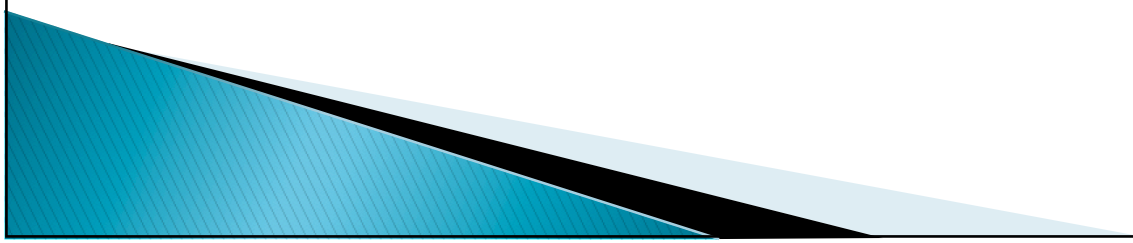
Extending Query Rewriting Techniques for Fine-Grained Access Control

Shariq Rizvi, Alberto Mendelzon,
S. Sudarshan, Prasan Roy

Presenter: Thao N. Pham

Introduction

- Fine-grained access control for databases
  - Views and role-based access control
  - Oracle's Virtual Private Database
  - Hippocratic databases and its extension
- Transparent access control: User queries is **modified before execution** (i.e, adding predicates) so that unauthorized data are filtered out without the user's awareness.

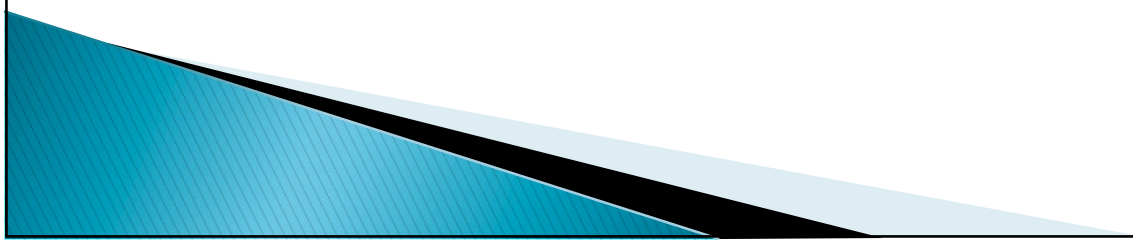


Introduction

- What are some issues?
  - Using “fixed” views: scalability when the system has many users?
  - → Parameterized authorization views
  - How to control not only what data can be accessed, but also how the data is accessed.
    - Example: bank tellers can access each customer’s account balance by giving the account number, but cannot see a list of all customers’ balances.
  - → Pattern views

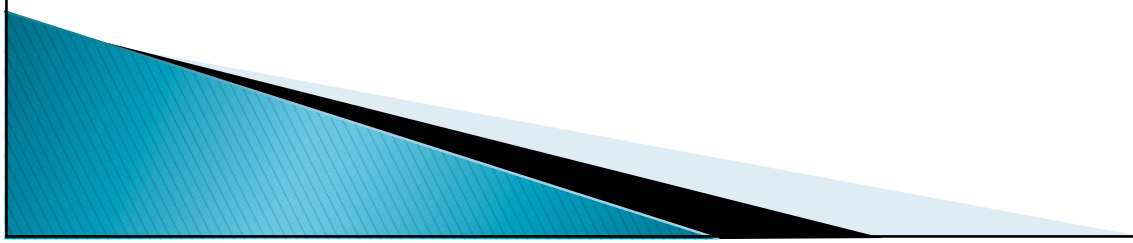
Introduction

- What are some issues (cont)?
  - When a query is allowed to access only part of the data it is asking
    - Transparently modify the query to filtered out unauthorized data (Truman Model)
    - vs Reject the query (Non-Truman).
      - Unconditionally valid, conditionally valid, inference rules for validity check, carrying out the validity check



Outline of the remaining talk

- Parameterized and pattern views
- The Truman vs Non-Truman model
- Unconditionally and conditionally validity of a query
- Inference rules
- Validity checking using query optimizing techniques
- Conclusion



Parameterized authorizaton views

- A normal relational view is defined by a query on data relation or on other views
- Parameterized authorization views can have parameters in its defining query
  - Example:

Students *(student-id, name, type)*
Courses *(courses-id, name)*
Registered *(student-id, course-id)*
Grades *(student-id, courses-id, grade)*

