Fine Grained Access Control for Relational Databases by Abstract Interpretation

Raju Halder and Agostino Cortesi

DAIS, Università Ca' Foscari Venezia, Italy
{halder,cortesi}@unive.it

Abstract. In this paper[1], we propose an observation-based fine grained access control (OFGAC) mechanism where data are made accessible at various level of abstraction according to their sensitivity level. In this setting, unauthorized users are not able to infer the exact content of the data cell containing confidential information, while they are allowed to get partial information out of it, according to their access rights. The traditional fine grained access control (FGAC) can be seen as a special case of the OFGAC framework.

Key words: Access Control, Relational Databases, Abstract Interpretation

1 Introduction

Access control mechanism [2, 4, 7] is one of the most effective solutions to ensure the safety of the information in an information system. The granularity of the traditional access control mechanism is coarse-grained and can be applied at database or table level only. The need of more flexible business requirements and security policies mandate the use of Fine Grained Access Control (FGAC) mechanisms [8, 10, 12, 14–16] that provide the safety of database information even at lower level such as individual tuple or cell level.

In general, FGAC aims at hiding confidential information, while giving public access to non-confidential information only. In other words, FGAC policy provides two extreme views to the database information: completely public or completely hidden. However, there are many application areas where some partial or relaxed view of the confidential information is desirable. For instance, suppose a database in an online transaction system contains credit card numbers for its customers, and according to the disclosure policy, the employees of the customer-care section are able to see last four digits of the credit card numbers, whereas remaining digits must be kept secret. The traditional FGAC policy is unable to implement this type of security framework without changing the database structure (*e.g.*, by splitting the *Credit Card Number* attribute into two sub-attributes - one public, and the other private). To implement this relaxed scheme where an observer is allowed to observe specific properties or partial views of the confidential information, we introduce an Observation-based Fine

[1] The paper is a revised and extended version of [6]

Grained Access Control (OFGAC) mechanism on top of the traditional FGAC, based on the Abstract Interpretation framework [3]. We call the disclosure policy under OFGAC framework as observation-based disclosure policy.

Unlike traditional FGAC, the database information which are unauthorized under observation-based disclosure policy, are abstracted by the information at various level of abstraction representing specific properties of interests, rather than NULL or special symbols [9, 14]. The unauthorized users, thus, could not be able to infer the exact content of the sensitive cells. Different level of sensitivity of the information correspond to different level of abstraction. Less sensitive values are abstracted by the abstract values at lower level of abstraction, whereas more sensitive values are abstracted by the abstract values at higher level of abstraction. This way, we can tune different parts of the same database content according to different level of abstraction at the same time, giving rise to various observational access control for various part.

The query issued by external users will be directed to and executed over the abstract database, yielding to a sound approximation of the query results. In particular, we extend [6] by carefully discussing scenarios where queries contain aggregate functions (AVG, SUM, MIN, MAX, COUNT) or set operations (UNION, INTERSECT, MINUS). The traditional FGAC can be seen as a special case of our OFGAC framework.

The structure of the paper is as follows: Section 2 introduces the notion of observation-based disclosure policy. Section 3 discusses all possible multi-party collusion attacks under the same or different observation-based disclosure policies. In Sections 4, we show how to preserve referential integrity constraints while abstracting the database information. The query evaluation techniques under OFGAC are discussed in Section 5. In Section 6, we survey the related work in the literature, and finally, we draw our conclusions in Section 7.

2 Observation-based Disclosure Policy

The traditional fine grained disclosure policy p determines the part of the database information to be allowed to disclose while answering the queries Q. It splits any database state into two distinct parts: a public one (insensitive data) and a private one (sensitive data). Generic users are able to see the information from public part only, while the private part remains undisclosed. We denote a database state σ under disclosure policy p by a tuple $\sigma_p = \langle \sigma_h, \sigma_l \rangle$, where σ_h and σ_l represent the states corresponding to the private and public part respectively. In reality, p depends on the context in which queries are issued, for instance, the identity of the issuer, the purpose of the query, the data provider's policy etc.

Given a database state σ_p under the policy p and a query Q, the execution of Q on σ_p returns the result $\xi = [\![Q]\!](\sigma_p)$, where the private part $\sigma_h \in \sigma_p$ is masked by special symbols (for example, NULL [9] or Type-1/Type-2 variable [14]). As far as the security of the system is concerned, the disclosure policy should comply with the *non-interference* policy [11], *i.e.* the results of admissible queries should not depend on the confidential data in the database. The *non-interference* says that a variation of private (sensitive) database values does not cause any variation of the public (insensitive) view (see Definition 1).

Definition 1 (Non-interference). *Let $\sigma_p = \langle \sigma_h, \sigma_l \rangle$ and $\sigma'_p = \langle \sigma'_h, \sigma'_l \rangle$ be two database states under the disclosure policy p. The non-interference policy says that*

$$\forall Q, \forall \sigma_p, \sigma'_p : \quad \sigma_l = \sigma'_l \implies [\![Q]\!](\sigma_p) = [\![Q]\!](\sigma'_p)$$

That is, if the public (insensitive) part of any two database states under disclosure policy p are the same, the execution of any admissible query Q over σ_p and σ'_p return the same results.

In the context of information flow security, the notion of non-interference is too restrictive and impractical in some real systems where intensional leakage of the information to some extent is allowed with the assumption that the power of the external observer is bounded. Thus, we need to weaken or downgrading the sensitivity level of the database information, hence, the notion of non-interference which considers weaker attacker model. The weaker attacker model characterizes the observational characteristics of the attacker and can be able to observe specific properties of the private data.

Definition 2 defines the observation-based disclosure policy under the OFGAC framework.

