2009 Eigth IEEE/ACIS International Conference on Computer and Information Science

On the Soundness Property for SQL Queries of Fine-grained Access Control in DBMSs

Jie Shi, Hong Zhu, Ge Fu, Tao Jiang
College of Computer Science & Technology
Huazhong University of Science and Technology
Wuhan, Hubei, 430074, P.R. China
{*shijie1123,whzhuhong,fuge2006*}*@gmail.com*

Abstract

The fine-grained access control approaches in DBMSs should satisfy soundness property which requires the answer of a query returned by the approach under the control of fine-grained access control is consistent with the answer when there is no fine-grained access control. However, existing techniques cannot guarantee soundness property for all SQL queries. Therefore, for an approach, there is a practical need to state that, for which kinds of SQL queries, soundness property can be guaranteed by this approach. In this paper, we present our initial effort for this objective. We firstly proposed a new algorithm with query modification. Then, we refine, extend and enhance the theory about soundness property, and state that, for which kinds of SQL queries, the soundness property is guaranteed by the proposed algorithm. Finally, we implement the algorithm using query modification and performance evaluation has also been conducted, which indicates this approach is feasible.

Keywords: database security, access control, fine-grained access control

1. Introduction

Fine-grained access control (FGAC) allows accessing to a table at the granularity of individual rows, columns and the cells within rows. In recent years, there are many new techniques which have been investigated to integrate FGAC into Database Management Systems(DBMSs) [1], [2], [3], [4], [5].

The soundness property for FGAC was proposed in [6]. An algorithm is sound if the answer returned by it is consistent with the answer when there is no FGAC, namely, the answer under the control of FGAC can not return wrong information. In [6], wang et al. pointed out that the existing approaches could not preserve soundness property, and presented that when a query contains any negation, as expressed using the keywords MINUS, NOT EXISTS or NOT IN, the approach in [7] would violate soundness property. To solve this problem, an algorithm was proposed in [6] to satisfy soundness property for these queries containing MINUS. However, the algorithm can't directly preserve the soundness property for those queries containing NOT EXISTS or NOT IN. Additional, there are still many other queries for which the algorithm dose not work well too. Let us see the following example.

EXAMPLE 1: Suppose there is a table "Employee" with four attributes: id, name, age and phone, where id is the primary key (See Table 1). There is a FGAC policy P_1. P_1 is over table "Employee" which only allows Andy to read all information of himself and the information about id and name of other person, but the whole information about the person whose id is 4 *can not be read. The FGAC policy P_1 can be defined as following:*

- *Policy P_1 over table "Employee" for Andy:*
  - *the restriction of the row-level policy: "id ≠ 4";*
  - *the restriction of the cell-level policy of attribute age: "id = 1"*
  - *the restriction of the cell-level policy of attribute phone: "id = 1"*

According to the FGAC policies, we mark each cell with "(Y)" or "(N)", indicating whether the cell is allowed by this policy. Suppose there are three queries issued by Andy:

- Q_1="SELECT *name* FROM *Employee* WHERE *phone* IS NULL";
- Q_2="SELECT *name* FROM *Employee* WHERE *id* > (SELECT COUNT(*) FROM *Employee*)";
- Q_3="SELECT *name* FROM *Employee* WHERE *name* NOT IN (SELECT name FROM *Employee* WHERE *age* > 28)";

For query Q_1, when there is no FGAC policy, the answer is {Peter}. However, under the FGAC policy P_1, the answers with the algorithms in [6], [7] are {John, Peter, Mary} which include John and Mary whose phone are not NULL. Apparently, they violate the soundness property. For query Q_2 and Q_3, they also can't hold the soundness property.

