# Policy Specification and Enforcement in Emerging Scenarios

**Pierangela Samarati**
Dipartimento di Tecnologie dell'Informazione
Università degli Studi di Milano
pierangela.samarati@unimi.it

IEEE International Symposium on Policies for Distributed Systems and Networks
Pisa, Italy - June 6, 2011



## 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

$\Longrightarrow$ 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

# Sovereign joins – 1

- 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



# Sovereign joins – 2

- 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

# Access patterns – 1

Specify limitations on how information sources can be accessed (e.g., [Calì 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:
  Insurance$^{oi}$(holder,plan)
  Hospital$^{oioo}$(patient,YoB,disease,physician)
  Nat_registry$^{ioo}$(citizen,YoB,healthaid)
- Query:

```
SELECT patient
FROM   Insurance JOIN Hospital ON holder=patient
WHERE  plan = "annual"
```

- The query cannot be answered in the tradional way since input attribute YoB of Hospital is not bounded by a value
  - access Insurance to retrieve holders
  - holders as input to access Nat_Registry and retrieve years
  - years as input to access 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 $\Longrightarrow$ 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

S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, P. Samarati, "Authorization Enforcement in Distributed Query Evaluation," in *Journal of Computer Security*, 2011.

# 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

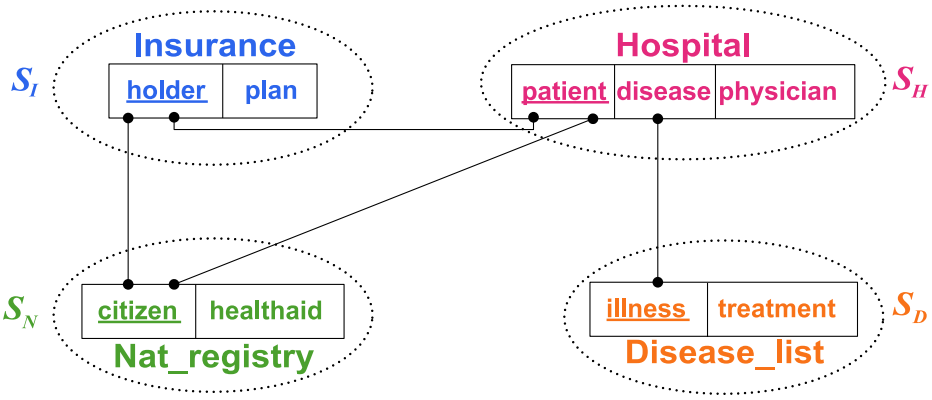
- support the collaboration among parties in distributed query execution on data subject to selective release
- define authorized views based on information content of a relation
- assign operations within the query to different parties in a way that is safe with respect to information that can be viewed by parties

# Data model

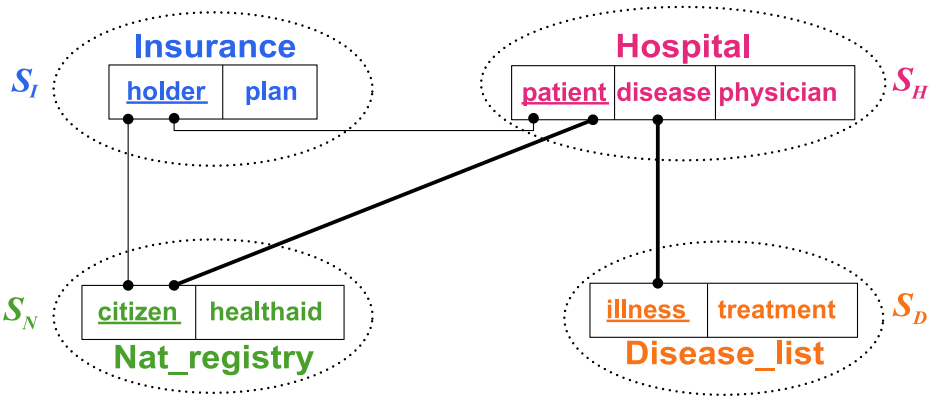
- $\mathscr{R}$: set of relations $R_i(A_{1_i},\ldots,A_{n_i})$, with $K_i \subseteq \{A_{1_i},\ldots,A_{n_i}\}$ primary key of $R_i$
- `I`: set of referential integrity constraints $\langle F_j, K_i\rangle$:
  - foreign key $F_j$ can assume only values in $R_i[K_i]$
- `J`: set of joins, $J = \langle A_l, A_r\rangle$ 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 *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
  - *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

