Introduction

- **Problems of current authorization semantics of SQL**
  - too complex due to triggers, objects, and other features.
  - Numerous special cases and unnecessary restrictions.
  - DBA must cope with diverse user communities.

- **Goal**
  - Reduce the ad hoc nature of authorization semantics.
  - Introduce explicit, simple, and formal principles.
  - Formalization and simplification start from practice.
Formalizing authorization in SQL

- A database consists of a set of objects
  - Objects: schemas, base tables, views, columns and procedures.
  - Each object has a set of actions that can be performed on it; e.g., select, update, insert, delete and execute.

- Operation: $(\alpha, O)$
  - Specifies a particular action $\alpha$ on a particular object $O$.

- ID (user): individuals, roles, groups, or Public.

- Privilege: $(\tau, \theta)$
  - Allows an ID $\tau$ to perform an operation $\theta$. 
Given a statement $S$, SQL implicitly defines a set of operations, $\text{OPS}(S)$, for checking authorization.

- That is, an ID $\tau$ is authorized to perform $S$ iff $\tau$ has a privilege for every operation in $\text{OPS}(S)$.
- $\text{OPS}(S)$ can be found by the following rules:
  - If $S$ is a query, $\text{OPS}(S)$ contains (select, $A$) for all columns $A$ mentioned in $S$.
  - If $S$ is an update, $\text{OPS}(S)$ contains (update, $A$) for each column $A$ being updated, plus (select, $B$) for all columns $B$ mentioned in $S$.
  - If $S$ is a call to routine $P$, $\text{OPS}(S)$ contains (execute, $P$), plus (select, $A$) for all columns $A$ mentioned in the argument list.
  - If $S$ contains a nested statement $S'$, $\text{OPS}(S)$ contains all operations of $S'$. 

Formalizing authorization in SQL
Formalizing authorization in SQL

- Example: Update T set A = C + 2
  where B1 in (select B2 from V)
  \( \Rightarrow \) \( \text{OPS}(S) = \{ (\text{select, T.B1}), (\text{select, T.C}), (\text{select, B.B2}), (\text{update, T.A}) \} \)

- If \( S \) is complex, the computation of \( \text{OPS}(S) \) may not be straightforward.
  - Unnecessary predicates; e.g., tautologies and constraints
  - select T.A from T where T.B is null or T.B * T.B >= 0
    - (select, T.B) should not be in \( \text{OPS}(S) \)
  - The detection of such predicates is not decidable; they are not considered.
Formalizing authorization in SQL

- **Grant**
  - An ID receives privileges via grant statements.
  - An ID is able to issue a grant statement for an operation if its privilege include a grant-option privilege for the operation.

- **Ownership**
  - When an object is created, the creator is given administrative authority over the object.
  - Two aspects: rights over the defined metadata and rights over the instance population
    1. Base table: the creator is given all possible privileges.
    2. Derived object: the creator is given full rights on the metadata and limited rights over the instance population.
Formalizing authorization in SQL

- Derived objects: procedures and views
  - Each derived object $Z$ has a defining statement, $\text{DEF}(Z)$.
  - Unlike base tables, when a derived object is created, the system infers the appropriate privileges based on the creator's privileges on underlying objects.
  - The general principle is that it is safe to infer privileges for tasks the user could accomplish by other means; i.e., inference may increase convenience, but not power.

- The SQL Inference Principle: Let $\theta$ be an operation on derived object $Z$. Then $Z$'s creator $\tau$ should receive privileges on $\theta$ provided that $\tau$'s ability to access and modify data does not increase.
Formalizing authorization in SQL

- Example: create view Z as select A, C from T where T.B > 2
  - Say the creator \( \tau \) has privileges on (select, T) and (update, T.A).
  - Then it is wrong to give \( \tau \) the privilege on (update, Z).
  - But it is okay to give \( \tau \) the privilege on (update, Z.A).