From the example above, we found that there are many queries that the algorithms in [6], [7] still do not satisfy soundness property. To the best of our knowledge, there is

1. Hong Zhu is the corresponding author

978-0-7695-3641-5/09 $25.00 © 2009 IEEE
DOI 10.1109/ICIS.2009.21

Table 1. Employee

id	name	age	phone
1(Y)	Andy(Y)	29(Y)	11111111(Y)
2(Y)	John(Y)	30(N)	22222222(N)
3(Y)	Peter(Y)	25(N)	*NULL*(N)
4(N)	Jack(N)	32(N)	44444444(N)
5(Y)	Mary(Y)	27(N)	55555555(N)

no algorithm which preserve soundness property for all SQL queries in all circumstances.

It is very difficult to find an algorithm to preserve soundness property for all queries in all situations, because SQL is an easy-to-use language with a number of high-level constructs and it allows writing very complex queries. Therefore, for an algorithm, there is a practical need to known that , for which kinds of SQL queries in all situations, soundness property can be certainly preserved. However, the definition of soundness in [6] is for all queries which is too rigorous to find an algorithm to satisfy it. Thus, a definition to define soundness for only a query or a kind of query should be presented.

The main contributions of this study are summarized as following:

- we propose an algorithm and implement it with query modification. Then we refine, extend and enhance the theory about soundness property. Additional properties are introduced using expression with relational algebra. Finally we state that, for which kinds of SQL queries, soundness property can be satisfied. It also has been theoretically proved.
- we conduct performance evaluation of our query modification approach which indicates that the cost of our query modification is small, and scalable to large databases which can not be obtained for the only existing algorithm [6] where the soundness property was considered.

The remainder of this paper is organized as follows. In Section 2, we introduce the related works. In Section 3, we propose an approach with query modification. We enrich the theory about soundness property , and state that, for which kinds of SQL queries, soundness property can be guaranteed in Section 4. Then we present experimental results of the implementation in Section 5. Finally, we conclude and present the future work in Section 6.

2. Related Work

The Virtual Private Database (VPD) in ORACLE [5] supports FGAC through functions which return predicates according to FGAC policy. When an user issues a query, this query is modified by attaching the predicates returned from the corresponding function to the WHERE clause to enforce FGAC. Sybase row level access control [4] allows users to define access control policies that restrict users to retrieve the data over the table. LeFevre et al. firstly presented a model to enforce FGAC at cell-level granularity in Hippocratic Databases [7]. Agrawal et al. proposed a framework for FGAC and they extended SQL statements to describe row level, column level and cell level access control policy [2]. Chaudhuri et al. proposed a scheme for fine-grained authorization based on adding predicates to authorization grants [1]. The scheme supports predicated authorization to specific columns, cell-level authorization with nullification, authorization for function/procedure execution, and grants with grant option. However, all works above didn't mention the soundness property of FGAC.

Wang et al. [6] proposed a formal notion of correctness for FGAC in databases, which first proposed the soundness property, and discussed why the existing approaches had limitations in some circumstances. Then they proposed a labeling approach for masking unauthorized data items and a query evaluation algorithm for FGAC in relational databases. The algorithm resolves the soundness problem caused by MINUS operation in SQL statement. However, it can't preserve soundness property for all SQL queries, and they also did not state for which kinds of SQL queries their algorithm can preserve soundness property.

3. Query Modification Algorithm

In the following, for a relation R, we use C_R to denote the set of all attributes in the relation R. Before introducing the query modification algorithm, we first introduce a definition.

DEFINITION 1 (key attribute of a relation for a query): For any query Q, R is a relation involved in query Q. For any attribute $A \in C_R$, if A belongs to the set of attributes of **WHERE** clause of Q , we say A is a key attribute of R for Q.

Let us see Q_1 in Example 1, the attribute *phone* is a *key attribute* of *Employee* for Q_1 because *phone* is in the **WHERE** clause of Q_1. For Q_3 in Example 1, *name* and *age* are the *key attributes* of *Employee* for Q_3 and *age* is the only *key attribute* of *Employee* for the subquery of Q_3.