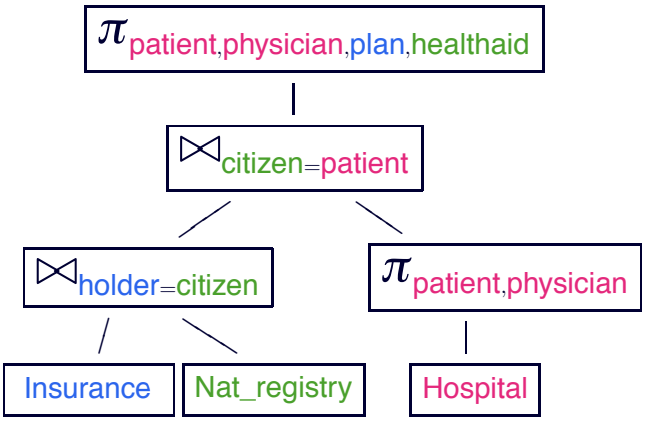



Example of join path for query execution:
{⟨citizen, patient⟩, ⟨disease, illness⟩}



# Example of query tree plan

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




Insurance(holder,plan) Hospital(patient,disease,physician)
Nat_registry(citizen,healthaid) Disease_list(illness,treatment)

# Permissions

Permission: [*Attributes, Join Path*]→*Subject*

- authorizes release to *Subject* of set *Attributes* of attributes resulting from the *Join Path*

Examples

- [(holder,plan),_] → $S_N$
- [(holder,plan,patient,physician),(⟨I.holder,H.patient⟩)] → $S_I$

| | |
|---|---|
| Insurance(holder,plan) | Hospital(patient,disease,physician) |
| Nat_registry(citizen,healthaid) | Disease_list(illness,treatment) |



# Relations in permissions – 1

*Join Path* may involve relations that do not have attributes in *Attributes*

- connectivity constraints: to build the association among attributes
  - [(holder,treatment),(⟨I.holder,H.patient⟩,⟨H.disease,D.illness⟩)] → $S_I$
- instance-based restrictions: to restrict the values of attributes to be released
  - [(holder,plan),(⟨I.holder,H.patient⟩)] → $S_I$

| | |
|---|---|
| Insurance(holder,plan) | Hospital(patient,disease,physician) |
| Nat_registry(citizen,healthaid) | Disease_list(illness,treatment) |

# Relations in permissions – 2

A join path in a permission:

- implies the release of fewer tuples
  does not imply the release of less information
  - [(holder,plan),_] $\rightarrow S_I$
    [(holder,plan),(⟨I.holder,H.patient⟩)] $\rightarrow S_I$
- ineffective when reachable via referential integrity constraints
  - [(patient,disease,physician),_] $\rightarrow S_I$
    [(patient,disease,physician),(⟨N.citizen,H.patient⟩,⟨H.disease,D.illness⟩)]$\rightarrow S_I$

| | |
|---|---|
| Insurance(holder,plan) | Hospital(patient,disease,physician) |
| Nat_registry(citizen,healthaid) | Disease_list(illness,treatment) |



# 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 $R$

| Operation | Profile $R^{\pi}$ | $R^{\bowtie}$ | $R^{\sigma}$ |
|---|---|---|---|
| $R := \pi_X(R_l)$ | $X$ | $R_l^{\bowtie}$ | $R_l^{\sigma}$ |
| $R := \sigma_X(R_l)$ | $R_l^{\pi}$ | $R_l^{\bowtie}$ | $R_l^{\sigma} \cup X$ |
| $R := R_l \bowtie_J R_r$ | $R_l^{\pi} \cup R_r^{\pi}$ | $R_l^{\bowtie} \cup R_r^{\bowtie} \cup J$ | $R_l^{\sigma} \cup R_r^{\sigma}$ |

# Relation profile – Example

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

Profile: $[R^{\pi}, R^{\bowtie}, R^{\sigma}]$
[(illness),(⟨D.illness,H.disease⟩),(treatment)]

| | |
|---|---|
| Insurance(holder,plan) | Hospital(patient,disease,physician) |
| Nat_registry(citizen,healthaid) | Disease_list(illness,treatment) |



