Scenario

Recent advances in the communications and information technology have led to the widespread development of emerging scenarios:

- involving dynamic collaboration and process integration
- involving sharing of information from different authorities
- involving multiplicity and heterogeneity of entities and security specifications
- in a possible mobile/dynamic environment
- parties may be unknown or not fully trusted
- data/services outsourcing
- cloud computing
Several contributions

The research community has been very active and produced several contributions and advancements in policy specification and enforcement, e.g.,:

- Powerful and flexible authorization languages and frameworks
- RBAC variants for multi-domain environments
- Policies for dynamic coalitions and virtual communities
- Privacy-enhanced access control and identity management
- Policy negotiation and trust management
- Policy composition
- Policies for ubiquitous and pervasive systems

Data access and query executions

Data access and query execution are more complex in emerging scenarios

- Data may be stored outside the data owner’s control
- Application/query executions may entail access to data under control of different parties
- Data can move around to different locations

⇒ Specification and enforcement of policies for regulating query execution in distributed multi-authority scenarios
Overview

- Distributed query processing
- Sovereign joins
- Access patterns
- View-based access control
- Authorization enforcement in distributed query evaluation
- Authorization composition for query execution

Distributed query processing

Define different strategies for executing distributed queries depending on the kinds of distributed systems [Kossmann, ACM CSUR 2000]

- Client-server, peer-to-peer, and multitier characterized by a client that makes a request and servers that respond to the request
  - shipping: data, query, hybrid, …
  - optimization: operators are executed at the client, server, both
  - execution: row blocking, semijoin, …

- Heterogeneous database systems characterized by component systems with different capabilities
A sovereign join operation computes a join in a way that nothing beyond the query result is revealed [Agrawal et al, ICDE 2006]

Scenario:
- the owners of the two relations do not trust each other, none of them should view the relation of the other or the query result
- the party posing the query is different from the two owners
- the server executing the query is not trusted for confidentiality
- the server is equipped with a tamper-resistant secure coprocessor that is authorized to view the content of the two relations

Examples:
- check that airline passengers are not blacklisted by a federal agency
- check DNA sequences against patient records

Query execution:
- the client sends the join operation to the server
- the server has the encrypted version of the original relations
- the server sends the query and the encrypted relations to the coprocessor
- the coprocessor decrypts the relations and executes the join
- the coprocessor encrypts the join result with a key shared with the client and returns the encrypted results to the server
- the server sends the encrypted result to the client

High computational cost and need for a trusted component
Specify limitations on how information sources can be accessed (e.g., [Cali et al, ICDE 2008])

- Support for scenarios where one must provide values for one of the attributes of a relation to obtain data
- Example: in a web-based application one must fill in certain fields for retrieving data
- Each relation/view is associated with an access pattern, where each attribute of the relation/view has a i (input) or o (output) value
- Relations can be accessed only according to the corresponding access patterns

Access patterns – 2 (Example)

- Relations:
  \[ \text{Insurance}^{oi}(\text{holder,plan}) \]
  \[ \text{Hospital}^{ooo}(\text{patient,YoB,disease,physician}) \]
  \[ \text{Nat\_registry}^{ioo}(\text{citizen,YoB,healthaid}) \]
- Query:
  \[
  \text{SELECT } \text{patient} \\
  \text{FROM } \text{Insurance} \text{ JOIN } \text{Hospital} \text{ ON } \text{holder=}\text{patient} \\
  \text{WHERE } \text{plan} = \text{"annual"}
  \]
- The query cannot be answered in the traditional way since input attribute \text{YoB} of \text{Hospital} is not bounded by a value
  - access \text{Insurance} to retrieve holders
  - holders as input to access \text{Nat\_Registry} and retrieve years
  - years as input to access \text{Hospital} and retrieve patients
Access patterns – 3

- Provide query evaluation and optimization in the presence of access patterns
- Queries are typically represented in terms of Datalog
- Twofold goal:
  - identify the types of queries that given access patterns can support
  - determine a query plan that matches the access patterns of the involved relations, while minimizing some cost parameters
- Consider only two actors (the data owner and a single requesting user)
- Do not address access control but could be exploited for that

View-based authorizations – 1

- Provide fine-grained content-dependent access control in relational databases (e.g., [Motro, JIIS 1989; Rosenthal and Sciore, DBSec 2001; Rizvi et al., SIGMOD 2004])
- Depart from having users to explicitly query views
- Allow queries to be written in an authorization-transparent way, two approaches:
  - query modification to answer queries (Truman model)
  - a query is valid if it can be answered using information in the authorization views available to the requesting user (non-Truman model)
View-based authorizations – 2 (Example)