In the following, the algorithm, which are called **key attribute based algorithm (KAB algorithm)**, would be introduced. It has mainly threes steps:

The first step: creating a temporary view for each relation involved in a SQL query statement. For a query Q, the *key attributes* can be found according to Definition 1. Then the row-level policy and cell-level policies over these key attributes can be obtained from FGAC policy. Suppose the relation involved in query is R , the row-level policy is P_{row} and the cell-level policy of key attribute is P_{cell}. A temporary view, called **operational relation**,would be created as follows:

(SELECT $*$ FROM R WHERE P_{row} and P_{cell})

The second step: using temporary views to replace these relations involved in the SQL query respectively.

The third step: each attribute in select_list of SQL query would be modified with "CASE" statement [7].

The KAB algorithm for SQL queries is introduced in detail in [2].

EXAMPLE 2: There were two SQL queries Q_1 and Q_3 under FGAC policies P_1 which were introduced in Example 1, we will use them to illustrate our modification approach.

For query Q_1, *phone* is the only *key attribute*, and the row-level policy is "$id \neq 4$", the cell-level policy of *phone* is "$id = 1$". So the temporary view is:

$$(\text{SELECT } id, name, age, phone \text{ FROM } Employee$$
$$\text{WHERE } id \neq 4 \text{ and } id = 1)$$

Because there is no cell-level policy over *name*, so in the modified query, there is no "CASE" statement to replace *name*. The final modified query $Q_{1_{modified}}$ is shown below. The query Q_3 can be modified into $Q_{3_{modified}}$ using the same approach. Suppose there is a cell-level policy over the attribute *name*: "$id = 1$", then the modified query of Q_1 is $Q_{1'_{modified}}$.

- $Q_{1_{modified}}$="SELECT *name* FROM (SELECT *id*, *name*, *age*, *phone* FROM *Employee* WHERE $id \neq 4$ and $id = 1$) WHERE *phone* IS NULL";
- $Q_{3_{modified}}$="SELECT *name* FROM (SELECT *id*, *name*, *age*, *phone* FROM *Employee* WHERE $id \neq 4$ and $id = 1$) WHERE *name* NOT IN (SELECT name FROM (SELECT *id*, *name*, *age*, *phone* FROM *Employee* WHERE $id \neq 4$ and $id = 1$) WHERE $age > 28$)"
- $Q_{1'_{modified}}$="SELECT CASE WHEN $id = 1$ THEN *name* ELSE NULL END AS *name* FROM (SELECT *id*, *name*, *age*, *phone* FROM *Employee* WHERE $id \neq 4$ and $id = 1$) WHERE *phone* IS NULL";

In [7], to preserve security, these rows which maybe break security are removed from the result by means of replacing the values of the key attributes of these rows with NULL and the evaluation rules "$NULL \neq NULL$" and "$NULL \neq c$" for any constant value c. The proposed algorithm directly remove these rows which maybe break security according to FGAC policy to create the temporary view, so the proposed approach can preserve security too.

4. Soundness Property

In this section, we will point out that, for which kinds of SQL queries, the KAB algorithm can guarantee soundness property.

The definition of soundness was firstly proposed in [6]. However, it can't be used to judge whether an algorithm is sound for a query or a kind of queries. So, it is necessary to define such notion.

4.1. Soundness Definition

Without loss of generalities, we will substitute *unauthorized value* with "Φ" which is a special symbol introduced in our model. In previous work, NULL is used to replace the *unauthorized value*. The reason of using "Φ" but NULL is that some truth values of attributes are NULL.By using "Φ", we can distinguish the truth values NULL from the values which are used to hide the truth data items. The "Φ" has the following properties:

- $\Phi \neq \Phi$;
- for any constant value c, $\Phi \neq c$;

DEFINITION 2: Given two tuples $t_x =< x_1, x_2, \cdots, x_n >$ *and* $t_y =< y_1, y_2, \cdots, y_n >$, *we say that* t_x *is* ***subsumed by*** t_y *(and write* $t_x \sqsubseteq_t t_y$*) if and only if* $\forall i \in [1 \ldots n]$ *such that* $(x_i = y_i \vee x_i = \Phi)$.