Definition 2 (Observation-based Disclosure Policy). *Given a domain of observable properties D, and an abstraction function $\alpha_D : \wp(val) \rightarrow D$, an observation-based disclosure policy op assigned to the observer O is a tagging that assigns each value v in the database state σ a tag $\alpha_D(v) \in D$, meaning that O is allowed to access the value $\alpha_D(v)$ instead of its actual value v.*

Example 1. Consider the database of Table 1, where the cells containing sensitive information are marked with '*N*' within parenthesis. In OFGAC framework, we

Table 1: Concrete Database

(a) "*emp*"

eID	*Name*	*Age*	*Dno*	*Sal*
1	Matteo (N)	30	2	2800 (N)
2	Pallab (N)	22	1	1500
3	Sarbani (N)	56 (N)	2	2300
4	Luca (N)	35	1	6700 (N)
5	Tanushree (N)	40 (N)	3	4900
6	Andrea (N)	52 (N)	1	7000 (N)
7	Alberto (N)	48	3	800
8	Mita (N)	29 (N)	2	4700 (N)

(b) "*dept*"

Dno	*Name*	*Loc*	*Phone*	*DmngrID*
1	Financial	Venice	111-1111 (N)	6
2	Research	Rome	222-2222 (N)	8
3	Admin	Treviso	333-3333 (N)	3

abstract these sensitive information by the abstract values representing specific properties of interests as depicted in Table 2. Observe that in $emp^\sharp$, the ages are abstracted by the elements from the domain of intervals, the salaries are abstracted by their relative measures: *Low*, *Medium*, *High*, *Very High*, and the

Table 2: Partially Abstract Database

(a) "$emp^\sharp$"

$eID^\sharp$	$Name^\sharp$	$Age^\sharp$	$Dno^\sharp$	$Sal^\sharp$
1	Male	30	2	Medium
2	Male	22	1	1500
3	Female	[50, 59]	2	2300
4	Male	35	1	Very high
5	Female	[40, 49]	3	4900
6	Male	[50, 59]	1	Very high
7	Male	48	3	800
8	Female	[20, 29]	2	High

(b) "$dept^\sharp$"

$Dno^\sharp$	$Name^\sharp$	$Loc^\sharp$	$Phone^\sharp$	$DmngrID^\sharp$
1	Financial	Venice	$\top$	6
2	Research	Rome	$\top$	8
3	Admin	Treviso	$\top$	3

names are abstracted by their sex properties. It is worthwhile to mention here that we assume employees' salaries to be more sensitive than their ages, and thus, we abstract salary values with a higher level of abstraction than the age values, although both are numeric. Most importantly, in the abstract table $dept^\sharp$, since the phone numbers of all departments are strictly confidential, they are abstracted by the top element $\top$ of their corresponding abstract lattice. We call the resulting database as Partially Abstract Database, in contrast to Fully Abstract Database, since only a subset of the database information is abstracted. The correspondence between the concrete values of salaries and the abstract values that partially hide sensitive salary values can formally be expressed by the abstraction and concretization functions α_{sal} and γ_{sal} respectively as follows:

$$\alpha_{sal}(\{\mu\}) \triangleq \begin{cases} Low \text{ if } 500 \leq \mu \leq 1999 \\ Medium \text{ if } 2000 \leq \mu \leq 3999 \\ High \text{ if } 4000 \leq \mu \leq 5999 \\ Very\ High \text{ if } 6000 \leq \mu \leq 10000 \end{cases}$$

$$\gamma_{sal}(d) \triangleq \begin{cases} \{x \ : \ 500 \leq x \leq 1999\} \text{ if } d = Low \\ \{x \ : \ 2000 \leq x \leq 3999\} \text{ if } d = Medium \\ \{x \ : \ 4000 \leq x \leq 5999\} \text{ if } d = High \\ \{x \ : \ 6000 \leq x \leq 10000\} \text{ if } d = Very\ High \end{cases}$$

Formally, the sensitive values of the data cells belonging to an attribute x are abstracted by following the Galois Connection $(\wp(D_x^{con}), \alpha_x, \gamma_x, D_x^{abs})$ [3], where $\wp(D_x^{con})$ and D_x^{abs} represent the powerset of concrete domain of x and the abstract domain of x respectively, whereas α_x and γ_x represent the corresponding abstraction and concretization functions (denoted $\alpha_x : \wp(D_x^{con}) \rightarrow D_x^{abs}$ and $\gamma_x : D_x^{abs} \rightarrow \wp(D_x^{con})$) respectively. In case of insensitive information, the abstraction and concretization functions represents the identity function id.

Given a concrete database state σ_{op} under observation-based disclosure policy op, the abstract state is obtained by performing $\sigma_{op}^\sharp = \alpha(\sigma_{op})$ where the abstraction function α can be expressed as collection of abstraction functions for all attributes in the database.

Definition 3 (Abstract Database). *Let σ_{op} be a database state under the observation-based disclosure policy op. The database state $\sigma^{\sharp}_{op} = \alpha(\sigma_{op})$ where α is the abstraction function, is said to be abstract version of σ_{op} if there exists a representation function γ, called concretization function such that for all tuple $\langle x_1, x_2, \ldots, x_n \rangle \in \sigma_{op}$ there exists a tuple $\langle y_1, y_2, \ldots, y_n \rangle \in \sigma^{\sharp}_{op}$ such that $\forall i \in [1 \ldots n]$, $(x_i \in id(y_i) \vee x_i \in \gamma(y_i))$.*

Observe that the traditional FGAC [9,13,16] is a special case of our OFGAC framework where each unauthorized cell is abstracted by the top element $\top$ of its corresponding abstract lattice.

3 Collusion Attacks

Wang et al. in [14] illustrate the security of the FGAC system in case of collusion and multi-query attacks. They define the security aspect in the context of *one-party single-query/weak security* and *multi-party multi-query/strong security*, and prove that the system with weak-security is also secure under strong-security.

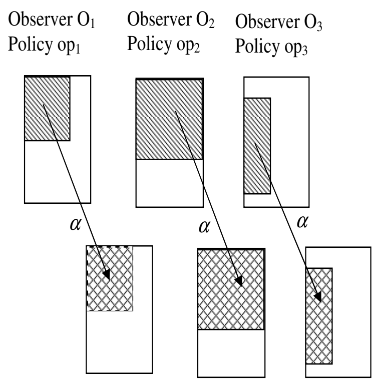
In OFGAC, transforming the system into an abstract domain means transforming the attackers, and the attackers are modeled by abstractions. The robustness of the database under the OFGAC policy depends on the ability of the external observers to distinguish the database states based on the observable properties of the query results. Here we consider three different scenarios: Figure 1(a), 1(b) and 1(c) illustrates these three cases where the shaded portions indicate the sensitive information and α ($\alpha_i \neq \alpha_j$ if $i \neq j$) is the abstraction function used to abstract those sensitive information.

Case 1: Multiple Policies/Single Level Abstraction: Suppose each of the n observers under $op_1, op_2, \ldots, op_n$ respectively issues a query Q. Let σ be a database state without any policy. Under op_i, $i = 1, \ldots, n$, the database state σ is represented by σ_{op_i} and is abstracted into $\sigma^{\sharp}_{op_i} = \alpha(\sigma_{op_i})$. Therefore, observer O_i under op_i will get the query result $\xi^{\sharp}_i = [\![Q^{\sharp}]\!](\sigma^{\sharp}_{op_i})$, where $Q^{\sharp}$ is the abstract version of Q. When these n users collude, they feed the query results $\xi^{\sharp}_i$, $i = \ldots, n$, to a function f which can perform some comparison or computation (*viz*, difference operation) among the results and infer about some sensitive information for some observers.

