# Access Control by Query Modification

TruSec Seminar

November 3, 2004

Ji-Won Byun

## Outline

- Access Control in a RDBMS by Query Modification

  - Michael Stonebraker and Eugene Wong, ACM CSC-ER, 1974
- Oracle VPD

  - Oracle Technical White Paper, 2002
- Extending Query Rewriting Techniques

  - Rizvi et al., SIGMOD, 2004

## Query Modification

- **Basic idea**
  - a user interaction with the database is modified to an alternate form which is guaranteed to have no access violations.
  - The modification takes place in a high level interaction language so that the processing of a resulting interaction can be accomplished without further regard for protection.
  - No controls on access paths to data need to be present.

## Query Modification - Examples

- **QUEL (Query Language)**

  – RANGE relation-name (symbol, ..., symbol):...
    RETRIEVE workspace-name:target-list:qualification

  - In SQL, RANGE = From
    target-list = Select
    qualification = Where

## Query Modification - Examples

- Suppose the following tables exist.

  employee (name, dept, salary, manager)

  department (dept, floor#, #emp, sales)

ex) Find the employees who work on the first floor

```
RANGE employee(X):department(Y)
RETRIEVE W:X.name:(X.dept=Y.dept) and (Y.floor#=1)
```

## Query Modification-Examples

- Suppose that Smith can see only information on himself.

  RANGE employee(X)

  RETRIEVE W1:X.name, X.dept, X.salary:X.name=Smith

- Smith wants to find out the salary of Jones

  RANGE employee(X)

  RETRIEVE W:X.salary:X.name=Jones

➔ RANGE employee(X)

  RETRIEVE W:X.salary:X.name=Jones ***and X.name=Smith***

## Query Modification-Algorithm

- For each user U, a set of access control interactions, $I = \{I_1, \ldots, I_k\}$ is stored.
- Each $I_j$ is logically of the form

  RANGE relation-name (symbol, ..., symbol):...

  RETRIEVE target-list:qualification
- Special keywords: “ALL” and “NO ACCESS”

## Query Modification-Algorithm

- For each tuple variable X, appearing in a given user interaction R, the following algorithm is executed.

1. If the relation which X references is a workspace, then exit.
2. Find all attributes in the target list or the unmodified qualification statement of R referenced with X. Call this set S.
3. Find all access control interactions with the same command as R and with a target list containing all attributes in S. Denote these by $I_R = \{I_1, \ldots, I_j\}$ and their qualifications by $Q_1, \ldots, Q_j$.
4. Delete from $I_R$ all interactions with a target list containing the target list of another interaction
5. Replace Q, the qualification in R, by Q and ($Q_1$ and … $Q_j$).

## Query Modification-Example

- Jones can see all salaries as long as his query does not include name or department in the target list or qualification. Moreover, except for employee Baker, he can see names, managers if salary is not present. Furthermore, he can see names, managers and salaries for all employees who earn more than their managers. Lastly, he can se all attributes for departments which sell more than the average department.

➔ RANGE employee(X, Y):department(Z)

```
1. RETRIEVE X.salary, X.manager
2. RETRIEVE X.name, X.dept, X.manager: X.name!=Baker
3. RETRIEVE X.name, X.salary, X.manager:
                Y.name=X.manager and X.salary>Y.salary
4. RETRIEVE ALL:Z.sales>AVE(Z.sales)
```

## Query Modification-Example

- **Jones issues a query to find all salaries**

  RANGE employee(X)

  RETRIEVE W:X.salary

➔ S = {salary}

**1. RETRIEVE X.salary, X.manager**

2. RETRIEVE X.name, X.dept, X.manager: X.name!=Baker

**3. RETRIEVE X.name, X.salary, X.manager:**
**Y.name=X.manager and X.salary>Y.salary**

4. RETRIEVE ALL:Z.sales>AVE(Z.sales)

➔ Since the target list of 3) contains the target list of 1), by the step 4, only 1) is applied.