- Relations:
  
  Treatment(ssn,iddoc,type,cost,duration)
  
  Doctor(iddoc,name,specialty)
  
- Integrity constraint: each treatment is supervisioned by a doctor
  
- Authorization view:
  
  CREATE AUTHORIZATION VIEW TreatDoct AS
  
  SELECT D.name, T.type, T.cost
  
  FROM Treatment AS T, Doctor AS D
  
  WHERE T.iddoc=D.iddoc

- Query: SELECT type, cost FROM Treatment

  The projection of TreatDoct on T.type, T.cost is equivalent to
  the query $\Rightarrow$ the query is valid

View-based authorizations – 3

- Authorizations are expressed as parameterized views and access pattern views [Rizvi et al., SIGMOD 2004], e.g.:
  
  - a patient can see only her hospitalization data (not those of other patients)
  
  - a doctor can see the data of any patient if she provides the patient-id (she cannot retrieve data of all patients together)

- If there exists a query written only by using the instantiated authorized views of the requesting user that is equivalent to the original query, the query is accepted

- Queries that violate the authorizations available to a user are rejected

- View composition is used to check whether a query is valid
View-based authorizations – 4

- Support queries in an authorization-transparent way
- Support of content-dependent authorizations and access patterns
- Analyze integration with DBMS optimizers
- Focuses on instantiated queries (where attributes are bound to specific values)
- Focuses on low-level enforcement aspects
- Relies on heuristics enforced by the query optimizer, and does not provide composition algorithms and security guarantees

Distributed Query Evaluation under Protection Requirements

Scenario

- **Distributed database system**
  - assume a relational database (not a limitation since relational databases are the core of any Web service)

- **Relations** are distributed at different servers

- **Query execution** may require cooperation and information sharing among different servers

- Each **server** is responsible for the definition of the access policy on its resources

Problem addressed

Regulate views and information sharing among different parties

- support the collaboration among parties in distributed query execution on data subject to selective release

- define authorized views based on information content of a relation

- assign operations within the query to different parties in a way that is **safe** with respect to information that can be viewed by parties
Data model

- $\mathcal{R}$: set of relations $R_i(A_{1i}, \ldots, A_{ni})$, with $K_i \subseteq \{A_{1i}, \ldots, A_{ni}\}$ primary key of $R_i$

- $I$: set of referential integrity constraints $\langle F_j, K_i \rangle$:
  - foreign key $F_j$ can assume only values in $R_i[K_i]$

- $J$: set of joins, $J = \langle A_l, A_r \rangle$ that combine tuples in different relations based on equality conditions
  - $joinpath(R_1, \ldots, R_n)$: sequence $J_1 = \langle A_{l1}, A_{r1} \rangle$, $\ldots$, $J_{n-1} = \langle A_{ln-1}, A_{rn-1} \rangle$ of joins connecting relations

- SQL queries: SELECT $A$ FROM $\textit{Joined relations}$ WHERE $C$
  - $\pi_A(\sigma_C(R_1 \bowtie_{J_1} \ldots \bowtie_{J_{n-1}} R_n))$
    - $A$: set of attributes
    - $C$: selection conditions
    - $\textit{Joined relations}$: $R_1 \bowtie_{J_1} \ldots \bowtie_{J_{n-1}} R_n$

- Query tree plan $T$: tree representing the query execution

Example of distributed relations
Example of distributed relations

Example of join path for query execution:
\{\langle \text{citizen, patient} \rangle, \langle \text{disease, illness} \rangle \}\n
Example of query tree plan

```
SELECT patient, physician, plan, healthaid
FROM Insurance JOIN Nat_registry ON holder=citizen
JOIN Hospital ON citizen=patient

\begin{tikzpicture}
  \begin{scope}[every node/.style={circle, draw}]
    \node (I) at (0,0) {\text{Insurance}};
    \node (H) at (4,0) {\text{Hospital}};
    \node (N) at (0,-2) {\text{Nat\_registry}};
    \node (D) at (4,-2) {\text{Disease\_list}};
    \end{scope}
    \begin{scope}[every edge/.style={draw}]
      \draw (I) edge (H)
      edge (N)
      \draw (H) edge (D)
      \draw (N) edge (D)
      \draw (I) edge[loop above] (I)
      \draw (H) edge[loop above] (H)
      \draw (N) edge[loop above] (N)
      \draw (D) edge[loop above] (D)
      \end{scope}
\end{tikzpicture}
```
Permissions