- Inferences are justified by using query modification
  - Take a statement \( S \) involving derived object Z, and produce an equivalent statement \( S' \) by replacing references to Z to tables in DEF(Z).
  - Select Z.A from Z \( \Rightarrow \) select T.A from T where T.B > 2
  - Thus, it would be wrong to give \( \tau \) an inferred privilege on (select, Z.A) unless \( \tau \) already has privileges on (select, T.A) and (select, T.B).
Formalizing authorization in SQL

- Query modification technique can provide a counterexample, but it cannot prove that an inference is correct.
  - We would have to examine every possible statement involving Z.

- Definition. Let Z be a derived object, and let θ be an operation on Z. OPS(θ) is found as follows:
  - OPS((select, Z.B)) consists of those operations (select, T.A) such that changing A-value of T can change the B-value of Z.
  - OPS((insert, Z.B)) consists of those operations (insert, T.A) if inserting into Z can cause an insertion into T, and Z.B is derived from T.A.
  - OPS((delete, Z)) consists of (delete, T) if deleting Z can cause a deletion from T.
  - OPS((update, Z.B)) consists of those operations (update, T.A) if updating the B-value of Z can cause a change in the A-value of T.
  - OPS((execute, P)) consists of the operations required to execute the body of P. That is, it contains each operation in OPS(DEF(P)).
Formalizing authorization in SQL

- The SQL Privilege Inference Rule: Let $\tau$ be the creator of derived object $Z$ and let $\theta$ be an operation on $Z$.
  1. Infer the privilege $(\tau, \theta)$ if $\tau$ has a privilege for every operation in $\text{OPS}(\theta)$.
  2. Infer the privilege $(\tau, \text{grant}\theta)$ if $\tau$ has grant-option privilege for every operation in $\text{OPS}(\theta)$.

- Theorem. The privileges inferred by this rule satisfies the SQL Inference Principle.
  - Proved in the paper
Proposed extension

- Inferred privileges on derived objects
  - In standard SQL, all privileges on a derived object stem from the creator.
  - The extension is to allow privileges on a derived object to be inferred to any ID, not just the object's creator.

- The inference Principle
  - Let $\theta$ be an operation on derived object $Z$. An ID $\tau$ receive privilege on $\theta$ as long as $\tau$'s ability to access and modify data does not increase.
Proposed extension

- Who may create a derived object?
  - As all privileges on a derived object stem from the creator, SQL does not allow an ID to create an object unless the creator receives a reasonable number of privileges.
  - Without this restriction, any user can create a derived object and receives whatever privileges the system infer.
  - However, the metadata (definition) of derived object must be explicitly controlled.
  - Introduce a new action, Visible.
  - Privilege on (visible, Z) allows ID to see Z's definition.
  - Now some users can use Z without knowing the definition of Z. Also the creator can allow some users to see the definition of Z without giving them privileges to use it.
Proposed extension: Benefits

- **Creators need not be administrators.**
  - Subjects with (visible, Z) and privileges on OPS(θ) are immediately able to use θ without any explicit grant by the creator.
  - The creator can give access to Z to anyone with sufficient authorization on the underlying object by granting (visible, Z) to Public.

- **Privileges can be kept consistent automatically.**
  - Consider a data warehouse, whose contents are a materialized view of its underlying source databases.
  - The proposed model provides a way to enforce consistency between the warehouse privileges and the source privileges.
Proposed extension: Benefits

- Explicit control over metadata privileges
  - SQL allows an ID with any privilege on an object to have the ability to see all metadata about the object; more is revealed than required.
  - A user with select privilege can see the constraints.
  - A user who can execute a procedure can see the definition.
  - In some cases, this is not desirable.