A student can see all grades of students in the courses he has registered

```
Create authorization view Co-StudentGrades as
Select Grades.*
From Grades, Registered
Where registered.student-id = $user-id
And  Grades.course-id = Registered.course-id
```

Access pattern views

- Says how the data should be accessed
- Example:

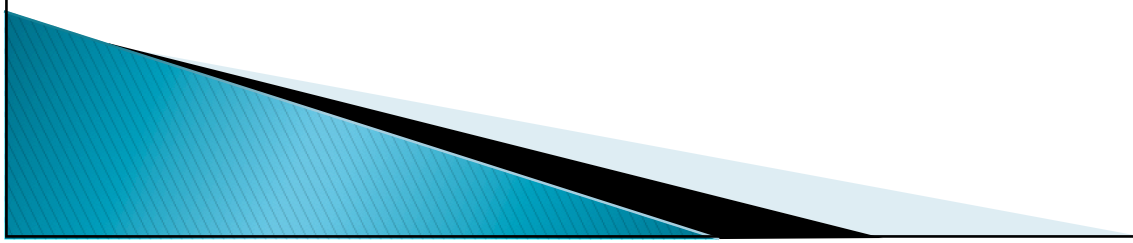
Students *(student-id, name, type)*
Courses *(courses-id, name)*
Registered *(student-id, course-id)*
Grades *(student-id, courses-id, grade)*

Specifying a student -id, the user can see the student's grades, but he cannot the grades of all students

Create authorization view SingleGrade **as**
Select * **from** Grades
where student-id = $$1

Truman model

- Conceptually, provide each user a restricted view of the complete database
  - The user's query on the original database is translated to be on the corresponding personal view.
- Technically, the user's query is modified to filtered out the portion of data that does not belong to the user's authorization views
  - Basically, each relation in the user's query is replaced by the corresponding authorization view for the user.



Truman model (cont)

- Problems with the Truman model: The user is oblivious to the modification on his query
  - Inconsistence between what the user expects to see and what the system returns to him.
    - Example

```
Create authorization view  MyGrade as
Select * from Grades
where student-id = $user-id
```

User query:

```
Select avg (grade) from Grades
```

system-modified query:

```
Select avg (grade) from MyGrades
```

→The student think that his average grade is the same as the overall average!!!

Non-Truman model

- Binary decision
  - If a query is valid, returns the result
  - Otherwise, reject the query!
- So, when a query is defined to be valid?

Unconditional validity

- A user's query q is unconditionally valid if, on all database states, there is a query q' on the user's authorization views that returns exactly the same result set as that of q
- Example:

Students *(student-id, name, type)*
Courses *(courses-id, name)*
Registered *(student-id, course-id)*
Grades *(student-id, courses-id, grade)*

Create authorization view MyGrade **as**
Select * **from** Grades
where student-id = $user-id

User query:

Select avg (grade) **from** Grades
Where student-id = '11'

equivalent query on MyGrades:

Select avg (grade) **from** MyGrades

Conditionally validity

- A user's query q is conditionally valid if there is a query q' on the user's authorization views that returns exactly the same result set as that of q on the current database state
- Example:

Students *(<u>student-id,</u> name, type)*
Courses *(<u>courses-id,</u> name)*
Registered *(<u>student-id,</u> course-id)*
Grades *(<u>student-id,</u> courses-id, grade)*

Create authorization view Co-StudentGrades **as**
Select Grades.*
From Grades, Registered
Where registered.student-id = $user-id
And Grades.course-id = Registered.course-id

User query:

Select * **from** Grades

Where course-id = 'CS101'

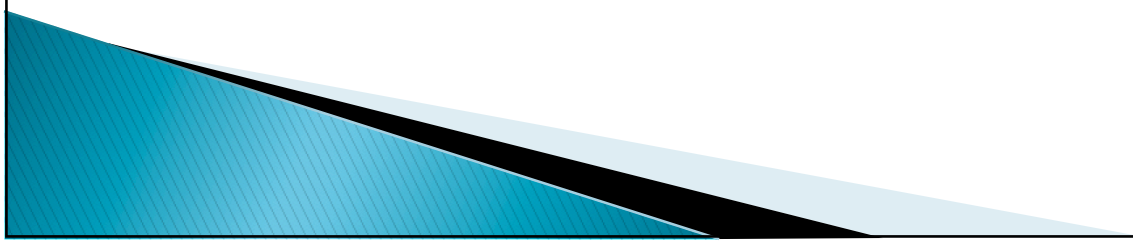
→Conditionally valid if the student is currently enrolled in the course CS101 and the student is allowed to know that. Why?