Permission: \([\text{Attributes, Join Path}] \rightarrow \text{Subject}\)
- authorizes release to Subject of set Attributes of attributes resulting from the Join Path

Examples
- \([\text{holder}, \text{plan}], \_ \rightarrow S_N\)
- \([\text{holder}, \text{plan}, \text{patient}, \text{physician}]), (\langle I. \text{holder}, H. \text{patient} \rangle) \rightarrow S_I\)

<table>
<thead>
<tr>
<th>Insurance(\text{holder, plan})</th>
<th>Hospital(\text{patient, disease, physician})</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nat_registry(\text{citizen, healthaid})</td>
<td>Disease_list(\text{illness, treatment})</td>
</tr>
</tbody>
</table>

Relations in permissions – 1

Join Path may involve relations that do not have attributes in Attributes
- connectivity constraints: to build the association among attributes
  - \([\text{holder}, \text{treatment}]), (\langle I. \text{holder}, H. \text{patient} \rangle, \langle H. \text{disease}, D. \text{illness} \rangle) \rightarrow S_I\)
- instance-based restrictions: to restrict the values of attributes to be released
  - \([\text{holder}, \text{plan}]), (\langle I. \text{holder}, H. \text{patient} \rangle) \rightarrow S_I\)

<table>
<thead>
<tr>
<th>Insurance(\text{holder, plan})</th>
<th>Hospital(\text{patient, disease, physician})</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nat_registry(\text{citizen, healthaid})</td>
<td>Disease_list(\text{illness, treatment})</td>
</tr>
</tbody>
</table>
A join path in a permission:
- implies the release of fewer tuples
  does not imply the release of less information
  - \[ (\text{holder}, \text{plan}), \_ \rightarrow S_I \]
  - \[ (\text{holder}, \text{plan}), (\text{l.holder}, \text{H.patient}) \rightarrow S_I \]

- ineffective when reachable via referential integrity constraints
  - \[ (\text{patient}, \text{disease}, \text{physician}), \_ \rightarrow S_I \]
  - \[ (\text{patient}, \text{disease}, \text{physician}), ((\text{N.citizen}, \text{H.patient}), (\text{H.disease}, \text{D.illness})) \rightarrow S_I \]

### Relation profile

- Capture the information content of either a base or derived (i.e., computed by a query) relation

- The relation profile of \( R \) is a triple \([ R^\pi, R^{\bowtie}, R^\sigma ]\), where:
  - \( R^\pi \): \( R \)'s schema
  - \( R^{\bowtie} \): join path (possibly empty) used in the definition of \( R \);
  - \( R^\sigma \): set of attributes (possibly empty) involved in selection conditions of \( R \)

<table>
<thead>
<tr>
<th>Operation</th>
<th>( R^\pi )</th>
<th>( R^{\bowtie} )</th>
<th>( R^\sigma )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( R := \pi_X (R_i) )</td>
<td>( X )</td>
<td>( R_i^{\bowtie} )</td>
<td>( R_i^\sigma )</td>
</tr>
<tr>
<td>( R := \sigma_X (R_i) )</td>
<td>( R_i^\pi )</td>
<td>( R_i^{\bowtie} )</td>
<td>( R_i^\sigma \cup X )</td>
</tr>
<tr>
<td>( R := R_i \bowtie \cup R_r )</td>
<td>( R_i^{\bowtie} \cup R_r^{\bowtie} )</td>
<td>( R_i^{\bowtie} \cup R_r^{\bowtie} \cup J )</td>
<td>( R_i^\sigma \cup R_r^\sigma )</td>
</tr>
</tbody>
</table>
Relation profile – Example

```
SELECT illness
FROM Disease_list JOIN Hospital ON illness=disease
WHERE treatment = 'antihistamine'
```

Profile: \([R^\pi, R^{\bowtie\triangledown}, R^\sigma]\
\left[\text{illness}, \langle \text{D.illness,H.disease} \rangle, \text{treatment} \right] \]

---

Authorized view

- Subject \(S\) is **authorized to view** a relation \(R\) iff \(∃ \left[\text{Attributes, Join Path}\right] → S: R^\pi \cup R^\sigma ⊆ \text{Attributes} \) and \(R^{\bowtie\triangledown} = \text{Join Path}\)