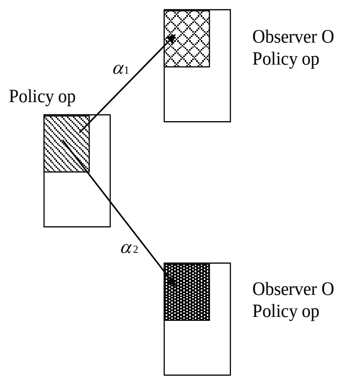
For instance, suppose a portion of database information is sensitive under policy op_j, while it is insensitive under another policy op_k, $j \neq k$. In the former case, this part of information will be abstracted, while in the latter case it will not. Thus, if this portion of information appears in both of the query results $\xi^{\sharp}_j$ and $\xi^{\sharp}_k$, then it is possible for the j^{th} observer to infer the exact content of that portion of information as it is not abstracted in ξ_k.

Let $\sigma_{op} = \langle \sigma_l, \sigma_h \rangle$ and $\sigma_{op'} = \langle \sigma'_l, \sigma'_h \rangle$ be the database states under two different policies op and op'. The database state $\sigma_{op \bullet op'}$ obtained by combining two policies op and op' are defined as follows:

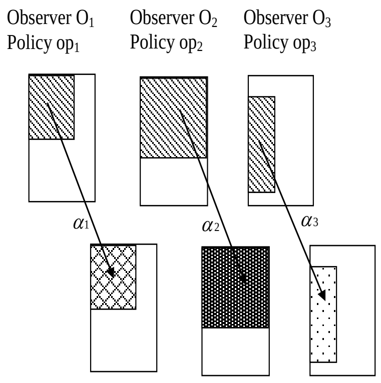
$$\sigma_{op \bullet op'} = \langle ((\sigma_l \cup \sigma_h) - (\sigma_h \cap \sigma'_h)), (\sigma_h \cap \sigma'_h) \rangle$$



(a) Multiple policies/Single abstraction



(b) Single policy/Multiple abstraction



(c) Multiple policies/Multiple abstraction

Fig. 1: Policies and Observations

This fact is depicted in Figure 2. So, when the observers under *op* and *op′* collude and share the query results, both will act as equivalent to the observer under the policy $op \bullet op'$ and thus they can infer the values belonging to the public part of $op \bullet op'$, *i.e.*, $((\sigma_l \cup \sigma_h) - (\sigma_h \cap \sigma'_h))$ by issuing a sequence of queries individually and by comparing the results together.

Case 2: Single Policy/Multiple Level Abstraction: Consider n different observers $O_1, O_2, \ldots, O_n$ under the same policy *op* and the sensitive information part is abstracted to different level of abstraction to different observers. Higher levels of abstraction make the database information less precise, whereas lower levels of abstraction represent them with more precision. Thus, the result of a query for the one with higher abstraction contains less precise information than that with lower abstraction.

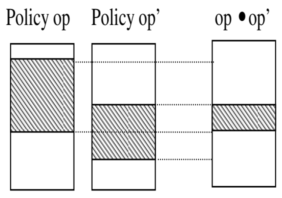


Fig. 2: Combination of policies

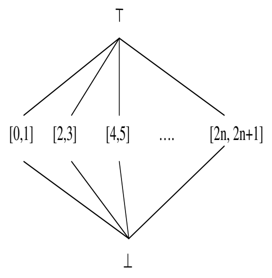
Consider two different observers O_1 and O_2 under *op* where the sensitive database information of σ_{op} are abstracted by the domains of abstraction D_1^{abs} and D_2^{abs}, yielding to $\sigma_{op}^{1\sharp}$ and $\sigma_{op}^{2\sharp}$ respectively.

First consider the case where D_2^{abs} is a more abstract domain than D_1^{abs}, *i.e.*, D_2^{abs} is an abstraction of D_1^{abs}. Since both observers are under the same policy, the query results over $\sigma_{op}^{1\sharp}$ and $\sigma_{op}^{2\sharp}$ may contain some common abstract information - one from D_1^{abs} and other from D_2^{abs}. Thus when O_1 and O_2 collude, it is possible for O_2 to obtain sensitive information with lower level of abstraction from the result obtained by O_1 as it is abstracted with lower level of abstraction for O_1. But no real collusion may arise in this case, as the overall information available to O_1 and O_2 together is at most as precise as the one already available to O_1.

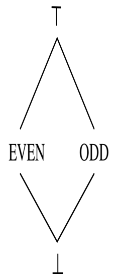
The other case is where the two domains are not one the abstraction of the other. For example, let in a particular database state an attribute of a table has the sensitive values represented by an ordered list $\langle 5, 0, 2, 3, 1\rangle$. Suppose the observer O_1 is limited by the property DOM represented by domain of intervals as shown in Figure 3(a), while O_2 is limited by parity property represented by the abstract domain PAR=$\{\bot$, EVEN, ODD, $\top\}$ as depicted in Figure 3(b). Thus O_1 sees $\langle[4, 5], [0, 1], [2, 3], [2, 3], [0, 1]\rangle$, while O_2 sees $\langle$ODD, EVEN, EVEN, ODD, ODD$\rangle$. When O_1 and O_2 collude, they can infer the exact values for the attribute, *i.e.*, $\langle 5, 0, 2, 3, 1\rangle$ by combining the query results. The corresponding combined lattice obtained by reduced product [3] of the above two abstract lattices DOM and PAR is shown in Figure 4(a).

Given an OFGAC under *Single policy/Multiple level abstraction* scenario where same information under the same policy is abstracted by n different level of abstraction to n different observers. Such OFGAC is *collusion-prone* when intersection of the sets (not singletons) obtained by concretizing abstract values of any common sensitive cell appearing in different query results for different observers, yield to a singleton. This is depicted in Definition 4.