# Authorized view

- Subject $S$ is authorized to view a relation $R$ iff ∃ [*Attributes, Join Path*]$\rightarrow S$: $R^{\pi} \cup R^{\sigma} \subseteq$ *Attributes* and $R^{\bowtie} =$ *Join Path*
- Examples
  - $S_D$ requires $R$ :
    SELECT illness
    FROM Disease_list JOIN Hospital ON illness=disease
    WHERE treatment='antihistamine'
  - Relation profile: [(illness),(⟨D.illness,H.disease⟩),(treatment)]
  - Authorization
    [(illness,treatment),(⟨D.illness,H.disease⟩)] $\rightarrow S_D$
    authorizes the query
  - Authorization [(illness,treatment),_] $\rightarrow S_D$
    does not authorize the query

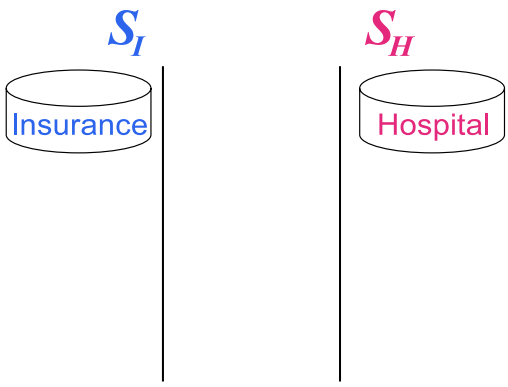
# Authorized data releases

- 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
  - $R_l \bowtie_{J_{lr}} R_r$ 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])

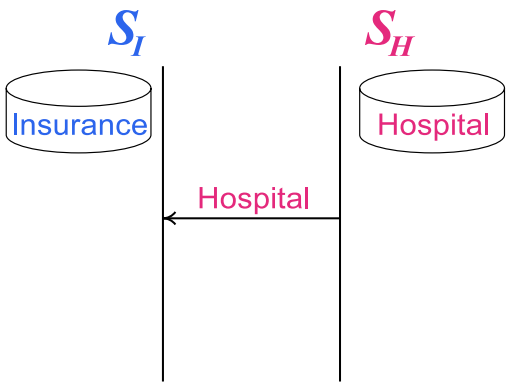
- $S_I$ requires $R$: $\pi_{\text{holder,plan,disease,physician}}$ (Insurance $\bowtie_{\text{holder=patient}}$ Hospital)
- Authorization: [(patient,disease,physician),_] $\rightarrow S_I$




| | |
|---|---|
| Insurance(holder,plan) | Hospital(patient,disease,physician) |
| Nat_registry(citizen,healthaid) | Disease_list(illness,treatment) |

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

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

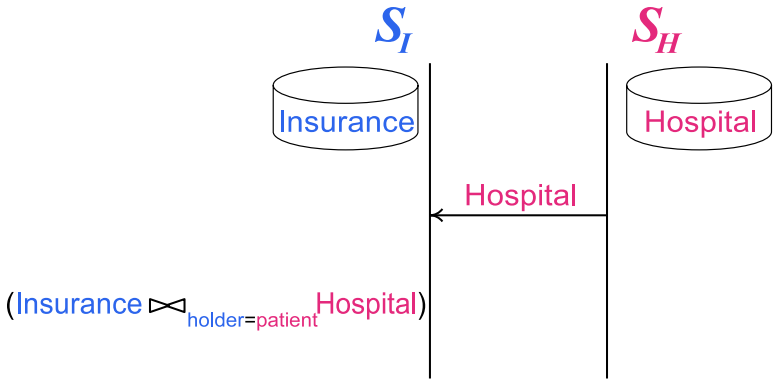



| | |
|---|---|
| Insurance(holder,plan) | Hospital(patient,disease,physician) |
| Nat_registry(citizen,healthaid) | Disease_list(illness,treatment) |