➔ No change

## Query Modification-Example

- **Jones issues a query to find the manager of Adam**

  RANGE employee(X)

  RETRIEVE W:X.manager:X.name=Adam

➔ S = {name, manager}

1. RETRIEVE X.salary, X.manager
2. **RETRIEVE X.name, X.dept, X.manager: X.name!=Baker**
3. **RETRIEVE X.name, X.salary, X.manager:**
   **Y.name=X.manager and X.salary>Y.salary**
4. RETRIEVE ALL:Z.sales>AVE(Z.sales)

➔ RANGE employee(X, Y)

RETRIEVE W:X.manager:X.name=Adam and X.name!=Baker
and Y.name=X.manager and X.salary>Y.salary

## Query Modification-Aggregates

- **What about aggregate functions?**
  - Suppose Adam can retrieve only salaries for employees in the toy department.

    ➔ RANGE employee(X)

    RETREIVE X.salary:X.department=Toy

  - Adam issues a query to find the average salary of the company.

    RANGE employee(X)

    RETREIVE W:AVE(X.salary)

## Query Modification-Aggregates

- **Four possibilities**
  1. Allow aggregates without restriction
  2. Allow aggregates without restriction if the minimum number of values aggregated exceeds some threshold
  3. Allow aggregates without restriction if they are unqualified (e.g. are aggregates over a whole relation)
  4. Allow aggregates only with access control qualifications appended inside the function

## Query Modification-Aggregates

1. Allow aggregates without restriction

   To find Smith’s salary:

   ```
   RANGE employee(X)
   RETRIEVE W:AVE(X.salary):X.name=Smith
   ```

## Query Modification - Aggregates

2. Allow aggregates without restriction if the minimum number of values aggregated exceeds some threshold

To find Smith's salary:

```
RANGE employee(X)
RETRIEVE W:COUNT(X.name):X.name>=Smith (R1)
RETRIEVE W:AVE(X.salary):X.name>=Smith (R2)
RETRIEVE W:AVE(X.salary):X.name>Smith  (R3)
```

➔ Smith's salary = $R_1 \times R_2 - [(R_1 - 1) \times R_3]$

## Query Modification-Aggregates

3. Allow aggregates without restriction if they are unqualified (e.g. are aggregates over a whole relation)
4. Allow aggregates only with access control qualifications appended inside the function

- 3) and 4) do not allow any potential violation.
- But some anomaly arises.

## Query Modification-Aggregates

- Suppose Adam is allowed to see only tuples of those employees in the toy department.
- Adam issues the following queries:

```
RANGE employee(X)
RETRIEVE W:AVE(X.salary)   (R1)
RETRIEVE W:AVE(X.salary):X.name>AAAAA (R2)
```

- R1 contains the average salary of the company, but R2 contains the average salary of the toy department.
- The price paid for the increased flexibility.

## Oracle VPD

- The Virtual Private Database (VPD) is the aggregation of server-enforced, fine-grained access control, together with a secure application context in the Oracle database.

- By dynamically appending SQL statements with a predicate, VPD limits access to data at row level and ties the security policy to the table (or view or synonym) itself.

## Oracle VPD

- How does it work?

  When users access a table (or view) that has a security policy,

  1. The Oracle server calls the policy function, which returns a predicate.
  2. Oracle then dynamically rewrites the query by appending the predicate to the user’s SQL statement.
  3. The modified SQL query is executed.

## Oracle VPD

- Suppose we have the following table.

  ```
  my_table(data varchar2(30),
                     owner varchar2(30) default USER);
  ```

- We want to allow users to access only the data they own. But Admin can access any data without restrictions.

## Oracle VPD-Example

**1. Create a policy function**

```
Create function sec_function(p_schema varchar2, p_obj varchar2)
Return varchar2
As
Begin
        if (USER = ‘ADMIN’) then
              return ‘ ’; // same as returning ‘true’
        else
              return ‘owner = USER’;
        end if;
End;
```

## Oracle VPD-Example

**2. Attach the policy function to my_table**

```
Begin
        dbms_rls.add_policy
        (  object_schema => ‘RLS’,
            object_name => ‘MY_TABLE’,
            policy_name => ‘MY_POLICY’,
            function_schema => ‘RLS’,
            policy_function => ‘SEC_FUNCTION’,
            statement_types => ‘select, insert, update, delete’,
            update_check => TRUE );
End;
```

## Oracle VPD-Example

**3. Use my_table**

```
connect rls/password

select * from my_table;
```

= only shows the rows that owner is ‘rls’

```
insert into my_table(data) values(‘Some data’); OK!
insert into my_table values(‘Other data’, ‘Scott’); NOT OK!
```

= because of the check option.