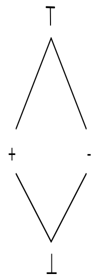
We now show an example where no collusion takes place in practice. Consider two observers O_1 and O_2, where O_1 is limited by the sign property depicted in Figure 3(c), whereas O_2 is limited by the parity property depicted in Figure 3(b). Let $\langle$-2, 0, 2, -1, 1$\rangle$ be a list of sensitive values appearing in the results of the queries issued by both O_1 and O_2. Thus, O_1 sees $\langle$-, +, +, -, +$\rangle$, while O_2 sees $\langle$EVEN, EVEN, EVEN, ODD, ODD$\rangle$. When O_1 and O_2 collude, they can infer the values as $\langle$EVEN$^-$, EVEN$^+$, EVEN$^+$, ODD$^-$, ODD$^+\rangle$ by combining the query results. However, although these combined abstract values represent more precise information than that of the individual result, the observer still could not be able to infer the exact content. Figure 4(b) shows the combined abstract lattice obtained by reduced product [3] of two abstract lattices SIGN and PAR.



(a) DOM

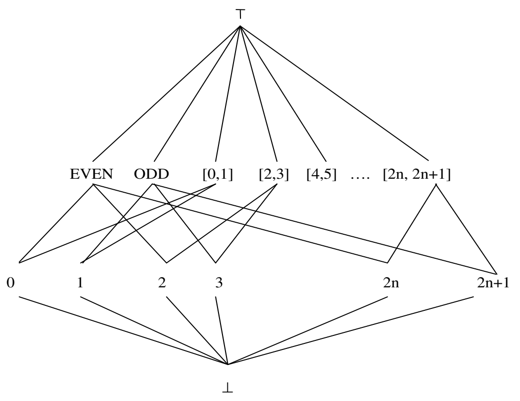


(b) PAR

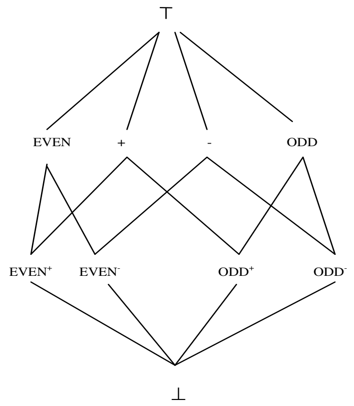


(c) SIGN

Fig. 3: Abstract Lattices of DOM, PAR and SIGN



(a) Combined lattice of DOM and PAR



(b) Combined lattice of SIGN and PAR

Fig. 4: Combination of lattices

Definition 4. *An OFGAC under Single policy/Multiple level abstraction scenario is collusion-prone, if the OFGAC uses n different abstract domains* $D_1^{abs}, \dots, D_n^{abs}$ *for n different observers and* $\exists\{d_i, \dots, d_j\} \in D_i^{abs} \times \cdots \times D_j^{abs}$ *for* $\{i, \dots, j\} \subseteq \{1, \dots, n\}$ *such that* $\bigcap_{k \in \{i,\dots,j\}} \gamma(d_k) = \{e\}$ *while* $\forall k \in \{i, \dots, j\}$, $\gamma(d_k) \neq \{e\}$.

Theorem 1. *An OFGAC using n different abstract domains $D_1^{abs}, \ldots, D_n^{abs}$ for n different observers under the same policy is collusion-prone if the reduced product [3] of $\{D_i^{abs}, \ldots, D_j^{abs}\} \subseteq \{D_1^{abs}, \ldots, D_n^{abs}\}$ is isomorphic to a concrete domain D.*

Case 3: Multiple Policies /Multiple Level Abstractions: This is the combination of the previous two cases. Observers may collude to act as the observer under the combination of their individual policies, or may try to infer about the confidential information appearing in the query results by combining (*e.g.*, intersecting) their domain of abstract values.

4 Preserving Referential Integrity Constraints between Database Relations under OFGAC

We know that the uniqueness of the values in primary or foreign key attributes is a crucial requirement in order to maintain the secure linking or referential integrity among database relations. When sensitive cells in primary or foreign key attributes are abstracted in OFGAC framework, the uniqueness of those abstract values are put under threat. As a result of this, the referential integrity constraints among database relations might be hampered. In [14], Wang et al. used Type-2 variable in order to keep these referential integrity constraints intact while masking operation is performed. The attribute values which are sensitive and act as primary key or foreign key are masked by the Type-2 variables.

We extend the same approach of Wang et al. [14] in our OFGAC framework. We denote the Type-2 variable by a pair (v, A) where A is an abstract value and $\gamma(A)$ is the domain of the variable v.

Definition 5 (Type-2 Variable). *A Type-2 variable is represented by a pair (v, A) where A is an abstract value and $\gamma(A)$ is the domain of the variable v. Given v_1, v_2, A_1, A_2 where v_1 is assumed to be different from v_2, then we have that (i) "$(v_1, A_1) = (v_1, A_1)$" and "$(v_1, A_1) \neq (v_2, A_1)$" are always true, (ii) "$(v_1, A_1) \neq (v_2, A_2)$" is true if $\gamma(A_1) \cap \gamma(A_2) = \emptyset$, (iii) "$(v_1, A_1) = (v_2, A_2)$" is $\top$ if $\gamma(A_1) \cap \gamma(A_2) \neq \emptyset$, and (iv) "$(v_1, A_1)=c$" is $\top$ where $c \in \gamma(A_1)$.*

Given two sensitive values x_1 and x_2 under the same primary/foreign key attribute. According to Definition 5, we abstract them as follows: (*i*) if $x_1 = x_2$ and $\alpha(x_1) = \alpha(x_2) = A$, then both x_1 and x_2 are abstracted by the Type-2 variable (v, A), (*ii*) if $x_1 \neq x_2$ and $\alpha(x_1) = \alpha(x_2) = A$, then x_1 and x_2 are abstracted by (v_1, A) and (v_2, A) respectively, (*iii*) if $x_1 \neq x_2$, $\alpha(x_1) = A_1$, $\alpha(x_2) = A_2$ and $\gamma(A_1) \cap \gamma(A_2) = \emptyset$, then x_1 and x_2 are abstracted by (v_1, A_1) and (v_2, A_2) respectively.