- Untrusted IDs can create useful derived objects.
  - As the creator of a derived object is the source of all privileges in SQL, only trusted users can create useful views.
  - In the proposed model, IDs can access the object even if the creator is untrusted or lazy.
Proposed extension: Benefits

- Invoker’s rights are integrated into the model.
  - The SQL standard requires that the creator of a procedure have grant-option privileges on all operations in the procedure.
  - Oracle introduced the invoker-right mechanism, which requires users have not only an execute privilege, but also all the privileges to execute operations in the procedure.
  - A contract programmer can write complex procedures and grant execute privilege to public. Then only the users having sufficient privileges can actually use the procedure.
  - The model extends invoker-right features beyond procedure to any operation $\theta$ on a derived object $Z$.
  - An administrator can choose to grant explicit privileges on $\theta$ to some IDs, and to allow possible inference of $\theta$ to other IDs by granting Visible privileges to them.
Another issue

- **Base table ownership**
  - It is beneficial to separate the metadata privileges on a derived object from the privileges on its content.
  - Is this separation possible for base tables? The creator obviously deserves all metadata privileges, but how do we assign the privileges on its content?
  - For example, a programmer or DBA can create a table, but should not have the right to see the data.
  - In SQL, one cannot remove the creator’s rights since deletion cascades.
  - A simple way is to provide a way to remove the rights from the creator without affecting their delegates. (non-cascading revoke)
Question?
View Security as the Basis for Data Warehouse Security

by Aaron Rosenthal and Edward Sciore (MITRE)

Ji-Won Byun
TruSe Reading Group
January 11, 2005
Introduction

- Problem
  - Currently, access permissions in a data warehouse are managed in a separate world from the sources’ policies.
  - The warehouse DBA has to manually specify access rights on all warehouse data.
  - The warehouse DBA must be trusted by all sources.
  - The consequences are inconsistencies, slow response to change, and a heavy work load for administrators.
  - Thus, the critical problem of data warehousing security is how to automatically coordinate the access rights of the warehouse with those of the sources.
Three extensions to SQL

1. Split the notion of “access permission on a table” into two separate issues:
   a. who is allowed to access what information (information permissions): enterprise-wide decision
      - Employee salary information is releasable to payroll clerks.
   b. who is allowed to access which physical tables (physical permissions): local decision
      - Payroll clerks are allowed to run queries on the warehouse.

2. Provide a powerful inference mechanism
   - In SQL, a user is allowed to execute a query Q if the user has permissions on all tables mentioned in Q.
   - In the proposed model, a user can also execute Q if there is an equivalent query Q’, called a witness for Q, for which the user has permissions.
Proposed extensions

- Three extensions to SQL
  3. Broaden the creation of views
    - In SQL, a view can be created only if there is a user that has Grant authority on all mentioned tables.
    - The extension allows the views to be over several mutually-suspicious sources, where no one is trusted to have Grant permission over all of them.
Basics

- Permission: (subject, operation, object, mode)
  - Subject: individual users, roles, group, process, etc.
  - Object: tables belonging to either a source or warehouse.
  - Operation: SQL operations (focus on Read and Grant-read).
  - Mode: either information or physical

- A view
  - Every warehouse table is a view over the tables exported by sources.
  - Defined by an SQL query Q.
  - The inputs to Q are the objects mentioned in Q: Q(T_1, ..., T_n).
Permissions

- A subject \( s \) is allowed to access a table only if \( s \) has both information and physical permissions on the table.
  - A permission \( (s, \text{op}, T, \text{“information”}) \) indicates that the content in \( T \) should be accessible to \( s \) for operation \( \text{op} \). It concerns releasability of knowledge, not physical access to \( T \). They are globally applied and unaffected by creation of redundant copies or new interface.
  - A permission \( (s, \text{op}, T, \text{“physical”}) \) authorizes an execution strategy to use a single physical resource; i.e., local policy.
Permission Inference

- A permission is explicit if it is granted directly by an authorized grantor.

- Permissions can be inferred by the system as well.
  - A subject should have permission to execute a query iff the query can be expressed in terms of tables (base or view) for which user has explicit permissions.

- Two useful definitions for inference
  - A query Q is equivalent to T if the output of Q always contains the same tuples as T.
  - A permission (s, op, T, mode) is implied if there exists an equivalent query Q(T1, ..., Tn) such that each permission (s, op, Ti, mode) has been granted explicitly. Query Q is called the witness for the implied permission.
Permission Inference

- An implied information permission \((s, \text{read}, T)\) means that the information in \(T\) is releasable to \(s\).
  - In effect, a subject need not care whether an information permission is explicit or implied, nor whether \(T\) is a materialized view or base table.