## Oracle VPD-Application Context

- Application contexts act as secure caches of data that may be applied to a fine-grained access control policy.
  - Upon logging into the database, Oracle sets up an application context in the user’s section.
  - You can define, set and access application attributes that you can use as a secure data cache.

## Oracle VPD-Application Context

- **Steps**
  1. **Create a PL/SQL package that sets the context**

```
Create package App_sec_context IS
        procedure Set_cust_num IS
              cusnum number;
        begin
              select cus_no into cusnum from customer
              where cust_name = SYS_CONTEXT(‘USERENV’, ‘SESSION_USER’);
              DBMS_SESSION.SET_CONTEXT(‘order_entry’, ‘cust_num’, cusnum);
        end;
End;
```

## Oracle VPD-Application Context

- **Steps**

  **2. Create a context and associate it with the package**

  Create Context order_entry Using App_sec_context;

  : order_entry is the context namespace, and App_sec_context is the package that sets the attributes in the context namespace.

  **3. Set the context before users retrieve data**

  Use an event trigger on login to pull session information into the context.

  **4. Use the context in a VPD function**

  return ‘cust_num = SYS_CONTEXT(‘’order_entry’’, ‘’cust_num’’)’;

## Extending Query Rewriting Techniques

- Goal
  - To provide a security model in which fine-grained authorization policies are defined and enforced in the database level.
  - Queries should be authorization-transparent. That is, queries should be written against the database relations without having to refer to the authorization views.
  - Why? Since different users may have different authorization views, this would require application programmers to code interfaces differently for each user.

## Extending Query Rewriting Techniques

- Mechanism: Authorization Views
  - Access control is specified using authorization views.
  - Traditional views or Parameterized views
  - A parameterized view is like a normal view, but with parameters like user-id, time and user-location appearing in its definition.
  - Can also be aggregate views.
  - Ex. Create Authorization View MyGrades As

    Select * From Grades Where student-id = $user-id;

## Authorization Models

- Truman Model (Oracle VPD)
  - Provide each user with a personal and restricted view of the complete database.
  - User queries are modified transparently to make sure the user does not get to see anything more than her view of database.
  - The returned result is correct with respect to the restricted view.

## Authorization Models

- A major drawback of Truman Model
  - Inconsistencies between what the user expects to see and what the system returns.
- Example

```
Create Authorization View MyGrades As
Select * From Grades Where student-id = $user-id;

User Query:      Select avg(grade) From Grades;
Modified Query: Select avg(grade) From MyGrades;
```

➔ The result will be the average of her own grades.

## Authorization Models

- Another example

  A user runs a query to find all students who have higher grades than the user.

  ```
  Select student-id
  From Grades, (Select avg(grade) as gr From Grades) A
  Where grade > A.gr;
  ```

  ➔ The result will return nothing.