- **Examples**
  - \(S_D\) requires \(R\) :
    ```
    SELECT illness
    FROM Disease_list JOIN Hospital ON illness=disease
    WHERE treatment='antihistamine'
    ```
  - Relation profile: \([\text{illness}, \langle \text{D.illness,H.disease} \rangle, \text{treatment}] \]
  - Authorization \([\text{illness,treatment}, \langle \text{D.illness,H.disease} \rangle] → S_D \)
    authorizes the query
  - Authorization \([\text{illness,treatment}, \_\_] → S_D \)
    does not authorize the query
Authorized data releases

- Each subject $S_i$ is authorized to view the relation $R_i$ it stores

- **Unary operations** can be executed by subject $S$ storing the relation
  - projection: $\pi_X(R)$
  - selection: $\sigma_X(R)$

- **A join operation** can be executed only if it entails release of authorized views
  - $R_i \bowtie J_{lr} R_r$ can be evaluated as a regular join or as a semi-join
  - $S_l$ and $S_r$ can play two roles: master computes the join; slave cooperates with the master for the computation

---

Regular joins – Example ([$S_I$, NULL])

- $S_I$ requires $R$: $\pi_{\text{holder, plan, disease, physician}} (\text{Insurance} \bowtie_{\text{holder=}\text{patient}} \text{Hospital})$

- Authorization: [($\text{patient, disease, physician}$), _] $\rightarrow S_I$

- $S_H$

<table>
<thead>
<tr>
<th>$S_I$</th>
<th>$S_H$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Insurance</strong> (holder, plan)</td>
<td><strong>Hospital</strong> (patient, disease, physician)</td>
</tr>
<tr>
<td><strong>Nat registry</strong> (citizen, healthaid)</td>
<td><strong>Disease list</strong> (illness, treatment)</td>
</tr>
</tbody>
</table>
Regular joins – Example ([\(S_I, \text{NULL}\)])

- \(S_I\) requires \(R:\ \pi_{\text{holder, plan, disease, physician}} (\text{Insurance} \bowtie_{\text{holder} = \text{patient}} \text{Hospital})\)
- Authorization: \([(\text{patient, disease, physician}), \_] \rightarrow S_I\)

\[
\begin{array}{c}
\text{Insurance(} \text{holder, plan}) \\
\text{Nat\_registry(} \text{citizen, healthaid})
\end{array}
\begin{array}{c}
\text{Hospital(} \text{patient, disease, physician}) \\
\text{Disease\_list(} \text{illness, treatment})
\end{array}
\]

© Pierangela Samarati

Regular joins – Example ([\(S_I, \text{NULL}\)])

- \(S_I\) requires \(R:\ \pi_{\text{holder, plan, disease, physician}} (\text{Insurance} \bowtie_{\text{holder} = \text{patient}} \text{Hospital})\)
- Authorization: \([(\text{patient, disease, physician}), \_] \rightarrow S_I\)

\[
\begin{array}{c}
\text{Insurance(} \text{holder, plan}) \\
\text{Nat\_registry(} \text{citizen, healthaid})
\end{array}
\begin{array}{c}
\text{Hospital(} \text{patient, disease, physician}) \\
\text{Disease\_list(} \text{illness, treatment})
\end{array}
\]

© Pierangela Samarati
Regular joins – Example ([SI,NULL])

- \( S_I \) requires \( R: \pi_{\text{holder,plan,disease,physician}} \) (\( \text{Insurance} \bowtie_{\text{holder}=\text{patient}} \text{Hospital} \))
- Authorization: \( [(\text{patient,disease,physician}),_] \rightarrow S_I \)

```
<table>
<thead>
<tr>
<th>Insurance(h)</th>
<th>Hospital(p,d)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nat_registry(c)</td>
<td>Disease_list(i,t)</td>
</tr>
</tbody>
</table>
```

© Pierangela Samarati

Semi-joins – Example ([SI,SH])

- \( S_I \) requires \( R: \pi_{\text{holder,plan,disease,physician}} \) (\( \text{Insurance} \bowtie_{\text{holder}=\text{patient}} \text{Hospital} \))
- Authorizations:
  - \( [(\text{holder}),_] \rightarrow S_H \)
  - \( [(\text{holder,plan,patient,disease,physician}),(\{I.h, H.p\})] \rightarrow S_I \)

```
<table>
<thead>
<tr>
<th>Insurance(h)</th>
<th>Hospital(p,d)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nat_registry(c)</td>
<td>Disease_list(i,t)</td>
</tr>
</tbody>
</table>
```