Based on the definition above, the following property is correct obviously:

PROPERY 1: $t_1 = t_2 \Rightarrow t_1 \sqsubseteq_t t_2$.

The notation can be extended to the relations as follows:

DEFINITION 3: Given two simple relations R_1 and R_2, we say that R_1 is **subsumed by** R_2 (denote as $R_1 \sqsubseteq R_2$) if and only if for $\forall\ t_1 \in R_1$, $\exists\ t_2 \in R_2$ such that $t_1 \sqsubseteq_t t_2$.

According the definitions above, the following lemmas would be proved. However, due to the limitation of pages, we omit all proof details in this paper which can be found in [8].

LEMMA 1: R_1 *and* R_2 *are two relations, such that*

$$R_1 \subseteq R_2 \Rightarrow R_1 \sqsubseteq R_2.$$

LEMMA 2: R_1, R'_1, R_2 and R'_2 are relations, such that

$$(R'_1 \sqsubseteq R_1) \wedge (R'_2 \sqsubseteq R_2) \Rightarrow (R'_1 \cup R'_2) \sqsubseteq (R_1 \cup R_2).$$

Based on the preparative definition above, we will address the notion of soundness property.

DEFINITION 4: Given a query processing algorithm A, takes as a database D, a FGAC policy P, and a query Q, and outputs a result $R = A(D, P, Q)$. *Let S denote the standard query answering procedure and* $S(D, Q)$ *the result of answering the query Q when the database state is D and there is no fine-grained access control policy. Then a query processing algorithm A is sound for the query Q, written as* $Sound(A, Q) = true$, *if and only if*

$$\forall_D \forall_P A(D, P, Q) \sqsubseteq S(D, Q).$$

and, if A does not preserve soundness for Q, we denote it as $Sound(A, Q) = false$.

From the definition above, the definition of soundness for algorithm *A* which is defined in [6] is equivalent to the one as follow:

DEFINITION 5: Given a query processing algorithm A and a query Q, the query processing algorithm A is sound, written as $Sound(A)$, *if and only if for* $\forall_Q, Sound(A, Q) = true$.

The definition of soundness and lemmas have been described above. In the following section, we would analyze soundness property using relational algebra expressions which are the basis of query statements. Then, theorems are obtained which are the basis that, for which kinds of SQL queries, the KAB algorithm can guarantee the soundness property.

4.2. Soundness for Relational Algebra Expression

In the following, we will use σ and π to express query which represent the the relational algebra operations *selection* and *projection* respectively [9]. The π is extended to $\pi_{(A,F,\Phi)}$:

- $\pi_{(A,F,\Phi)}(R) =$

  $\{t'[A] |$ if $F(t)$ then $t'[A] = t[A]$ else $t'[A] = \Phi, t \in R\}$.

- $\pi_{(A_1,F_1,\Phi),\cdots,(A_n,F_n,\Phi)}(R) =$

  $\{(t'[A_1],\cdots,t'[A_n])|$ if $F_i(t)$ then $t'[A_i] = t[A_i]$

  else $t'[A_i] = \Phi, i \in [1,\cdots,n], t \in R\}$.

For simplicity, we firstly introduce following notations:

- D : Database; R : Relation; t : tuple; A : Attribute; F, F', $F_1 \cdots F_n, F'_1 \cdots F'_n$: Predicates
- $\pi_{(A,true,\Phi)}(R) = \pi_A(R)$ and $\pi_{(A_1,true,\Phi),(A_2,true,\Phi),\cdots,(A_n,true,\Phi)}(R) = \pi_{A_1,A_2,\cdots,A_n}(R)$

A definition about predicate is introduced as follows, which will simplify our descriptions.