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

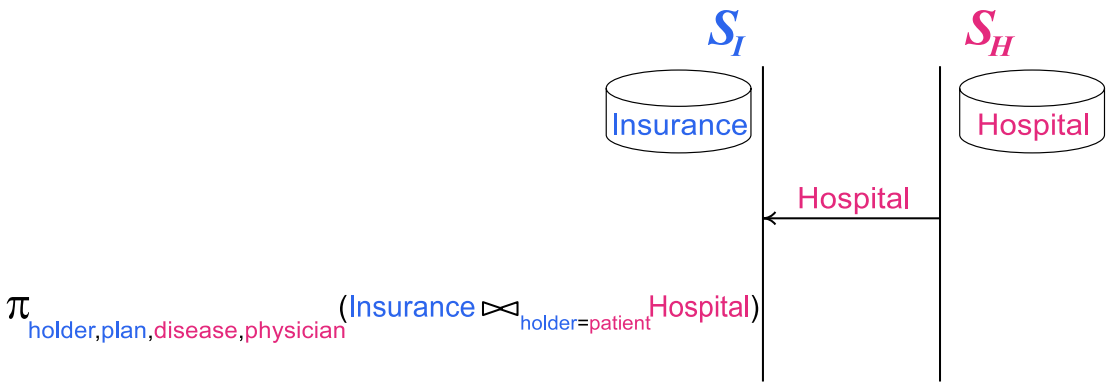
- $S_I$ requires $R$: $\pi_{\text{holder,plan,disease,physician}}$ (Insurance $\bowtie_{\text{holder=patient}}$ Hospital)
- Authorization: [(patient,disease,physician),_] $\rightarrow S_I$




| | |
|---|---|
| Insurance(holder,plan) | Hospital(patient,disease,physician) |
| Nat_registry(citizen,healthaid) | Disease_list(illness,treatment) |

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

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

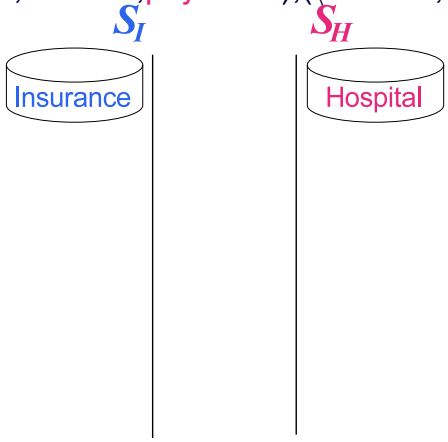



| | |
|---|---|
| Insurance(holder,plan) | Hospital(patient,disease,physician) |
| Nat_registry(citizen,healthaid) | Disease_list(illness,treatment) |



# Semi-joins – Example ([$S_I$,$S_H$])

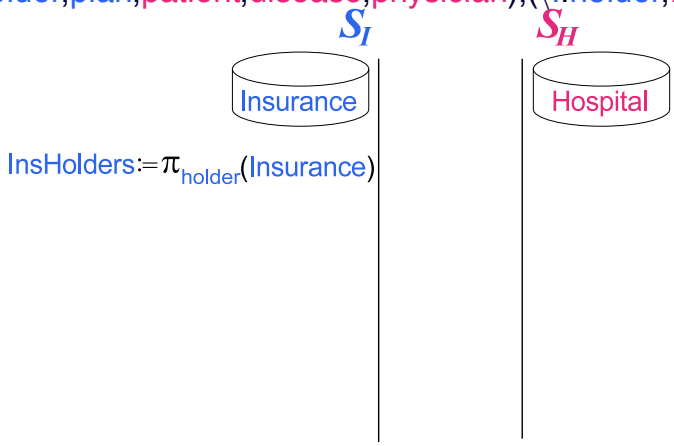
- $S_I$ requires $R$: $\pi_{\text{holder,plan,disease,physician}}$ (Insurance $\bowtie_{\text{holder=patient}}$ Hospital)
- Authorizations:
  [(holder),_]$\rightarrow S_H$
  [(holder,plan,patient,disease,physician),($\langle$I.holder,H.patient$\rangle$)] $\rightarrow S_I$




| | |
|---|---|
| Insurance(holder,plan) | Hospital(patient,disease,physician) |
| Nat_registry(citizen,healthaid) | Disease_list(illness,treatment) |

# Semi-joins – Example ([$S_I$,$S_H$])

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

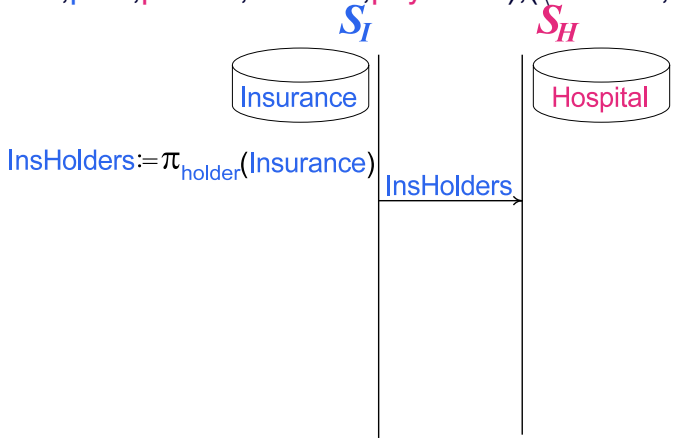



| | |
|---|---|
| Insurance(holder,plan) | Hospital(patient,disease,physician) |
| Nat_registry(citizen,healthaid) | Disease_list(illness,treatment) |