© Pierangela Samarati
Semi-joins – Example \([\{S_I, S_H\}]\)

- \(S_I\) requires \(R\): \(\pi_{\text{holder,plan,disease,physician}}(\text{Insurance} \bowtie_{\text{holder=patient}} \text{Hospital})\)
- Authorizations:
  \[\text{InsHolders} := \pi_{\text{holder}}(\text{Insurance})\]

\[\begin{align*}
\text{Insurance}(\text{holder,plan}) & \quad \text{Hospital}(\text{patient,disease,physician}) \\
\text{Nat\_registry}(\text{citizen,healthaid}) & \quad \text{Disease\_list}(\text{illness,treatment})
\end{align*}\]

© Pierangela Samarati 30/55

Semi-joins – Example \([\{S_I, S_H\}]\)

- \(S_I\) requires \(R\): \(\pi_{\text{holder,plan,disease,physician}}(\text{Insurance} \bowtie_{\text{holder=patient}} \text{Hospital})\)
- Authorizations:
  \[\text{InsHolders} := \pi_{\text{holder}}(\text{Insurance})\]

\[\begin{align*}
\text{Insurance}(\text{holder,plan}) & \quad \text{Hospital}(\text{patient,disease,physician}) \\
\text{Nat\_registry}(\text{citizen,healthaid}) & \quad \text{Disease\_list}(\text{illness,treatment})
\end{align*}\]

© Pierangela Samarati 30/55
Semi-joins – Example ([SI, SH])

- SI requires R: \( \pi_{\text{holder, plan, disease, physician}} (\text{Insurance } \bowtie_{\text{holder} = \text{patient}} \text{Hospital}) \)
- Authorizations:
  \[ ((\text{holder}), \_ ) \rightarrow SI \]
  \[ ((\text{holder, plan, patient, disease, physician}), ((\text{I}, \text{holder}, \text{H}, \text{patient}))) \rightarrow SI \]

\[ \text{InsHolders} := \pi_{\text{holder}} (\text{Insurance}) \]
\[ \text{InsHos} := \text{InsHolders} \bowtie_{\text{holder} = \text{patient}} \text{Hospital} \]

- SH requires R: \( \pi_{\text{patient, disease, physician}} (\text{Hospital}) \)
- Authorizations:
  \[ ((\text{holder}), \_ ) \rightarrow SH \]
  \[ ((\text{holder, plan, patient, disease, physician}), ((\text{I}, \text{holder}, \text{H}, \text{patient}))) \rightarrow SI \]

\[ \text{InsHos} := \text{InsHolders} \bowtie_{\text{holder} = \text{patient}} \text{Hospital} \]

\( \odot \text{Pierangela Samarati} \)

Semi-joins – Example ([SI,SH])

• **SI** requires **R**: \(\pi_{\text{holder, plan, disease, physician}} (\text{Insurance} \bowtie_{\text{holder} = \text{patient}} \text{Hospital})\)

• **Authorizations**:
  
  \[\{ (\text{holder}), \_ \} \rightarrow_{\text{SH}} \{ (\text{holder, plan, patient, disease, physician}), (\{ \text{l, holder, H, patient} \}) \} \rightarrow_{\text{SI}} \]

\[\text{InsHolders} := \pi_{\text{holder}} (\text{Insurance}) \]

\[\text{InsHos} := \text{InsHolders} \bowtie_{\text{holder} = \text{patient}} \text{Hospital} \]

\[\text{Insurance}(\text{holder, plan}) \quad \text{Hospital}(\text{patient, disease, physician}) \quad \text{Nat registry}(\text{citizen, healthaid}) \quad \text{Disease list}(\text{illness, treatment})\]

© Pierangela Samarati

Semi-joins – Example ([SI,SH])

• **SI** requires **R**: \(\pi_{\text{holder, plan, disease, physician}} (\text{Insurance} \bowtie_{\text{holder} = \text{patient}} \text{Hospital})\)

• **Authorizations**:
  
  \[\{ (\text{holder}), \_ \} \rightarrow_{\text{SH}} \{ (\text{holder, plan, patient, disease, physician}), (\{ \text{l, holder, H, patient} \}) \} \rightarrow_{\text{SI}} \]

\[\text{InsHolders} := \pi_{\text{holder}} (\text{Insurance}) \]

\[\text{InsHos} := \text{InsHolders} \bowtie_{\text{holder} = \text{patient}} \text{Hospital} \]

\[\pi_{\text{holder, plan, disease, physician}} (\text{Insurance} \bowtie_{\text{holder} = \text{InsHos}}) \]

\[\text{Insurance}(\text{holder, plan}) \quad \text{Hospital}(\text{patient, disease, physician}) \quad \text{Nat registry}(\text{citizen, healthaid}) \quad \text{Disease list}(\text{illness, treatment})\]

© Pierangela Samarati
Feasible query plan

- **Goal**: given a query tree plan, determine for each operation a subject (pair of subjects) responsible for the execution such that all views are authorized.