DEFINITION 6: F and F′ are two predicates, we say F′ is ***predicate-subsumed*** *by F, written as $F' \leq F$, if*

$$\forall t, F'(t) = true \Rightarrow F(t) = true.$$

LEMMA 3: $\forall F, F',\ F' \leq F \Rightarrow \sigma_{F'}(R) \sqsubseteq \sigma_F(R)$.

Based on Lemma 3, we have the following two corollaries.

COROLLARY 1:

$$\forall F_1, F_2 \Rightarrow \begin{cases} \sigma_{F_1 \wedge F_2}(R) \sqsubseteq \sigma_{F_1}(R) \\ \sigma_{F_1 \wedge F_2}(R) \sqsubseteq \sigma_{F_2}(R) \end{cases}$$

COROLLARY 2: $\forall F, \sigma_F(R) \sqsubseteq R$.

LEMMA 4: $\pi_{(A,F,\Phi)}(R) \sqsubseteq \pi_A(R)$.

In essence, the mean of Lemma 4 is that when the values of an attribute of any rows in a relation R are replaced with Φ to form another relation R', R' is **subsumed by** R ($R' \sqsubseteq R$). Apparently, when an attribute is extended to many attributes, the lemma is also correct. So, the following corollary is correct too.

COROLLARY 3: $A_1, A_2, \cdots, A_n$ *are different attributes of R, then*

$$\pi_{(A_1,F_1,\Phi),(A_2,F_2,\Phi),\cdots,(A_n,F_n,\Phi)}(R) \sqsubseteq \pi_{A_1,A_2,\cdots,A_n}(R).$$

According to the Lemmas and corollaries above, the following Lemma is correct.

LEMMA 5: $A_1, A_2, \cdots, A_n$ *are different attributes of R, then*

$$F' \leq F \Rightarrow$$

$$\pi_{(F'_1,A_1,\Phi),\cdots,(F'_n,A_n,\Phi)}(\sigma_{F'}(R)) \sqsubseteq \pi_{A_1,\cdots,A_n}(\sigma_F(R)).$$

Based on the Lemma 5, the following Theorem 1 is right which would be used in this paper.

THEOREM 1: $A_1, A_2, \cdots, A_n$ are different attributes of R, then

$$R' \subseteq R \Rightarrow$$

$$\pi_{(F_1,A_1,\Phi),\cdots,(F_n,A_n,\Phi)}(\sigma_F(R')) \sqsubseteq \pi_{A_1,\cdots,A_n}(\sigma_F(R)).$$

According to the analysis above, the Lemma 5 and Theorem 1 are extended to the following lemma and theorem which include multiple tables.

LEMMA 6: $A_1, A_2, \cdots, A_n$ are different attributes of $R_1, R_2, \cdots, R_m$, then

$$F' \leq F \Rightarrow$$

$$\pi_{(F'_1,A_1,\Phi),\cdots,(F'_n,A_n,\Phi)}(\sigma_{F'}(R_1 \times R_2 \times \cdots \times R_m)) \sqsubseteq$$

$$\pi_{A_1,\cdots,A_n}(\sigma_F(R_1 \times R_2 \times \cdots \times R_m)).$$

THEOREM 2: $A_1, A_2, \cdots, A_n$ are different attributes of $R_1, R_2, \cdots, R_m$, then

$$R'_1 \subseteq R_1, \cdots, R'_m \subseteq R_m \Rightarrow$$

$$\pi_{(F_1,A_1,\Phi),\cdots,(F_n,A_n,\Phi)}(\sigma_F(R'_1 \times R'_2 \times \cdots \times R'_m)) \sqsubseteq$$

$$\pi_{A_1,\cdots,A_n}(\sigma_F(R_1 \times R_2 \times \cdots \times R_m)).$$

Relational algebra is the basis of SQL query, so these properties presented in this section are the basic work for the further research.