# Semi-joins – Example ([$S_I$,$S_H$])

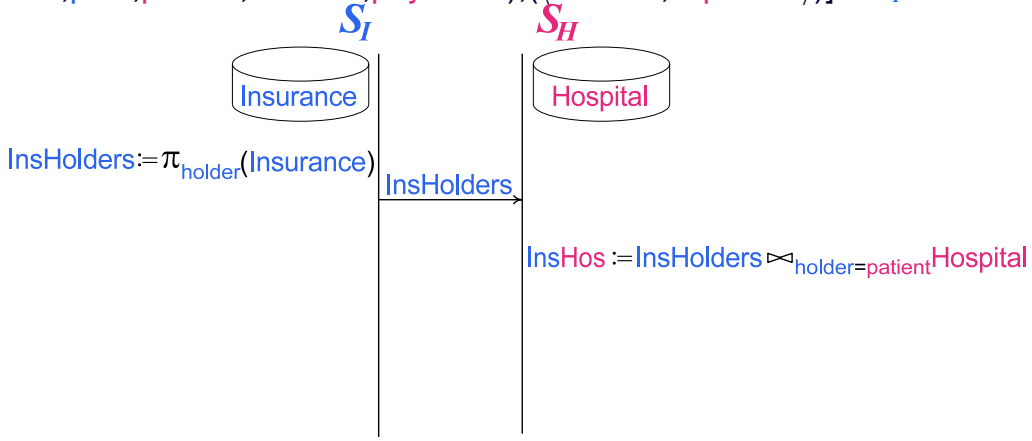
- $S_I$ requires $R$: $\pi_{\text{holder,plan,disease,physician}}$ (Insurance $\bowtie_{\text{holder=patient}}$ Hospital)
- Authorizations:
  [(holder),_]$\rightarrow S_H$
  [(holder,plan,patient,disease,physician),($\langle$I.holder,H.patient$\rangle$)] $\rightarrow S_I$




| | |
|---|---|
| Insurance(holder,plan) | Hospital(patient,disease,physician) |
| Nat_registry(citizen,healthaid) | Disease_list(illness,treatment) |

# Semi-joins – Example ([$S_I$,$S_H$])

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

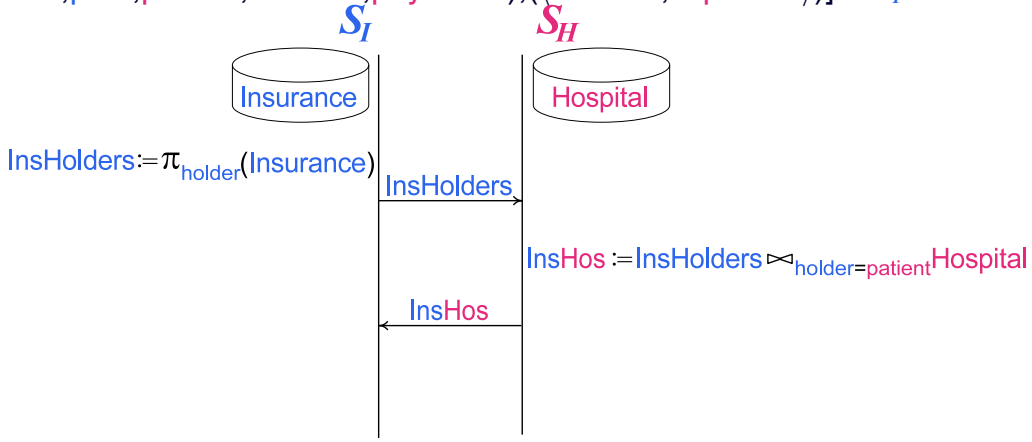



| | |
|---|---|
| Insurance(holder,plan) | Hospital(patient,disease,physician) |
| Nat_registry(citizen,healthaid) | Disease_list(illness,treatment) |