- An implied physical permission \((s, \text{read}, T)\) asserts that there exists at least one way to compute \(T\) for which the physical permissions are available.
  - If \(T\) is a materialized table, the subject does not have physical access to \(T\). The subject need to use the tables from the witness query.
Permission Inference

- Now, it is required for the systems to be able to a witness query equivalent to T.
  - Three important rewriting strategies
    1. View substitution: If a subject s has the necessary permissions on the source tables mentioned in a view, then s also has permission on the view. (SQL requires an explicit grant to access a view.)
    2. Semantic query optimization: If the user queries a view V, some source data that underlies V may be irrelevant to the query result. (Let V be a join of two tables. SQL requires permissions on both tables. This is not necessary in some cases.)
    3. Rewrite in terms of other views: Subjects are often given access to information though views when they do not have permissions on the base table.
  - A complete set of equivalents is impossible because the general rewrite problem is undecidable. Thus, the benchmark is to do better than others, rather than pursuing completeness.
Administering a warehouse

- SQL infers view privileges only when a view is defined.
  - The view definer receives the intersection of her privileges on the input tables.
  - Then the definer must explicitly specify all other permissions on the view.

- In the proposed model, view privileges are inferred whenever the view is accessed by any user.
  - Provides a more flexible way to coordinate the control of both the source administrator (controlling information permissions) and the warehouse administrator (controlling physical permissions).
Administering a warehouse

- **Computing local permissions**
  - When a query is issued to a warehouse, testing permissions should be performed there, not referring the sources.
  - The system needs to populate permission tables on the warehouse.
  - The set of users authorized to execute a query (not considering equivalence) is the intersection of the user sets of its inputs. Thus, the user set of a view can be determined by taking the union of these individual user sets for each equivalent query found.
  - Given T, first computes all queries equivalent to T, and structure as a DAG where each subexpression appears only once. Then traverse the graph (bottom up), computing permissions for each table in the graph from its predecessors.
Within-view permissions

- Consider a warehouse where multiple parts of an organization or multiple organizations participate.
  - With the current SQL, each organization must grant a warehouse DBA Read and Grant-read permissions on their exported data. Then a W-DBA define and administer a view over the combined information.
  - What happens if no one can be universally trusted?
  - One approach is to allow a source to stipulate that its exported data can be used only for computing a less sensitive view.
  - grant select to s on T within V
  - s can access T, but only from within V.
Within-view permissions

- **Examples**
  1. **Totals over a large set**
     - A warehouse supporting financial studies of hospitals.
     - Each hospital chooses to release its information within state-wide totals or city-wide totals.
  2. **Peer-to-peer intersection**
     - Table Entrant (border patrol): people entering the country
     - Table Wanted (Police): people who are wanted
     - Select * from Entrant, Wanted where match(Entrant, Wanted)
  3. **Intersection of child and parent**
     - Table Patient(P#, Age, ...): Parent table
     - Table Surgery(P#, Procedure, Date, ...): Child table
     - Select * from Patient, Surgery where Patient.P# = Surgery.P# and Patient.Age > 80
Within-view permissions

- **Semantics**
  - A within-view permission: (subject, operation, object, mode, view).
  - A subject $s$ is able to access view $V$ if $s$ has access to each input table within $V$.
  - The witness semantics is extended as follows:
    - A permission $(s, op, T, mode)$ is implied if there exists a query $Q$ and views $\{v_i\}$ equivalent to $T$, such that for each object $T_i$ mentioned in $Q$, either
      - The permission $(s, op, T_i, mode)$ has been explicitly granted, or
      - The within-view permission $(s, op, T_i, mode, V_i)$ has been explicitly granted, where $Q$ is equivalent to $V_i$.
  - Users are able to access a view even when nobody is trusted to receive permissions to all underlying tables.
Question?