4.3. Soundness Property of KAB Algorithm

Using relational algebra, there are four forms of queries because there are five basic operations such as "π", "σ", "×", "∪", "−" and the other operations can be derived from them [10]. The first one only involves one table, the second involves multiple tables and the others are combined with the first and second form using set operations such as: "∪" and "−" . They are expressed as follows:

- Form1 :$Q_{form1} = \pi_{(A_1,A_2,\cdots,A_n)}(\sigma_F(R_1))$;
- Form2 :$Q_{form2} = \pi_{(A_1,A_2,\cdots,A_n)}(\sigma_F(R_1 \times \cdots \times R_m))$;
- Form3 :$Q_{form3} = Q_1 \cup Q_2$, Q_1 and Q_2 have the Form1 or Form2;
- Form4 :$Q_{form4} = Q_1 - Q_2$, Q_1 and Q_2 have the Form1 or Form2.

In [10], a translator, which translates a relevant subset of SQL queries into relational algebras, was proposed. Using that, we can map a SQL queries to a relational algebra. Namely, given a SQL query Q, we can find an expression

with relational algebra Q' which is equivalent to Q. Then the following Lemma is presented.

LEMMA 7: A SQL query Q is equivalent to a relational algebra expression Q', if relations $R_1, \cdots, R_n$ involved in Q are replaced with other relations $R'_1, \cdots, R'_n$ to form SQL query Q_1 and $R'_i \subseteq R_i$, $i \in [1 \cdots n]$, then there is a query expressed by relational algebra Q'_1 which is obtained from Q' using $R'_1, \cdots, R'_n$ to replace the relations $R_1, \cdots, R_n$ in Q', and Q_1 is equivalent to Q'_1.

We know that the relational algebra expression is only affected by the schema of relation, the tuples in the relation has nothing to do with the relational algebra. Thus, when $R'_i \subseteq R_i$, $i \in [1 \cdots n]$, the Q is same to Q_1 and Q' is same to Q'_1 because of the same schema of R_i and R'_i, $i \in [1 \cdots n]$. Of course, when Q is equivalent to Q_1, Q' is equivalent to Q'_1 too.

In Section 3, we modify the SQL queries to implement FGAC according to the KAB algorithm. In essence, for any SQL query Q involving relations $R_1, \cdots, R_n$, the modified query Q' is formed from Q using the temporary views $R'_1, \cdots, R'_n$ to replace $R_1, \cdots, R_n$ respectively, where $R'_i \subseteq R_i, i \in [1 \cdots n]$. Thus, according to the Lemma 7, if a SQL query Q is equivalent to relational algebra expression with Form1, Form2, Form3 or Form4, then the modified query according to the KAB algorithm Q' is equivalent to the following forms respectively, where $R'_i \subseteq R_i$, $i \in [1 \cdots m]$:

- Form1' :$Q_{form1'} = \pi_{((F_1,A_1,\Phi),(F_1,A_2,\Phi),\cdots,(F_1,A_n,\phi))}(\sigma_F(R'_1))$;
- Form2' :$Q_{form2'} = \pi_{((F_1,A_1,\Phi),(F_1,A_2,\Phi),\cdots,(F_1,A_n,\phi))}(\sigma_F(R'_1 \times \cdots \times R'_m))$;
- Form3' :$Q_{form3'} = Q'_1 \cup Q''_2$, Q'_1 and Q''_2 have the Form1 or Form2;
- Form4' :$Q_{form4} = Q'_1 - Q''_2$, Q'_1 and Q''_2 have the Form1 or Form2.

According to Theorem 1 and Theorem 2, we can get $Q_{form1'} \sqsubseteq Q_{form1}, Q_{form2'} \sqsubseteq Q_{form2}$. According to Lemma 2, we can get $Q_{form3'} \sqsubseteq Q_{form3}$. Namely, $Q' \sqsubseteq Q$. So the following theorem is right.

THEOREM 3: For any SQL query which is equivalent to a relational algebra expression with Form1, Form2 or Form3, the KAB algorithm can preserve the soundness property.