- **Executor assignment function** $\lambda_T$:
  - leaf node $n$: $\lambda_T(n) = [S, \text{NULL}]
  - $n$ in $\{\pi, \sigma\}$: $\lambda_T(n) = [S, \text{NULL}]
  - $n$ in $\{\bowtie\}$: $\lambda_T(n) = [\text{master}, \text{slave}]
    - $\text{master} \in \{S_l, S_r\}$, $\text{slave} \in \{S_l, S_r, \text{NULL}\}$, and $\text{master} \neq \text{slave}$

- $\lambda_T$ is a safe executor assignment iff $\forall n \in N$, $\lambda_T(n)$:
  - $n$ is a leaf
  - $n$ is $\pi$ or $\sigma$
  - $n$ is $\bowtie$ and all the views entailed by the assignment are authorized

---

Example of executor assignment

```
SELECT patient, physician, plan, healthaid
FROM Insurance JOIN Nat_registry ON holder=citizen
JOIN Hospital ON citizen=patient
```

---

<table>
<thead>
<tr>
<th>Insurance(holder, plan)</th>
<th>Hospital(patient, disease, physician)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nat_registry(citizen, healthaid)</td>
<td>Disease_list(illness, treatment)</td>
</tr>
</tbody>
</table>
Example of executor assignment

SELECT patient, physician, plan, healthaid
FROM Insurance JOIN Nat_registry ON holder=citizen
JOIN Hospital ON citizen=patient

©Pierangela Samarati
32/55
Algorithm

- Possible allocation algorithm

  1. **Find_candidates**: post-order visit of $T$ to push up potential candidates (including third parties)

  2. **Assign_ex**: pre-order visit of $T$ to push down selected executors

     - prefer semi-join to full-join

     - prefer the subject that is involved in a higher number of join operations

---

**Authorized views: Syntactic vs semantic**

What does it means for a subject (server/user) to be authorized to see the result of a query?

Two approaches:

- **Syntactic.** Subject $S$ is **authorized to view** a relation $R$ iff $\exists [\text{Attributes, Join Path}] \rightarrow S: R^\pi \cup R'^\alpha \subseteq \text{Attributes}$ and $R^{\text{Join}} = \text{Join Path}$

  + simple

  - limited or burden in authorization specification

- **Semantic.** Subject $S$ is **authorized to view** a relation $R$ iff $S$ has permissions to view the information content carried by the relation

  - a query should be authorized if the set of permissions available to the subject would allow the subject to independently compute the query result
Authorized views: Syntactic vs semantic – Example

- Authorizations:
  $[(\text{holder}, \text{plan}),_] \rightarrow S_I$
  $[(\text{patient}, \text{disease}),_] \rightarrow S_I$

- $S_I$ requires $R$:
  $$\text{SELECT} \ \text{holder}, \ \text{disease}$$
  $$\text{FROM} \ \text{Insurance} \ \text{JOIN} \ \text{Hospital} \ \text{ON} \ \text{holder} = \text{patient}$$

- Profile: $[(\text{holder,disease}), (\{I.\text{holder}, H.\text{patient}\}),_]$

**Syntactic** approach: the query is not authorized since the authorizations does not explicitly permit the join between Insurance and Hospital

**Semantic** approach: the query is authorized since $S_I$ could compute the query result

<table>
<thead>
<tr>
<th>Insurance(\text{holder,plan})</th>
<th>Hospital(\text{patient,disease,physician})</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nat_registry(\text{citizen,healthaid})</td>
<td>Disease_list(\text{illness,treatment})</td>
</tr>
</tbody>
</table>

© Pierangela Samarati

---

Access control with the semantics approach

- A query should be authorized if the set of permissions available to the subject would allow the subject to independently compute the query result.