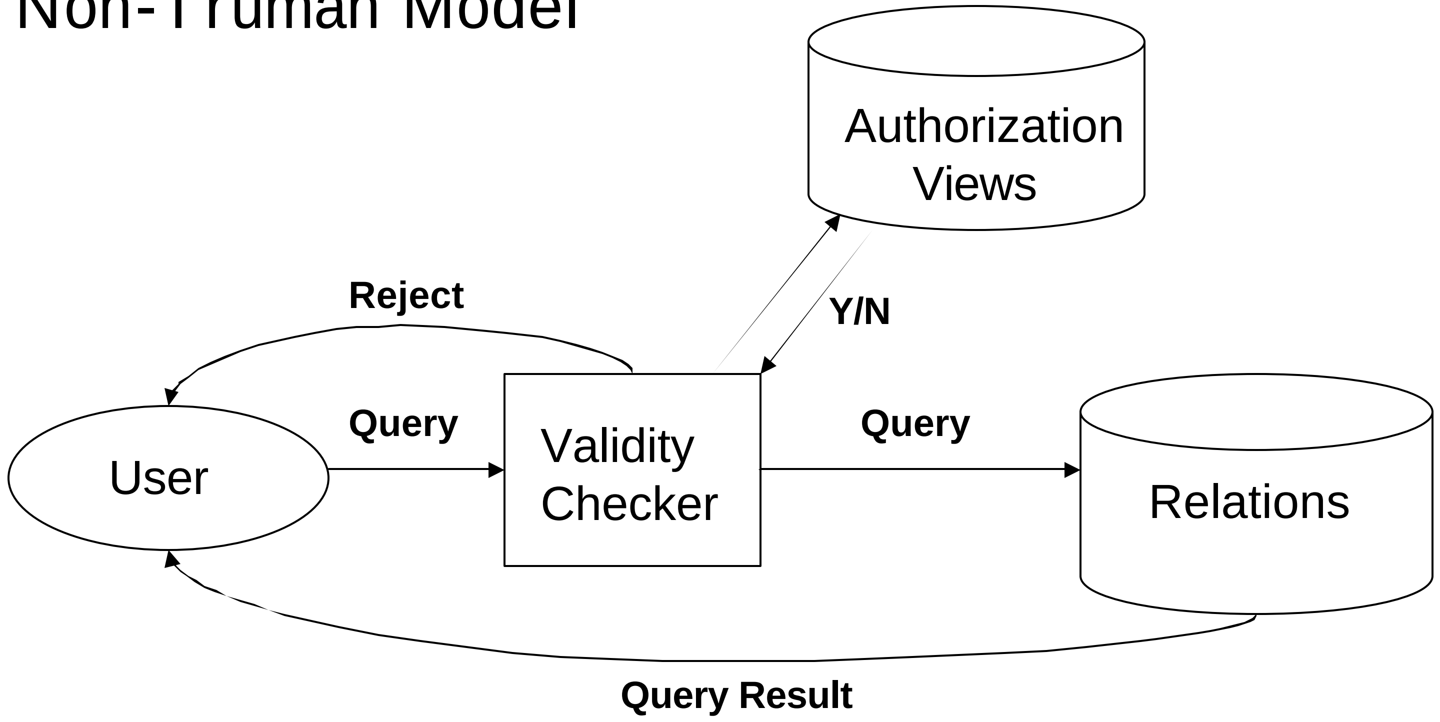
- User queries should be executed without any modification or rejected outright.

## Authorization Models

- Non-Truman Model
  - The query is subjected to a validity test.
  - If failed, the query is rejected and the user is notified about this.
  - If succeeded, the query is allowed to executed normally, without any modification.
  - The DBA creates several authorization views, one for each access policy, and grant them to users. Any of these views can testify for the validity of the user’s query.

# Authorization Models

- Non-Truman Model



## Authorization Models

- What is a valid query?
  1. Unconditional Validity: a query Q is said to be unconditionally valid if there is a query Q’ that is written using only the authorization views, and is equivalent to Q, that is, Q’ produces the same result as Q on all database states.

Ex. Select avg(grade) From Grades
    Where student-id = 11;
    → Provided that user-id = 11, this query is unconditionally valid.

## Authorization Models

- What is a valid query?

   2. Conditional Validity: a query Q is said to be conditionally valid in a database state D, if there is a query Q' that is written using only the authorization views, and is equivalent to Q on all database states that are PA-equivalent to D.

   Ex. Suppose average grades of courses that had more than 10 student enrolled are allowed to be accessed.

   Select ave(grade) from Grades where course-id = 101;

   ➔ The validity depends on the database state.

## Authorization Models

- Testing for Validity
  - Use Inference Rules
  - Lots of work done for optimization
  - We will not discuss it here

## Access Control by Query Modification

Question?