Example 2. Consider the supplier-parts database and its abstract version under an observation-based access control policy, depicted in Table 3. The attributes *S-id* and *P-id* are the primary keys of the tables "*Supplier*" and "*Part*" respectively, whereas the composite attribute {*S-id*, *P-id*} is used as the primary key of the table "*Supp-Part*". Observe that *S-id* and *P-id* in "*Supp-Part*" are used as the foreign keys that link to the primary keys of "*Supplier*" and "*Part*" respectively, and relate the suppliers with the parts sold by them. Suppose according to the disclosure policy, all values of the attributes *S-id*, *P-id* and some values of *QTY* in "*Supp-Part*" are confidential (marked with 'N' in the parenthesis). If we abstract these values by the abstract values from the domain of intervals, we may loose the ability to identify the tuples uniquely and the secure linking between "*Supplier*" and "*Part*" might be disturbed. To preserve the uniqueness of the values in abstract domain, we use Type-2 variable, as depicted in the abstract tables "*Supplier*$^\sharp$", "*Part*$^\sharp$" and "*Supp-Part*$^\sharp$". Observe that since the attribute *QTY* is not primary key or foreign key, we abstract them only by the abstract values from the domain of intervals.

Table 3: Preserving Referential Integrity Constraint by using Type-2 variable

(a) "*Supplier*"

S-id	*Name*	*Age*
S230 (N)	Alice	24
S201 (N)	Bob	21
S368 (N)	Tea	22

(b) "*Supp – Part*"

S-id	*P-id*	*QTY*
S230 (N)	P140 (N)	120
S201 (N)	P329 (N)	260 (N)
S230 (N)	P563 (N)	200
S368 (N)	P329 (N)	450 (N)
S368 (N)	P140 (N)	430 (N)

(c) "*Part*"

P-id	*Pname*
P140 (N)	Screw
P329 (N)	Bolt
P563 (N)	Nut

(d) "*Supplier*$^\sharp$"

S-id$^\sharp$	*Name*$^\sharp$	*Age*$^\sharp$
$(v_1, [S200, S249])$	Alice	24
$(v_2, [S200, S249])$	Bob	21
$(v_3, [S350, S399])$	Tea	22

(e) "*Supp – Part*$^\sharp$"

S-id$^\sharp$	*P-id*$^\sharp$	*QTY*$^\sharp$
$(v_1, [S200, S249])$	$(v_4, [P100, P149])$	120
$(v_2, [S200, S249])$	$(v_5, [P300, P349])$	[250,299]
$(v_1, [S200, S249])$	$(v_6, [P550, P599])$	200
$(v_3, [S350, S399])$	$(v_5, [P300, P349])$	[450, 499]
$(v_3, [S350, S399])$	$(v_4, [P100, P149])$	[400, 449]

(f) "*Part*$^\sharp$"

P-id$^\sharp$	*Pname*$^\sharp$
$(v_4, [P100, P149])$	Screw
$(v_5, [P300, P349])$	Bolt
$(v_6, [P550, P599])$	Nut

5 Query Evaluation under OFGAC

A general framework for Abstract Interpretation of Relational Databases has been introduced in [5]. Here, we briefly recall some notions on query abstraction, and we extend them by considering queries on multiple abstractions as well.

Example 3. Consider the concrete database of Table 1 and the corresponding partially abstract database depicted in Table 2 under the observational disclosure policy *op*. Suppose an external user issues a query Q_1 under *op* as below:

$$Q_1 = \texttt{SELECT * FROM}\ emp\ \texttt{WHERE}\ Sal > 4800;$$

The system transforms Q_1 into the corresponding abstract version[2] (denoted $Q_1^\sharp$) as shown below:

$$Q_1^\sharp = \texttt{SELECT * FROM}\ emp^\sharp\ \texttt{WHERE}\ Sal^\sharp > 4800\ \texttt{OR}\ Sal^\sharp >^\sharp High;$$

The result of $Q_1^\sharp$ on $emp^\sharp$ is depicted in Table 4.

[2] We use apex $\sharp$ to denote abstract functions/states/values.

Table 4: $\xi_1^\sharp$: The result of the query $Q_1^\sharp$

$eID^\sharp$	$Name^\sharp$	$Age^\sharp$	$Dno^\sharp$	$Sal^\sharp$
4	Male	35	1	Very high
5	Female	[40, 49]	3	4900
6	Male	[50, 59]	1	Very high
8	Female	[20, 29]	2	High

We denote any SQL command Q by a tuple $Q \triangleq \langle A_{sql}, \phi \rangle$. We call the first component A_{sql} the *action part* and the second component ϕ the *pre-condition part* of Q. In an abstract sense, Q first identifies an active data set from the database using the pre-condition ϕ and then performs the appropriate operations on that data set using the SQL action A_{sql}. The pre-condition ϕ appears in SQL commands as a well-formed formula in first-order logic [5]. In concrete domain, ϕ evaluates to *true* for the active data set on which A_{sql} operates, whereas in abstract domain the abstract pre-condition $\phi^\sharp$ evaluates to *true* or $\top$. The logic value $\top$ indicates that the tuple may or may not satisfy the semantic structure of $\phi^\sharp$. For instance, in Example 3, $\phi^\sharp$ (represented by WHERE clause) evaluates to *true* for the first three tuples in the result $\xi_1^\sharp$, whereas it evaluates to $\top$ (may be *true* or may be *false*) for the last tuple. The result of $Q_1^\sharp$ is *sound* [5] as it over-approximate the result of the query Q_1. Observe in fact that $Q_1^\sharp$ includes also the "false positive" corresponding to the concrete information about *Mita*.

5.1 Query Evaluation in presence of Aggregate Functions

Let $T_\phi^\sharp$ be a set of abstract tuples for which $\phi^\sharp$ evaluates to *true* or $\top$. The application of abstract GROUP BY function on $T_\phi^\sharp$ based on a sequence of abstract expressions, yields to a set of abstract groups $\{G_1^\sharp, G_2^\sharp, \ldots, G_n^\sharp\}$. When no abstract GROUP BY function appears in SELECT statement, we assume $T_\phi^\sharp$ as a single abstract group.

Given an abstract group $G^\sharp$, we can partition the tuples in $G^\sharp$ into two parts: $G_{yes}^\sharp$ for which $\phi^\sharp$ evaluates to *true*, and $G_{may}^\sharp$ for which $\phi^\sharp$ evaluates to $\top$. Thus we can write $G^\sharp = G_{yes}^\sharp \cup G_{may}^\sharp$, where $G_{yes}^\sharp \cap G_{may}^\sharp = \emptyset$.

Let $s^\sharp$ be an abstract aggregate function and $e^\sharp$ be an abstract expression. To ensure the soundness, the computation of $s^\sharp(e^\sharp)$ on $G^\sharp$ is defined as follows: $s^\sharp(e^\sharp)[G^\sharp] = [min^\sharp(a^\sharp), max^\sharp(b^\sharp)]$, where $a^\sharp = fn^\sharp(e^\sharp)[G_{yes}^\sharp]$ and $b^\sharp = fn^\sharp(e^\sharp)[G^\sharp]$.

