On the Correctness Criteria of Fine-Grained Access Control in Relational Databases

Qihua Wang, Ting Yu, Ninghui Li
Jorge Lobo, Elisa Bertino
Keith Irwin, Ji-Won Byun

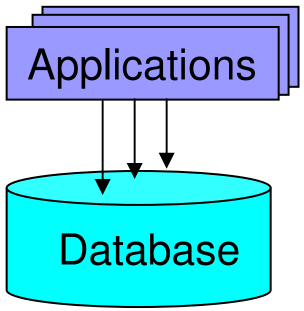
PURDUE UNIVERSITY NC STATE UNIVERSITY IBM

Outline

- Introduction
- Correctness Criteria
- A Fine-Grained Access Control Solution
- Implementation and Experiments
- Conclusions

Introduction

- What is fine-grained access control?
  - Row-level or cell-level access control
  - In contrast to table-level
- Why fine-grained access control?
  - Privacy: access respects individual preferences
- How to implement?
  - Application-level
  - Database-level
    - Hard to bypass
    - Consistency between various applications



Introduction

- Existing DB-Level approaches
  - VPD in Oracle
  - Label-based access control in DB2
  - Limiting disclosure in Hippocratic databases
- Fine-grained access control affects query results
  - No formal notion of correctness
  - Could lead to incorrect or misleading query results

Example

ID	Name	Age	Phone
C001	Linda	32	11111
C002	Mary	29	22222
C003	Nick	NULL 34	33333
C004	Jack	21	44444
C005	Mary	30	NULL 55555

- Q_1 = SELECT Name, Phone FROM T
- Q_2 = SELECT Name, Phone FROM T WHERE Age≥25
- $Q = Q_1 - Q_2$
  - Select information of customers younger than 25

Example

- Q_1 = SELECT Name, Phone FROM T

Name	Phone
Linda	11111
Mary	22222
Nick	33333
Jack	44444
Mary	NULL

Example

ID	Name	Age	Phone
C001	Linda	32	11111
C002	Mary	29	22222
C003	Nick	NULL	33333
C004	Jack	21	44444
C005	Mary	30	NULL

- Q_2 = SELECT Name, Phone FROM T WHERE Age≥25

Name	Phone
Linda	11111
Mary	22222
Mary	NULL

Example

- $Q = Q_1 - Q_2$

Name	Phone
Linda	11111
Mary	22222
Nick	33333
Jack	44444
Mary	NULL

$-$

Name	Phone
Linda	11111
Mary	22222
Mary	NULL

$=$

Name	Phone
Nick	33333
Jack	44444

Example

ID	Name	Age	Phone
C001	Linda	32	11111
C002	Mary	29	22222
C003	Nick	34	33333
C004	Jack	21	44444
C005	Mary	30	55555

- Q_1 = SELECT Name, Phone FROM T
- Q_2 = SELECT Name, Phone FROM T WHERE Age≥25
- $Q = Q_1 - Q_2$
  - Select information of customers younger than 25

Example

- Without fine-grained access control

Name	Phone
Jack	44444

- With fine-grained access control

Name	Phone
Nick	33333
Jack	44444

Outline

- Introduction
- Correctness Criteria
- A Solution
- Implementation and Experiments
- Conclusions

Intuitive Explanation

- Sound
  - Be consistent with when there is no access control
- Secure
  - Do not leak information not allowed by policy
- Maximum
  - Return as much correct information as allowed by policy

Formal Definitions

- *D*: Database
- *P*: Disclosure policy
  - Determine what information may be disclosed
  - Defines an equivalence relation among database states
    - $D \equiv_P D'$

Name	Age	Phone
Alice	25	111
Bob	30	888

$\equiv_P$

Name	Age	Phone
Alice	33	111
Bob	30	666

Formal Definitions

- *R*: Relation
  - A cell may take the value *unauthorized*
- A tuple is subsumed by another: $t_1 \sqsubseteq t_2$
  - $<x_1 \ldots x_n> \sqsubseteq <y_1 \ldots y_n>$ if and only if $x_i = y_i$ or $x_i = unauthorized$
  - *E.g.* $<Alice, unauthorized> \sqsubseteq <Alice, 28>$
- A relation is subsumed by another: $R1 \sqsubseteq R2$
  - Exists a mapping f: $R_1 \rightarrow R_2$
  - For every tuple t in R_1, $t \sqsubseteq f(t)$