The Theorem 3 not only include the simple SQL queries and also include some complex SQL queries, because there are many complex SQL queries which can be translated into equivalent relational algebra expression with Form1, Form2 , Form3 [10], such as Q_3 in Example 1. But, unfortunately, when a SQL query is equivalent to a relational algebra expression which has Form4, the soundness property can't be satisfied in all situations by the KAB algorithm.

In Example 1, we point out that for query Q_1 and Q_3, the algorithm in [7] can not hold soundness property. However, using the KAB algorithm to modify the two query, the soundness property can be preserved because they can be translated into the equivalent relational algebra expression with Form1 and Form3 respectively. For Q_1, the result is {Peter} without FGAC policy. When there is FGAC policy P_1, Q_1 is modified into $Q_{1_{modified}}$ to execute over the table "Employee" and the result is empty. So, it don't violate the soundness property. For Q_3 which is a complex SQL, when there is no FGAC policy, the answer is {Peter, Mary}. When the FGAC policy P_1 is working, by using the KAB algorithm Q_3 would be modified to $Q_{3_{modified}}$, the result is empty. The soundness property is preserved.

5. Experiments

We briefly describe here the results of experiments studying the performance of the KAB algorithm. More comprehensive results can be found in [8]. However, only the approach in [6] can preserve soundness property for queries containing MINUS, so we mainly compare the performance of our KAB algorithm with the algorithm in [6]. In the following, we mainly make use of the experimental method in [6] to compare the performance. First, we introduce the experimental parameters:

- **Table Size**: The number of tuples in a table;
- **Selectivity**: The percentage of tuples in a table that are selected by an issued query;
- **Disclosure Probability**: The probability that a cell in a sensitive attribute can be disclosed;
- **Operational Relation Probability**: The percentage of tuples in a table that are in the operational relation for an issued query.

5.1. Experimental Setup

We measure the performance with the table (described in [8]) which is generated based on the Wisconsin Benchmark [11] and is same to the table in [6]. We implemented our query modification approach in Java. The test environment includes a desktop computer with 2.8GHz Intel Pentium (R) D CPU, 1GB of RAM and 120GB disk and Windows XP operating system, Oracle 10g DBMS.

The approach in [6] modifies a query in the same way as the approach in [7], when there is no negation in the query. And it can only guarantee soundness property for those queries containing MINUS. Therefore, in our experiments, we only use queries contain MINUS to measure performance . To measure the cost of executing queries, every query was run 6 times, flushing the buffer pool and query cache between any two executions of query. The results below give the average warm performance numbers for each query.

5.2. Experimental Results and Analysis

For convenience, we refer to KAB algorithm as *KAB*, the approach in [6] as *Sound* , the approach in [7] as *Hippo*

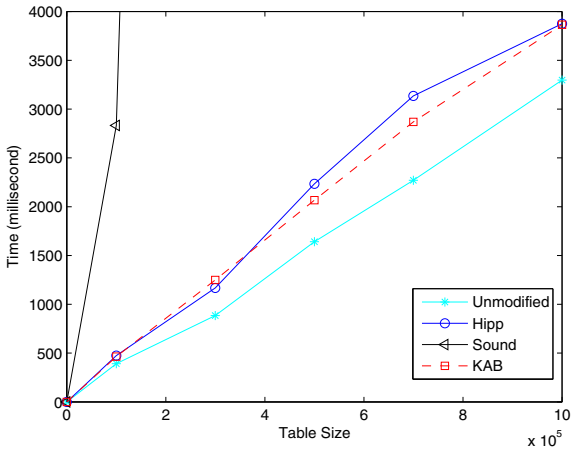

Figure 1. Scalability over Table Size. Parameters: Selectivity =100%, Disclosure probability=75%, Operational Relation probability=100%.

and the normal query evaluation (without access control policies) as $Unmodified$. We only describe the scalability of algorithm here; the **Impact of Selectivity**, **Impact of Disclosure Probability** and **Impact of Operational Relation Probability** are shown in [8].