Discussion…

- What are some weaknesses of the non-Truman model?
  - The user is aware of the access control policy applied to him → policy leakage
  - It's not easy to carry out validity check…

Inference rules for validity check

- Rules for unconditional validity
  - (U1)A query defining an authorization view is unconditionally valid
  - (U2) If queries q_1, …q_n is unconditionally valid then E(q_1,…q_n) is unconditionally valid.



Inference rules for validity check (cont)

- Rules for unconditional validity (cont)
  - (U3): unconditional validity using integrity constraint:
    - Example:

Students *(student-id, name, type)*
Courses *(courses-id, name)*
Registered *(student-id, course-id)*
Grades *(student-id, courses-id, grade)*

User query:

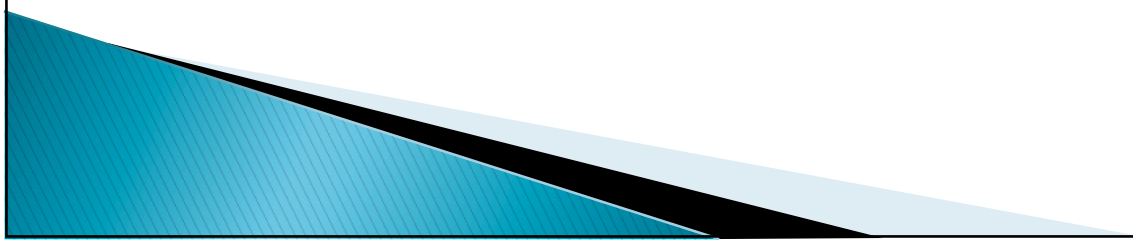
Select distinct name, type
From Students

Create authorization view RegStudents **as**
Select Registered.course-id, Students.name, Student.type
From Registered, Students
Where Students.student-id = Registered.Student-id

→unconditionally valid if for each tuple in Students (*core*) there is at least a tuple of the same student-id in Registered (*remainder*)

Inference rules for validity check(cont)

- Rules for conditional validity
  - (C1) If a query is unconditionally valid, it is conditionally valid
  - (C2) If queries q_1, …q_n is conditionally valid then $E(q_1,\ldots q_n)$ is conditionally valid



Inference rules for validity check (cont)

- Rules for conditional validity (cont)
  - (U3): example:

Create authorization view Co-StudentGrades **as**
Select Grades.*
From Grades, Registered
Where registered.student-id = $user-id
And Grades.course-id = Registered.course-id

User query:

Select * from Grades
Where course-id = 'CS101'

core: Grades, remainder: Registered

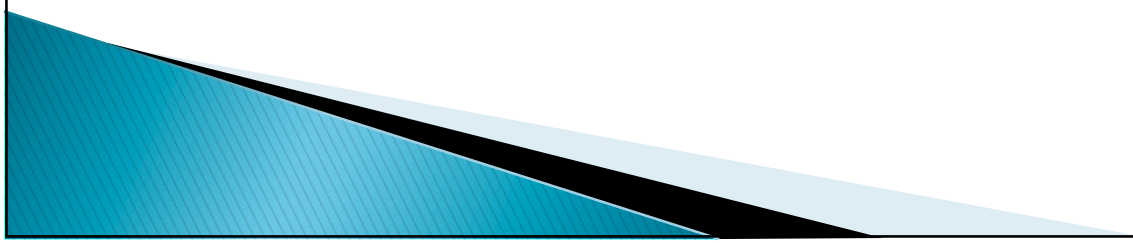
conditionally valid in a database state D if the following query is conditionally valid in D:

Select distinct 1 **from** Registered *(remainder)*
where student-id = 11(P_{ic}) **and** course-id = 'CS101' (P_r)

Inference rules for validity check (cont)