By $fn^\sharp(e^\sharp)[G_{yes}^\sharp]$ and $fn^\sharp(e^\sharp)[G^\sharp]$ we mean that the function $fn^\sharp$ is applied on the set of abstract values obtained by evaluating $e^\sharp$ over the tuples in $G_{yes}^\sharp$ and $G^\sharp$ respectively, yielding to an abstract aggregate value as result. For instance, in case of $s^\sharp = AVG^\sharp$, the corresponding $fn^\sharp(e^\sharp)[G^\sharp]$ returns average of the set of values obtained by evaluating $e^\sharp$ over the set of tuples $G^\sharp$. The computation of

$fn^\sharp$ is defined differently by considering two different situations that can arise: (*i*) when the primary key is abstracted, yielding two or more tuples mapped into a single abstract tuple, and (*ii*) when the primary key is not abstracted and the identity of each tuples are preserved in abstract domain. Both the functions $min^\sharp$ and $max^\sharp$ take single abstract value $a^\sharp$ and $b^\sharp$ respectively as parameters which are obtained from $fn^\sharp$, and returns a concrete numerical value as output. $min^\sharp(a^\sharp)$ returns the minimum of the set of concrete values $\gamma(a^\sharp)$ and $max^\sharp(b^\sharp)$ returns the maximum of the concrete values $\gamma(b^\sharp)$, where γ is the concretization function.

Example 4. Consider the abstract database of Table 2 and the following abstract query $Q_2^\sharp$ containing aggregate functions as follows:

$Q_2^\sharp$ =`SELECT COUNT`$^\sharp$`(*), AVG`$^\sharp$`(`$Age^\sharp$`) FROM` $emp^\sharp$
`WHERE (`$Age^\sharp$ `BETWEEN 32 AND 55) OR (`$Age^\sharp$ `BETWEEN`$^\sharp$`[30, 39] AND [50, 59]);`

The result of $Q_2^\sharp$ on $emp^\sharp$ is depicted in Table 5. In the example, the evaluation

Table 5: $\xi_2^\sharp$: The result of the query $Q_2^\sharp$

COUNT$^\sharp$(*)	AVG$^\sharp$($Age^\sharp$)
[3, 5]	[41, 50]

of the abstract WHERE clause extracts five tuples in total where three tuples with $eID^\sharp$ equal to 4, 5, 7 belong to $G_{yes}^\sharp$, whereas two tuples with $eID^\sharp$ equal to 3, 6 belong to $G_{may}^\sharp$. Thus, in case of AVG$^\sharp$($Age^\sharp$), we get $a^\sharp = fn^\sharp(\{35, [40, 49], 48\}) = average^\sharp(\{35, [40, 49], 48\})=[41, 44]$ and $b^\sharp = fn^\sharp(\{[50, 59], 35, [40, 49], [50, 59], 48\}) = average^\sharp(\{[50, 59], 35, [40, 49], [50, 59], 48\}) = [44, 50]$. Hence, AVG$^\sharp$($Age^\sharp$) $= [min^\sharp(a^\sharp), max^\sharp(b^\sharp)]=[41, 50]$. Similarly, in case of COUNT$^\sharp$(*), we get $a^\sharp = count^\sharp(G_{yes}^\sharp)=[3, 3]$ and $b^\sharp = count^\sharp(G^\sharp) = [5, 5]$. Thus, COUNT$^\sharp$(*) =[3, 5]. Observe that the result is sound, *i.e.*, $\xi_2 \in \gamma(\xi_2^\sharp)$ where ξ_2 is the result of a concrete query $Q_2 \in \gamma(Q_2^\sharp)$.

5.2 Query Evaluation in presence of UNION, INTERSECT, MINUS

Given an abstract query $Q^\sharp$ and an abstract database state $\sigma_{op}^\sharp$ under an observation-based policy op, the result of $Q^\sharp$ can, thus, be denoted by a tuple

$$\xi^\sharp = [\![Q^\sharp]\!](\sigma_{op}^\sharp) = \langle \xi_{yes}^\sharp, \xi_{may}^\sharp \rangle$$

where $\xi_{yes}^\sharp$ is the part of the query result for which semantic structure of $\phi^\sharp$ evaluates to *true* and $\xi_{may}^\sharp$ represents the remaining part for which $\phi^\sharp$ evaluates to $\top$. Observe that we assume $\xi_{yes}^\sharp \cap \xi_{may}^\sharp = \emptyset$. For example, in the query result of $Q_1^\sharp$ in Table 4, the first three tuples belong to $\xi_{yes}^\sharp$, whereas the last tuple belong to $\xi_{may}^\sharp$.

Consider an abstract query of the form $Q^\sharp = Q^\sharp_l$ $\texttt{MINUS}^\sharp$ $Q^\sharp_r$. Let the result of $Q^\sharp_l$ and $Q^\sharp_r$ be $\xi^\sharp_l = (\xi^\sharp_{yes_l}, \xi^\sharp_{may_l})$ and $\xi^\sharp_r = (\xi^\sharp_{yes_r}, \xi^\sharp_{may_r})$ respectively. The abstract difference operation $\texttt{MINUS}^\sharp$ is defined as follows:

$$\begin{aligned}\xi^\sharp &= \xi^\sharp_l \ \texttt{MINUS}^\sharp\ \xi^\sharp_r = \langle \xi^\sharp_{yes_l}, \xi^\sharp_{may_l} \rangle \ \texttt{MINUS}^\sharp\ \langle \xi^\sharp_{yes_r}, \xi^\sharp_{may_r} \rangle \\ &= \langle \xi^\sharp_{yes_l} - (\xi^\sharp_{yes_r} \cup \xi^\sharp_{may_r}), (\xi^\sharp_{may_l} \cup \xi^\sharp_{may_r}) - \xi^\sharp_{yes_r} \rangle \qquad (1)\end{aligned}$$

Observe that the first component $(\xi^\sharp_{yes_l} - (\xi^\sharp_{yes_r} \cup \xi^\sharp_{may_r})) \in \xi^\sharp$ contains those tuples for which $\phi^\sharp$ strictly evaluates to *true*, whereas for the second component $((\xi^\sharp_{may_l} \cup \xi^\sharp_{may_r}) - \xi^\sharp_{yes_r}) \in \xi^\sharp$, $\phi^\sharp$ evaluates to $\top$.