- Need to support composition of permissions while ensuring no indirect release:
  - conditions on attributes that are not returned
    - captured in the definition of profile
  - join conditions restricting the tuples returned by the query
    - captured based on the coloring of the graphs that we will use to model database schema, permissions, and queries.

Simplifying assumptions

- Schema is acyclic

- Attributes that can be joined appear with the same name in the different relations (natural joins)

- Restrict analysis to a subject:
  - permission $[\text{Attr}, \text{Join Path}] \rightarrow \text{Subject}$ can be viewed as $[\text{Attr}, \text{Rel}]$, where $\text{Rel}$ are the relations involved in Join Path

- Relation profile $[R^\pi, R^{\bowtie}, R^\sigma]$ can be viewed as $[\text{Attr}, \text{Rel}]$:
  - $\text{Attr} = R^\pi \cup R^\sigma$
  - $\text{Rel}$ are the relations involved in join path $R^{\bowtie}$
Schema graph

Schema graph of a set $\mathcal{R}$ of relations is a mixed graph $G(N,E)$

- $N = \{R_i.\ast \mid R_i \in \mathcal{R}\}$ corresponds to the set of attributes

- $E = J \cup I \cup \{(R_i.K,R_i.A) \mid R_i \in \mathcal{R} \land A \notin K\}$ corresponds to:
  - natural joins (non-oriented)
  - referential integrity constraints (oriented)
  - functional dependencies (oriented)

---

Schema graph – Example

$\mathcal{R}$

<table>
<thead>
<tr>
<th>Employee(ssn,job,salary)</th>
<th>Patient(ssn,dob,race)</th>
<th>Treatment(ssn,iddoc,type,cost,duration)</th>
<th>Doctor(iddoc,name,specialty)</th>
</tr>
</thead>
</table>

\[ E \quad I \quad J \]

Diagram:

- $E_{ssn}$, $E_{job}$, $E_{salary}$
- $P_{ssn}$, $P_{dob}$, $P_{race}$
- $T_{ssn}$, $T_{iddoc}$, $T_{type}$, $T_{cost}$, $T_{duration}$
- $D_{iddoc}$, $D_{name}$, $D_{specialty}$
Views

Permissions and queries \([Attr, Rel]\) are views over set of relations \(R\)

- \(Rel\) extended through referential integrity constraints
- **Closure** \(Rel^*\): relations obtained by closing \(Rel\) with respect to referential integrity constraints (ineffective joins)

**Example**
- \(Rel = \{\text{Treatment}\}\)
- \(Rel^* = \{\text{Treatment, Patient, Doctor}\}\)

<table>
<thead>
<tr>
<th>Employee(ssn, job, salary)</th>
<th>Treatment(ssn, iddoc, type, cost, duration)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Patient(ssn, dob, race)</td>
<td>Doctor(iddoc, name, specialty)</td>
</tr>
</tbody>
</table>

\(G_V(N, E, \lambda_V)\) **view graph** of \(V = [Attr, Rel]\): obtained by coloring \((\lambda_V)\) the schema graph

- **black**: information that view carries
  - attributes in \(Attr\)
  - arcs in \(joinpath(\ Rel^*)\) or going from the key of a relation in \(Rel^*\) to attributes in \(Attr\) or \(joinpath(\ Rel^*)\)

- **white**: non-black attributes of relations in \(Rel^*\) and arcs connecting them to the primary key

- **clear**: any other attribute or arc
View graph – 1 (Examples)

\[(\text{ssn, dob, race}), \text{(Patient)}] \rightarrow \text{Alice}

View graph – 1 (Examples)
[(ssn, dob, race), (Patient)] → Alice

© Pierangela Samarati
View graph – 1 (Examples)

\[\{(\text{ssn,dob,race}), (\text{Patient})\} \rightarrow \text{Alice}\]

\[\{(\text{ssn,type,cost,duration}), (\text{Treatment})\} \rightarrow \text{Alice}\]

© Pierangela Samarati
SELECT E.ssn, salary
FROM Employee AS E JOIN Patient AS P ON E.ssn = P.ssn
JOIN Treatment AS T ON T.ssn = P.ssn
WHERE cost > 250
Authorizing permissions

Permission $p$ authorizes query $q$ iff the direct or indirect information released by $q$ is a subset of the information authorized by $p$

- $p=[Attr_p,Rel_p]$ is applicable to $q=[Attr_q,Rel_q]$ iff $Rel_p^* \subseteq Rel_q^*$
  - i.e., the permission refers to the complete set of tuples requested by the query (it does not contain additional joins apart from those corresponding to referential integrity constraints)