The scalability of our query modification algorithm is measured by varying table sizes. To measure this cost, we consider the scenario, where selectivity is 100% , disclosure probability is 25% and operational relation probability is 100%. The experimental result is reported in Figure 1. From the figure, we observe that, KAB scales as well as $Hipp$ and $Unmodified$, and better than $Sound$. For $Sound$ approach, when the table size is 100000, the cost is 2833 millisecond and when the table size is varied to 200000, the cost is 34495 millisecond (this is not described in the Figure 1). The reason of non-scalability of $Sound$ is that, it introduces a join operation for providing sound answers to MINUS . However, aim to preserving soundness property, we only add predicates to the where clause of the query. Therefore, KAB is better than $Sound$ when the table size grows, and is similar to $Hipp$ which can not preserve soundness property.

6. Conclusions

In this study, We firstly proposed an algorithm to guarantee soundness property to certain types of SQL queries. Then we extended and enhanced the theory about soundness property, and introduced new properties about soundness property using relational algebra expression. The queries expressed by relational algebra can be divided into four forms which were analyzed respectively for soundness property. Base on these theory, we stated that, for which kinds of SQL queries, the proposed algorithm can guarantee soundness property. Finally, we conducted the performance evaluation of the algorithm. The experimental results show that the proposed algorithm almost doesn't increase the cost and is feasible.

In this paper, our focus is only on SELECT operation. However, the operations controlled by the fine-grained access control are SELECT, UPDATE, INSERT and DELETE. So, in the future work, we will consider the other operations in fine-grained access control.

7. Acknowledgments

The work presented in this paper is supported by 863 hi-tech research and development program of China, granted number: 2006AA01Z430.

References

[1] S. Chaudhuri, T. Dutta, and S.Sudarshan, "Fine grained authorization through predicated grants," in *Proc. of ICDE*, Istanbul, Turkey, April 2007.

[2] R. Agrawa, P. Bird, T. Grandison, J. Kiernan, S. Logan, and W. Rjaibi, "Extending relational database systems to automatically enforce privacy policies," in *Proc. of ICDE*, Tokyo, Japan, April 2005, pp. 1013–1022.

[3] M. Stonebraker and E. Wong, "Access control in a relational data base management system by query modification," in *ACM/CSC-ER Proc. of the 1974 annual conference*, 1974.

[4] *Sybase Adaptive Server Enterprise 12.5, System Administration Guide.Row Level Access Control*, http://sybooks.sybase.com/onlinebooks.

[5] *The Virtual Private Database in Oracle9ir2: An Oracle Technical White Paper*, http://otn.oracle.com/deploy/security/oracle9ir2/pdf/vpd9ir2twp.pdf.

[6] Q. Wang, T. Yu, N. Li, J. Lobo, and E. Bertino, "On the correctness criteria of fine-grained access control in relational databases," in *Proc. of VLDB*, September 2007, pp. 23–28.

[7] K. LeFevre, R. Agrawal, V. Ercegovac, R. Ramakrishnan, Y. Xu, and D. DeWitt, "Limiting disclosure in hippocratic databases," in *Proc. of VLDB*, Toronto, Canada, August 2004.

[8] J. Shi, "On soundness property for sql queries of fine-grained access control in dbmss," http://202.114.1.86/publishcenter/detail.jsp?u=JieShi&p=1&id=1, Tech. Rep., 2008.

[9] R. Ramakrishnan and J. Gehrke, *Database Management Systems, Second Edition*.

[10] S. Ceri and G. Gottlob, "Translating sql into relational algebra: optimization, semantics, and equivalence of sql queries," in *IEEE Trans. on Software Engineering*, vol. SE-11, no. 4, April 1985, pp. 324–345.

[11] D. J.DeWitt, "The wisconsin benchmark: Past, present, and future," Tech. Rep., 1993.