Formal Definitions

- *R*: Relation
- *Q*: Query
- *A*: Query processing algorithm that takes disclosure policy into account
- *A(D,P,Q)*: Answer to *Q* on *D* with policy *P*
- *S*: Standard query processing algorithm
- *S(D,Q)*: Answer to *Q* on *D* without access control

Sound

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

- May return less information due to access control
- Should not return wrong information that is not in standard answer

Name	Phone
Nick	NULL
Jack	44444

$\not\sqsubseteq$

Name	Phone
Jack	44444

Secure

$$\forall_P \forall_Q \forall_D \forall_{D'} \left[(D \equiv_P D') \rightarrow (A(D, P, Q) = A(D', P, Q)) \right]$$

- Answer does not depend on information that is not disclosed by policy
- Implies stronger security guarantee
  - Multi-user collusion resistance
  - Multi-query resistance

Maximum

Given any (*D, P, Q*), for any relation *R* such that

$$\forall_{D'} \left[(D \equiv_P D') \rightarrow (R \sqsubseteq S(D', Q)) \right]$$

We have

$$R \sqsubseteq A(D, P, Q)$$

- No other sound and secure answer that contains more information than the answer returned by *A*

Correctness Criteria

- Any query processing algorithm that provides fine-grained access control should be sound and secure, and strive to be maximum.
- Many existing approaches are
  - Secure
  - Not sound
  - Not maximum
    - Too little information is returned in certain cases

Outline

- Introduction
- Correctness Criteria
- A Solution
- Implementation and Experiments
- Conclusions

Solution

- A sound query evaluation algorithm
  - Low evaluation $Q_{\_}$: tuples definitely correct
  - High evaluation Q^{-} : tuples possibly correct
  - $Q_1 - Q_2$ is evaluated as $Q_{1\_} - Q_2^{-}$
- A variable-based labeling mechanism
  - Use variables instead of NULL to hide information
  - Secure
  - Preserves more information

Variable-Based Labeling Mechanism

- Existing approaches: replace every piece of unauthorized information with NULL
  - Too much information is lost
  - Unknown: NULL = 100?, NULL = NULL?

Name	Age
Alice	NULL 25

Q = SELECT Name FROM T WHERE Age = Age

Result is an EMPTY relation!

Variable-Based Labeling Mechanism

- Information useful in query evaluation without leaking concrete value
  - A cell equals to itself
  - Cells in primary key take different values
  - Certain linkages through foreign key
    - Information of the same person stored in two tables so as to comply with normal forms
- Our approach: replace unauthorized information with variables

Two Types of Variables

- Type-1 variable: v
  - Variable is equivalent to itself
    - True: $v_1 = v_1$, $v_2 = v_2$ (in contrast to NULL ≠ NULL)
  - Unknown when compared with other variables or constants
    - Unknown: $v_1 = v_2$?, $v_1 = 100$?
- Type-2 variable: $<name, domain>$
  - In the same domain, compare names
    - True: $<a, 1> = <a, 1>$, $<a, 1> \neq <b, 1>$
  - Otherwise, unknown
    - Unknown: $<a, 1> = <a, 2>$?, $<a, 1> \neq <b, 2>$?
    - Unknown: $<a, 1> = v_1$?, $<a, 1> = 100$?

Example

Based tables

SSN	Name	Age
1111	Alice	19
2222	Bob	35
3333	Carol	19

SSN	Occupation
1111	Student
1111	Waiter
2222	Professor
3333	Secretary
3333	Dancer

Traditional approach

SSN	Name	Age
NULL	Alice	NULL
NULL	Bob	35
NULL	Carol	NULL

SSN	Occupation
NULL	Student
NULL	Waiter
NULL	Professor
NULL	Secretary
NULL	Dancer

Our labeling approach

SSN	Name	Age
$\langle a,1 \rangle$	Alice	v_1
$\langle b,1 \rangle$	Bob	35
$\langle c,1 \rangle$	Carol	v_2

SSN	Occupation
$\langle a,1 \rangle$	Student
$\langle a,1 \rangle$	Waiter
$\langle b,1 \rangle$	Professor
$\langle c,1 \rangle$	Secretary
$\langle c,1 \rangle$	Dancer

Variable-Based Labeling Mechanism

- Provides security
  - Variables hide concrete values
- Makes it possible to return more information
  - Strive for maximum
- Does not deal with sound

