskaite, Irina and Andrei Voronkov (editors): *Perspectives of Systems Informatics, Proceedings of the 6th International Andrei Ershov Memorial Conference, PSI 2006*, volume 4378 of *Lecture Notes in Computer Science*, pages 400–408. Springer, 2007.

[Sti94] Stickel, Mark E.: *Elimination of inference channels by optimal upgrading.* In *Proceedings of the 15th IEEE Symposium on Security and Privacy*, pages 168–174. IEEE Computer Society, 1994.

[SW74] Stonebraker, Michael and Eugene Wong: *Access control in a relational data base management system by query modification.* In *Proceedings of the ACM Annual Conference*, pages 180–186. ACM, 1974.

[Swe02] Sweeney, Latanya: *k-anonymity: a model for protecting privacy.* International Journal on Uncertainty, Fuzziness and Knowledge-Based Systems, 10(5):557–570, 2002.

[Wie10] Wiese, Lena: *Keeping secrets in possibilistic knowledge bases with necessity-valued privacy policies.* In Hüllermeier, Eyke, Rudolf Kruse, and Frank Hoffmann (editors): *Computational Intelligence for Knowledge-Based Systems Design, Proceedings of the 13th International Conference on Information Processing and Management of Uncertainty, IPMU 2010*, volume 6178 of *Lecture Notes in Artificial Intelligence*, pages 655–664. Springer, 2010.

[YL98a] Yip, Raymond W. and Karl N. Levitt: *Data level inference detection in database systems.* In *Proceedings of the 11th IEEE Computer Security Foundations Workshop, CSFW 1998*, pages 179–189, 1998.

[YL98b] Yip, Raymond W. and Karl N. Levitt: *The design and implementation of a data level database inference detection system.* In Jajodia, Sushil (editor): *Database Security XII: Status and Prospects, Proceedings of the 12th Annual IFIP WG 11.3 International Working Conference on Database Security, DBSec 1998*, volume 142 of *IFIP Conference Proceedings*, pages 253–266. Kluwer, 1998.

[ZJB08] Zhang, Lei, Sushil Jajodia, and Alexander Brodsky: *Simulatable binding: beyond simulatable auditing.* In Jonker, Willem and Milan Petkovic (editors): *Secure Data Management, Proceedings of the 5th VLDB Workshop, SDM 2008*, volume 5159 of *Lecture Notes in Computer Science*, pages 16–31. Springer, 2008.

Index

Danksagung

Viele Menschen haben mich auf dem Weg der Promotion begleitet und dadurch zur Entstehung dieser Dissertation beigetragen. Ich danke all diesen Menschen an dieser Stelle für ihre Unterstützung; ein paar von ihnen möchte ich aber besonders hervorheben.

An erster Stelle danke ich Joachim Biskup, der mich in seine Arbeitsgruppe aufgenommen und mir damit die Möglichkeit gegeben hat, in einem wissenschaftlichen Umfeld zu arbeiten. Bei ihm habe ich in puncto Forschung und wissenschaftliches Publizieren vieles gelernt. Seine unermüdliche Diskussionsbereitschaft einerseits und die Gewährung der notwendigen Freiräume andererseits haben das „Projekt Dissertation“ überhaupt erst möglich gemacht. Dankbar bin ich auch dafür, dass ich durch die Teilnahme an Konferenzen meine Ideen dem Fachpublikum vorstellen und ein paar schöne Teile der Welt kennen lernen durfte.

Des Weiteren danke ich Gabriele Kern-Isberner für viele konstruktive Anregungen und ihre Bereitschaft, das Zweitgutachten für meine Dissertation zu schreiben, sowie Heiko Krumm und Hubert Wagner für ihre Mitwirkung in meiner Prüfungskommission.

Eine große Rolle spielt natürlich auch die Arbeitsumgebung, in der eine Dissertation entsteht. In der Arbeitsgruppe ISSI am LS VI habe ich mich immer wohl und „gut aufgehoben“ gefühlt. Ich danke allen aktuellen und ehemaligen Kolleginnen und Kollegen für die gute Arbeitsatmosphäre, gemeinsame Kaffeerunden und spannende Pokerturniere.

Mit Sandra Wortmann durfte ich lange Zeit Büro und Dissertationsstress teilen. Für die perfekte Balance zwischen Arbeit und Ablenkung bin ich sehr dankbar.

Schließlich danke ich von ganzem Herzen meiner Frau Sabine – für ihre Liebe, ihre stetige Ermutigung, ihre Geduld und ihre Fähigkeit, mich auch immer wieder mal auf andere Gedanken zu bringen.

Dortmund, im November 2010 Jan-Hendrik Lochner