Example 5. Consider the abstract database of Table 2 and the query $Q^\sharp_3 = Q^\sharp_l$ - $Q^\sharp_r$, where

$Q^\sharp_l$ =`SELECT * FROM` $emp^\sharp$ `WHERE` $Sal^\sharp > 2500$ `OR` $Sal^\sharp >^\sharp$ *Medium*; and

$Q^\sharp_r$ =`SELECT * FROM` $emp^\sharp$ `WHERE` $Sal^\sharp > 5500$ `OR` $Sal^\sharp >^\sharp$ *High*;

The execution of $Q^\sharp_l$ and $Q^\sharp_r$ on $emp^\sharp$ are depicted in Table 6(a) and 6(b) respectively. In Table 6(a), for the first tuple the pre-condition of $Q^\sharp_l$ evaluates to $\top$

Table 6: The result of the query $Q^\sharp_3$

(a) Query result of $Q^\sharp_l$

$eID^\sharp$	$Name^\sharp$	$Age^\sharp$	$Dno^\sharp$	$Sal^\sharp$
1	Male	30	2	Medium
4	Male	35	1	Very high
5	Female	[40, 49]	3	4900
6	Male	[50, 59]	1	Very high
8	Female	[20, 29]	2	High

(b) Query result of $Q^\sharp_r$

$eID^\sharp$	$Name^\sharp$	$Age^\sharp$	$Dno^\sharp$	$Sal^\sharp$
4	Male	35	1	Very high
6	Male	[50, 59]	1	Very high
8	Female	[20, 29]	2	High

(c) $\xi^\sharp_3$: Query result of $Q^\sharp_3$

$eID^\sharp$	$Name^\sharp$	$Age^\sharp$	$Dno^\sharp$	$Sal^\sharp$
1	Male	30	2	Medium
5	Female	[40, 49]	3	4900
8	Female	[20, 29]	2	High

(thus, belongs to $\xi^\sharp_{may_l}$), whereas for the remaining four tuples it evaluates to *true* (thus, belongs to $\xi^\sharp_{yes_l}$). Similarly, in Table 6(b), for the first two tuples the pre-condition of $Q^\sharp_r$ evaluates to *true* (hence, belongs to $\xi^\sharp_{yes_r}$), whereas for the last one it evaluates to $\top$ (hence, belongs to $\xi^\sharp_{may_r}$). Thus, the first component $(\xi^\sharp_{yes_l} - (\xi^\sharp_{yes_r} \cup \xi^\sharp_{may_r})) \in \xi^\sharp_3$ contains the tuple with $eID^\sharp$ equal to 5, and the second component $((\xi^\sharp_{may_l} \cup \xi^\sharp_{may_r}) - \xi^\sharp_{yes_r}) \in \xi^\sharp_3$ contains the tuples with $eID^\sharp$ equal to 1 and 8, as shown in Table 6(c). The result is sound, *i.e.*, $\xi_3 \in \gamma(\xi^\sharp_3)$ where ξ_3 is the result of a concrete query $Q_3 \in \gamma(Q^\sharp_3)$.

Similarly, the abstract intersection operation $\texttt{INTERSECT}^\sharp$ is defined as follows:

$$\begin{aligned}
\xi^\sharp &= [\![Q_l^\sharp \ \texttt{INTERSECT}^\sharp \ Q_r^\sharp]\!](\sigma_{op}^\sharp)\\
&= [\![Q_l^\sharp]\!](\sigma_{op}^\sharp) \ \texttt{INTERSECT}^\sharp \ [\![Q_r^\sharp]\!](\sigma_{op}^\sharp)\\
&= \xi_l^\sharp \ \texttt{INTERSECT}^\sharp \ \xi_r^\sharp\\
&= \langle \xi_{yes_l}^\sharp, \xi_{may_l}^\sharp \rangle \ \texttt{INTERSECT}^\sharp \ \langle \xi_{yes_r}^\sharp, \xi_{may_r}^\sharp \rangle\\
&= \langle (\xi_{yes_l}^\sharp \cap \xi_{yes_r}^\sharp), ((\xi_{may_l}^\sharp \cap \xi_r^\sharp) \cup (\xi_{may_r}^\sharp \cap \xi_l^\sharp)) \rangle
\end{aligned}$$

The abstract union operation $\texttt{UNION}^\sharp$ is defined as follows:

$$\begin{aligned}
\xi^\sharp &= [\![Q_l^\sharp \ \texttt{UNION}^\sharp \ Q_r^\sharp]\!](\sigma_{op}^\sharp)\\
&= [\![Q_l^\sharp]\!](\sigma_{op}^\sharp) \ \texttt{UNION}^\sharp \ [\![Q_r^\sharp]\!](\sigma_{op}^\sharp)\\
&= \xi_l^\sharp \ \texttt{UNION}^\sharp \ \xi_r^\sharp\\
&= \langle \xi_{yes_l}^\sharp, \xi_{may_l}^\sharp \rangle \ \texttt{UNION}^\sharp \ \langle \xi_{yes_r}^\sharp, \xi_{may_r}^\sharp \rangle\\
&= \langle (\xi_{yes_l}^\sharp \cup \xi_{yes_r}^\sharp), ((\xi_{may_l}^\sharp \cup \xi_{may_r}^\sharp) \backslash (\xi_{yes_l}^\sharp \cup \xi_{yes_r}^\sharp)) \rangle
\end{aligned}$$

5.3 Soundness of Query Evaluation

Suppose, σ_{op} and $\sigma_{op}^\sharp$ represent a concrete and corresponding abstract database states respectively under an observation-based disclosure policy *op*. The correspondence between σ_{op} and $\sigma_{op}^\sharp$ are described using the concretization and abstraction maps γ and α respectively. If Q and $Q^\sharp$ are representing the SQL queries on concrete and abstract domain respectively, let ξ and $\xi^\sharp$ be the results of applying Q and $Q^\sharp$ on σ_{op} and $\sigma_{op}^\sharp$ respectively. The following fact illustrate the soundness condition of abstraction:

$$\begin{tikzcd}
\sigma_{op} \arrow[r, "Q"] & \xi \sqsubseteq \gamma(\xi^\sharp) \\
\sigma_{op}^\sharp \arrow[u, "\gamma"'] \arrow[r, "Q^\sharp"'] & \xi^\sharp \arrow[u, "\gamma"']
\end{tikzcd}$$

Lemma 1. *Let $\sigma_{op}^\sharp$ be an abstract database state under an observation-based disclosure policy op and $Q^\sharp$ be an abstract query. $Q^\sharp$ is sound iff $\forall \sigma_{op} \in \gamma(\sigma_{op}^\sharp),\ \ \forall Q \in \gamma(Q^\sharp)$: $[\![Q]\!](\sigma_{op}) \subseteq \gamma([\![Q^\sharp]\!](\sigma_{op}^\sharp))$.*

Observe that the treatment of queries under our OFGAC framework extends the corresponding scenario discussed in [13, 14, 16], that can be seen as a special case when the only abstraction considered on to NULL or special variables (and the "may" part of the query results is always empty).