A Sound Query Evaluation Algorithm

- Low evaluation: Q_-
  - Contains tuples that are definitely correct
- High evaluation: Q^-
  - Contains tuples that are possibly correct
- Tuples $<x_1,\ldots x_n>$ and $<y_1,\ldots y_n>$ are compatible if it is possible make to them identical by setting the values of variables
  - Different type-2 variables in the same domain must have different values

A Sound Query Evaluation Algorithm

- $Q = R$: $Q_{\_} = Q^{-} = L(R)$
- $Q = \sigma_c Q_1$: $Q_{\_} = \sigma_c Q_{1\_}$ and $Q^{-} = \sigma_{c \vee IsUn(c)} Q_1^{-}$
- $Q = \pi_{a1\ldots} Q_1$: $Q_{\_} = \pi_{a1\ldots} Q_{1\_}$ and $Q^{-} = \pi_{a1\ldots} Q_1^{-}$
- $Q = Q_1 \times Q_2$: $Q_{\_} = Q_{1\_} \times Q_{2\_}$ and $Q^{-} = Q_1^{-} \times Q_2^{-}$
- $Q = Q_1 \cup Q_2$: $Q_{\_} = Q_{1\_} \cup Q_{2\_}$ and $Q^{-} = Q_1^{-} \cup Q_2^{-}$
- $Q = Q_1 - Q_2$
  - $Q_{\_}$ contains all tuples t in $Q_{1\_}$ such that no tuple in Q_2^{-} is compatible with t
    - Intuitively, $Q_{\_} = Q_{1\_} - Q_2^{-}$
  - Q^{-} contains all tuples that are in Q_1^{-} but not in $Q_{2\_}$
    - Intuitively, $Q^{-} = Q_1^{-} - Q_{2\_}$

A Sound and Secure Solution

- Given any query *Q*
  1. Perform variable-based labeling
  2. Compute and return *Q*_
- Sound and secure
- Returns at least as much information as existing algorithms for fine-grained access control

Example

ID	Name	Age	Phone
C001	Linda	32	11111
C002	Mary	29	22222
C003	Nick	34 v_1	33333
C004	Jack	21	44444
C005	Mary	30	55555 v_3

- Q_1 = SELECT Name, Phone FROM T
- Q_2 = SELECT Name, Phone FROM T WHERE Age≥25
- Q_3 = SELECT Name, Phone FROM T WHERE Age < 30
- $Q = Q_1 - (Q_2 - Q_3)$
  - Select information of customers younger than 30

Example

- Given $Q = Q_1 - (Q_2 - Q_3)$, compute $Q\_$
  - Compute $Q_{1\_}$
  - Compute $(Q_2 - Q_3)^{-}$
    - Compute Q_2^{-} and $Q_{3\_}$

Example

- Q_1 = SELECT Name, Phone FROM T
- Q_{1-} :

Name	Phone
Linda	11111
Mary	22222
Nick	33333
Jack	44444
Mary	v_3

Example

ID	Name	Age	Phone
C001	Linda	32	11111
C002	Mary	29	22222
C003	Nick	v_1	33333
C004	Jack	21	44444
C005	Mary	30	v_3

- Q_2 = SELECT Name, Phone FROM T WHERE Age≥25
- Q_2^-:

Name	Phone
Linda	11111
Mary	22222
Nick	33333
Mary	v_3

Example

ID	Name	Age	Phone
C001	Linda	32	11111
C002	Mary	29	22222
C003	Nick	v_1	33333
C004	Jack	21	44444
C005	Mary	30	v_3

- Q_3 = SELECT Name, Phone FROM T WHERE Age < 30
- Q_{3-} :

Name	Phone
Mary	22222
Jack	44444

Example

- $(Q_2 - Q_3)^-$

Name	Phone
Linda	11111
Mary	22222
Nick	33333
Mary	v_3

Q_2^-

–

Name	Phone
Mary	22222
Jack	44444

Q_{3-}

=

Name	Phone
Linda	11111
Nick	33333
Mary	v_3

Example

- $Q_{-} = (Q_1 - (Q_2 - Q_3))_{-}$

Name	Phone
Linda	11111
Mary	22222
Nick	33333
Jack	44444
Mary	v_3

Q_{1-}

$-$

Name	Phone
Linda	11111
Nick	33333
Mary	v_3

$(Q_2 - Q_3)^{-}$

$=$

Name	Phone
Jack	44444

Final result

Example

- Without fine-grained access control

Name	Phone
Mary	22222
Jack	44444

- Hippocratic database approach

Name	Phone
Mary	22222
Nick	33333
Jack	44444

Outline

- Introduction
- Correctness Criteria
- A Solution
- Implementation and Experiments
- Conclusions