- $p=[Attr_p,Rel_p]$ authorizes $q=[Attr_q,Rel_q]$ iff
  - $p$ is applicable to $q$
  - $G_q$ and $G_p$ have the same black join/referential integrity arcs
  - all nodes that are black in $G_q$ are also black in $G_p$

Applicable, authorizing permission – Example

<table>
<thead>
<tr>
<th>Query</th>
<th>Permission</th>
</tr>
</thead>
<tbody>
<tr>
<td>E.ssn E.job E.salary</td>
<td>E.ssn E.job E.salary</td>
</tr>
<tr>
<td>P.ssn P.dob P.race</td>
<td>P.ssn P.dob P.race</td>
</tr>
<tr>
<td>T.ssn T.iddoc T.type T.cost T.duration</td>
<td>T.ssn T.iddoc T.type T.cost T.duration</td>
</tr>
<tr>
<td>D.iddoc D.name D.specialty</td>
<td>D.iddoc D.name D.specialty</td>
</tr>
</tbody>
</table>

Authorized | Applicable
Non applicable permission – Example

Applicable, non authorizing permission – Example
Composition of permissions

A query can be executed if the subject has permissions to view the information content carried by the query

- A query should be authorized if the set of permissions available to the subject would allow the subject to independently compute the query result

- A single permission may not be sufficient to authorize a query

\[ \Rightarrow \text{composition of permissions} \]

- Need to be careful not to enact indirect disclosure

---

Composition of permissions – 1 (Examples)

<table>
<thead>
<tr>
<th>Query</th>
<th>Permissions</th>
<th>Authorized</th>
</tr>
</thead>
<tbody>
<tr>
<td>E.ssn E.job E.salary</td>
<td>E.ssn E.job E.salary</td>
<td></td>
</tr>
<tr>
<td>P.ssn P.dob P.race</td>
<td>P.ssn P.dob P.race</td>
<td></td>
</tr>
<tr>
<td>T.ssn T.iddoc T.type T.cost T.duration</td>
<td>T.ssn T.iddoc T.type T.cost T.duration</td>
<td></td>
</tr>
<tr>
<td>D.iddoc D.name D.specialty</td>
<td>D.iddoc D.name D.specialty</td>
<td></td>
</tr>
</tbody>
</table>

©Pierangela Samarati
Two permissions can be safely composed iff their composition does not add information

- $p_i \rightarrow p_j$: $p_j = [\text{Attr}_j, \text{Rel}_j]$ depends on $p_i = [\text{Attr}_i, \text{Rel}_i]$ iff $\exists$ a path from nodes corresponding to attributes in $\text{Attr}_i \cap \text{Attr}_j$ to all black nodes in $G_{p_j}$

- Two permissions $p_i$ and $p_j$ can be safely composed iff at least one of $p_i \rightarrow p_j$ or $p_i \rightarrow p_j$ holds

- Composed permission $p_i \otimes p_j = [\text{Attr}_i \cup \text{Attr}_j, \text{Rel}_i \cup \text{Rel}_j]$
  - graph coloring of composed permission derived from colors of operands
Safe composition – Example

Permissions

Composed permission

Non Composable
Access control enforcement

1. Determine set of permissions \( \mathcal{P} \) applicable to query \( q \)
2. Compute closure on composition of \( \mathcal{P} \) (fixpoint computation)
3. \( q \) is authorized \( \iff \exists p \in \mathcal{P}^\otimes \) such that \( p \) authorizes \( q \)

Efficiency guaranteed by exploiting permission implication

- If \( p_j \rightarrow p_i \), \( \forall p_k \in \mathcal{P} \):
  - \( p_j \rightarrow p_k \Rightarrow (p_i \otimes p_j) \rightarrow p_k \)
  - \( p_k \rightarrow p_j \Rightarrow p_k \rightarrow (p_i \otimes p_j) \)

\( \Rightarrow \) when adding \( p_i \otimes p_j \), \( p_j \) can be removed from \( \mathcal{P} \)
- No need to compute all \( 2^n - 1 \) compositions of \( n \) permissions
- Computational complexity remains polynomial: \( O(n^3) \)

Some open issues

- Administration of authorizations
- Architectural solutions
- Content-based access restrictions
- Enforcement of policies by external storage providers
- Support for correctness guarantees on the query results
- Propagation of access restrictions and metadata together with the data (sticky policies, provenance)
- Support for multi-ownership policies