6 Related Works

Wang et al. [14] proposed a formal notion of correctness in fine-grained database access control. They showed why the existing approaches [9] fall short in some circumstances with respect to soundness and security requirements, like when queries contain negation operations. Moreover, they proposed a labeling approach for masking unauthorized information by using two types of special variables (Type-1 or Type-2) as well as a secure and sound query evaluation algorithm in case of cell-level disclosure policies.

In [13, 16], the authors observed that the proposed algorithm in [14] is unable to satisfy the soundness property for the queries containing the negation operations NOT IN or NOT EXISTS. They proposed an enforcing rule to control the information leakage where the query is executed on an operational relation rather than the original relation. However, although the algorithm for Enforcing Rule satisfies the soundness and security properties for all SQL queries, it would not reach the maximum property [14].

Agrawal et al. [1] introduced the syntax of a fine grained restriction command at column level, row level, or cell level. The enforcement algorithm automatically combines the restrictions relevant to individual queries annotated with purpose and recipient information, and transforms the user's query into an equivalent query over a dynamic view that implements the restriction.

In [15], the authors extended the SQL language to express the FGAC security policies. Rizvi et al. in [10] described two models for fine-grained access control: Truman and Non-Truman models. Both models support authorization-transparent querying. Unlike the Truman model, the Non-Truman model avoids the pitfalls of the query modification approach and allows a great deal of flexibility in authorization, such as authorization of aggregate results.

Kabra et al. [8] defined the circumstances when a query plan is safe with respect to user defined functions (UDFs) and other unsafe functions (USFs). They proposed techniques to generate safe query plans. However, these safe query plans may yield to un-optimized plans.

7 Conclusions

In this paper, we introduce an observation-based fine grained access control (OFGAC) framework on top of the traditional FGAC where the confidential information in the database are abstracted by their observable properties and the external observers are able to see this partial or abstract view of the confidential information rather than their exact content. The traditional FGAC can be seen as a special case of our OFGAC framework, where the confidential information are abstracted by the top element $\top$ of the corresponding abstract lattices.

Acknowledgement

Work partially supported by RAS L.R. 7/2007 Project TESLA.

References

1. Agrawal, R., Bird, P., Grandison, T., Kiernan, J., Logan, S., Rjaibi, W.: Extending relational database systems to automatically enforce privacy policies. In: Proc. of the 21st Int. Conf. on Data Engineering (ICDE '05). pp. 1013–1022. IEEE CS (2005)
2. Bertino, E., Jajodia, S., Samarati, P.: A flexible authorization mechanism for relational data management systems. ACM Transactions on Information Systems 17(2), 101–140 (1999)
3. Cousot, P., Cousot, R.: Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: Conf. Record of the 6th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '77). pp. 238–252. ACM Press, Los Angeles, CA, USA (1977)
4. Griffiths, P.P., Wade, B.W.: An authorization mechanism for a relational database system. ACM Transactions on Database Systems 1(3), 242–255 (1976)
5. Halder, R., Cortesi, A.: Abstract interpretation for sound approximation of database query languages. In: Proc. of the IEEE 7th Int. Conf. on Informatics and Systems (INFOS 2010). pp. 53–59. IEEE CFP1006J-CDR, Cairo, Egypt (2010)
6. Halder, R., Cortesi, A.: Observation-based fine grained access control for relational databases. In: Proc. of the 5th Int. Conf. on Software and Data Technologies (ICSOFT '10). pp. 254–265. INSTICC Press, Athens, Greece (2010)
7. Jajodia, S., Samarati, P., Subrahmanian, V.S., Bertino, E.: A unified framework for enforcing multiple access control policies. SIGMOD Record 26(2), 474–485 (1997)
8. Kabra, G., Ramamurthy, R., Sudarshan, S.: Redundancy and information leakage in fine-grained access control. In: Proc. of the ACM SIGMOD Int. Conf. on Management of Data (SIGMOD '06). pp. 133–144. ACM Press, Chicago, IL, USA (2006)
9. LeFevre, K., Agrawal, R., Ercegovac, V., Ramakrishnan, R., Xu, Y., DeWitt, D.: Limiting disclosure in hippocratic databases. In: Proc. of the 30th int. conf. on Very Large Data Bases (VLDB '04). pp. 108–119 (2004)
10. Rizvi, S., Mendelzon, A., Sudarshan, S., Roy, P.: Extending query rewriting techniques for fine-grained access control. In: Proc. of the ACM SIGMOD Int. Conf. on Management of Data (SIGMOD '04). pp. 551–562. ACM Press, Paris, France (2004)
11. Sabelfeld, A., Myers, A.C.: Language-based information-flow security. IEEE Journal on selected areas in Communications 21(1), 5–19 (2003)
12. Shi, J., Zhu, H.: A fine-grained access control model for relational databases. Journal of Zhejiang University - Science C 11, 575–586 (2010)
13. Shi, J., Zhu, H., Fu, G., Jiang, T.: On the soundness property for sql queries of fine-grained access control in dbmss. In: Proc. of the 8th IEEE/ACIS Int. Conf. on Computer and Information Science (ICIS '09). pp. 469–474. IEEE CS, Shanghai, China (2009)
14. Wang, Q., Yu, T., Li, N., Lobo, J., Bertino, E., Irwin, K., Byun, J.W.: On the correctness criteria of fine-grained access control in relational databases. In: Proc. of the 33rd int. conf. on Very Large Data Bases (VLDB '07). pp. 555–566. Vienna, Austria (2007)
15. Zhu, H., Lü, K.: Fine-grained access control for database management systems. In: Proc. of the 24th British National Conf. on Databases. pp. 215–223. Springer LNCS, Volume 4587, Glasgow, UK (2007)
16. Zhu, H., Shi, J., Wang, Y., Feng, Y.: Controlling information leakage of fine-grained access model in dbmss. In: Proc. of the 9th Int. Conf. on Web-Age Information Management (WAIM '08). pp. 583–590. IEEE CS, Zhangjiajie, China (2008)