Implementation Approaches

- Query modification
  - Pros: can be applied in existing DBMS
  - Cons: performance penalty
- Modify DBMS query evaluation engines
  - Pros: better performance
  - Cons: require source codes

Query Modification

- Q = SELECT Name, Age FROM T WHERE Age≥25
- Revision:

SELECT Name, Age FROM
(SELECT CASE WHEN C_{name}
THEN Name ELSE NULL END AS Name,
CASE WHEN C_{age}
THEN Age ELSE NULL END AS Age
FROM T)
WHERE Age≥25

Query Modification

- Q_1 = SELECT $a_1,\ldots a_n$ FROM T_1
- Q_2 = SELECT $a_1,\ldots a_n$ FROM T_2
- $Q = Q_1 - Q_2$
- Revision:

SELECT $a_1,\ldots a_n$ FROM T_1
MINUS
SELECT $a_1,\ldots a_n$ FROM T_1, T_2 WHERE
(($T_1.a_1 = T_2.a_1$) OR ($T_1.a_1$ IS NULL) OR ($T_2.a_1$ IS NULL))
AND … AND
(($T_1.a_n = T_2.a_n$) OR ($T_1.a_n$ IS NULL) OR ($T_2.a_n$ IS NULL))

Query Modification

- Use CASE statements to replace each piece of unauthorized information with NULL
  - Notice: existing DBMS do not support variables
- Use JOIN operation to handle MINUS
  - Tuple compatibility not directly supported by DBMS

Query Modification

- Q_1 = SELECT $a_1,\ldots a_n$ FROM T_1
- Q_2 = SELECT $a_1,\ldots a_n$ FROM T_2
- $Q = Q_1 - Q_2$
- Revision of Q:

  SELECT $a_1,\ldots a_n$ FROM T_1

  MINUS

  SELECT $a_1,\ldots a_n$ FROM T_1, T_2 WHERE

  (($T_1.a_1$ = $T_2.a_1$) OR ($T_1.a_1$ IS NULL) OR ($T_2.a_1$ IS NULL))

  AND … AND

  (($T_1.a_n$ = $T_2.a_n$) OR ($T_1.a_n$ IS NULL) OR ($T_2.a_n$ IS NULL))

Experiments

- Objectives
  - Performance when evaluate queries with minus
  - Factors that affect performance

Parameters

- Table size
  - Number of tuples
- Selectivity
  - Percentage of selected tuples in a table
- Sensitivity
  - Number of selected attributes that are governed by policy
- Uniformity
  - Expected number of tuples having the same value in an attribute
- Disclosure probability
  - Probability that a cell is disclosed by policy

Comparison

- Standard evaluation algorithm
  - Without access control
- Limiting disclosure approach in Hippocratic Databases
  - Could return results that are unsound

Experimental Results

- Not as scalable as the other two approaches
  - Costly to perform JOIN operation
  - Reasonable performance when table size is moderate
  - Answer in 2 seconds when table size is 10000
- Perform significantly better when uniformity is small
  - Because join operation can be computed faster
- Perform better when disclosure probability is large
  - Because conditions are evaluated faster
- Perform significantly better when sensitivity is small
  - Because selection conditions are simpler

Experimental Results

- Not as scalable as the other two approaches
  - Costly to perform JOIN operation
  - Reasonable performance when table size is moderate
  - Answer in 2 seconds when table size is 10000
- Performance affected by distribution of data and disclosure percentage
  - Details in paper

Conclusion

- We have
  - Pointed out existing fine-grained access control algorithms may return misleading results
  - Formally proposed the notions of sound, secure and maximum as correctness criteria
  - Proposed a variable-based labeling mechanism
  - Designed a sound and secure algorithm
  - Presented a query-modification approach
  - Performed experiments

Relation with Works on Incomplete Information Databases

- Some ideas and techniques in incomplete information databases can be applied to fine-grained access control
- New contributions
  - Formalize the notion of security
  - Propose novel labeling scheme that uses two types of variables
  - Design a query modification approach to evaluate queries in a sound and secure manner
  - Study factors that affect performance

Thank you!

End