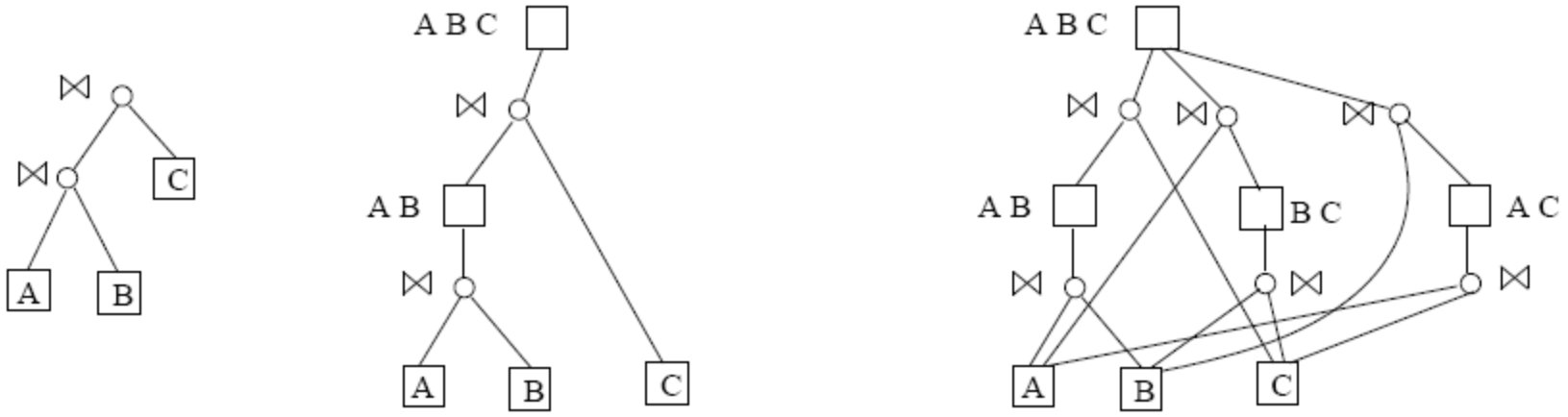
- This set of inference rule is **not complete**
  - If a query is rejected while it should be able to access the acquired data, the user needs to rephrase the query to make it valid!

- Discussion: If a query should be conditionally valid but the system did not recognize that, how can the user know whether this is because the query is not valid in the current state or because he did not write it properly?



Validity checking using query optimizing techniques

- Volcano algorithm with sub-tree unification and materialized view usages

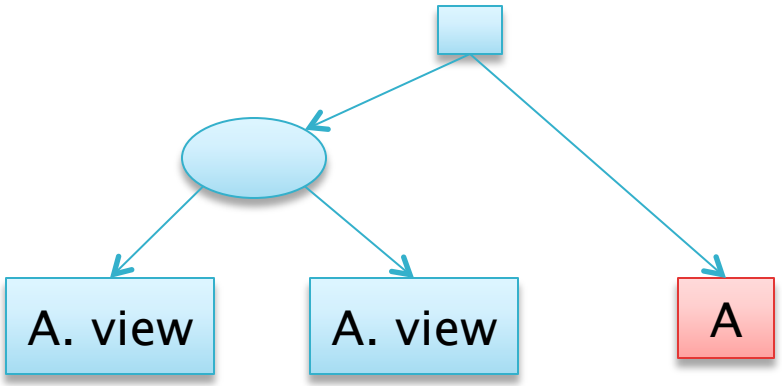


(a) Initial Query (b) DAG representation of query (c) Expanded DAG after transformations (Commutativity not shown)

Figure 1: Volcano Data Structures

Validity checking using query optimizing techniques

- Basic rule checking (applying rules U1, U2, C1, C2)
  - Replace authorization view in the place of materialized view in the technique
  - An “or” (equivalence) node is valid if any of its children is valid
  - An “and” (operation) node is valid if all of its children is valid



Discussion…

- Detailed and complete checking methods for all proposed inference rules are left as future work
- Even if such methods are available…
  - Optimization and authorization check is two difference purposes, so an optimization algorithm does not aim to find the plan that satisfies authorization policies!

Conclusion

- Strengths:
  - Non-Truman: new model for fine-grained access control using parameterized and pattern authorization view, together with a set of meaningful inference rules.
  - The concept of conditional validity allows more flexibility in access control enforcement
  - The paper somehow provides access control transparency by still allowing users to write his query on original relations instead of on authorization views

Conclusion (cont)

- Weaknesses:
  - Policy leakage of the non-Truman model
  - The conditional validity might add more to the unrepeatable read problem in transaction concurrency control.
  - The set of inference rule is not complete
  - A lot more work needed for a complete validity checking.