# Semi-joins – Example ([$S_I$,$S_H$])

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

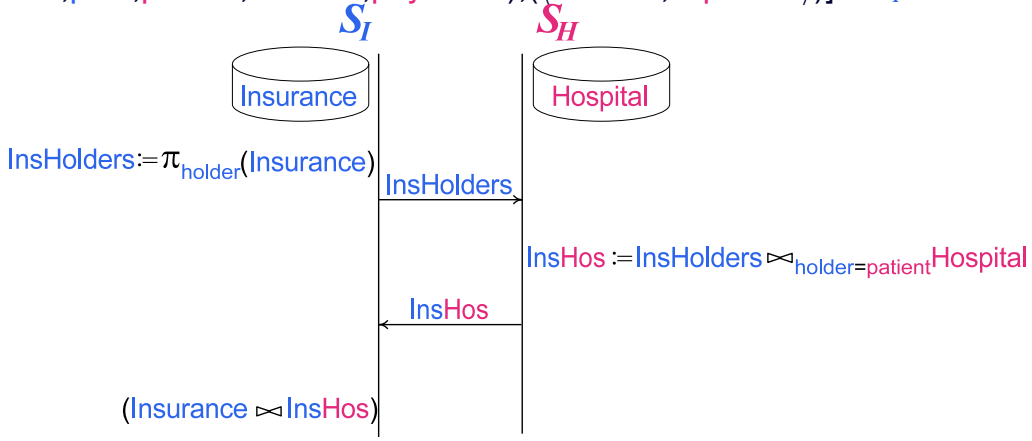



| | |
|---|---|
| Insurance(holder,plan) | Hospital(patient,disease,physician) |
| Nat_registry(citizen,healthaid) | Disease_list(illness,treatment) |

©Pierangela Samarati 30/55

# Semi-joins – Example ([$S_I$,$S_H$])

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

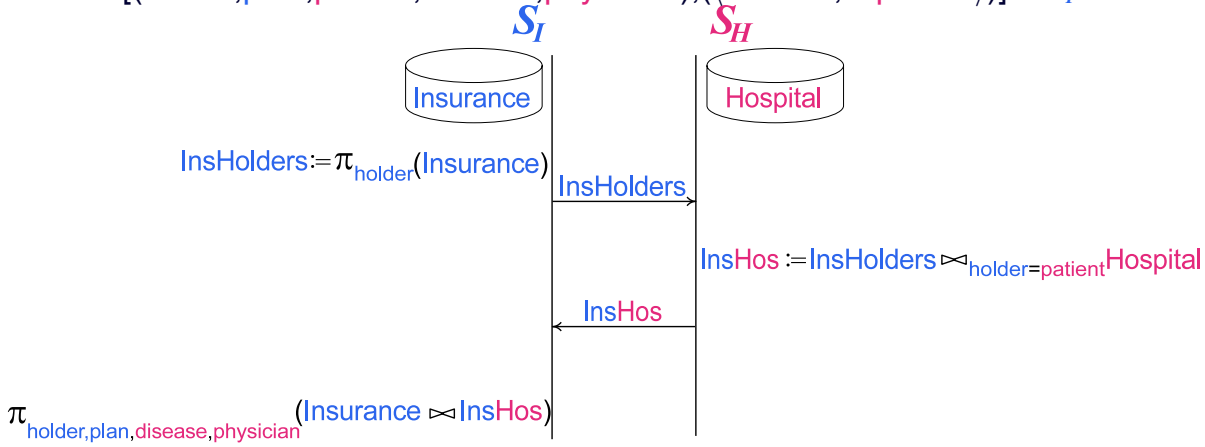



| | |
|---|---|
| Insurance(holder,plan) | Hospital(patient,disease,physician) |
| Nat_registry(citizen,healthaid) | Disease_list(illness,treatment) |



# Semi-joins – Example ([$S_I$,$S_H$])

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




| | |
|---|---|
| Insurance(holder,plan) | Hospital(patient,disease,physician) |
| Nat_registry(citizen,healthaid) | Disease_list(illness,treatment) |

# 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 $\{\bowtie\}$: $\lambda_T(n) = [master,slave]$
    *master* $\in \{S_l, S_r\}$, *slave* $\in \{S_l, S_r, \text{NULL}\}$, and *master*$\neq$*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



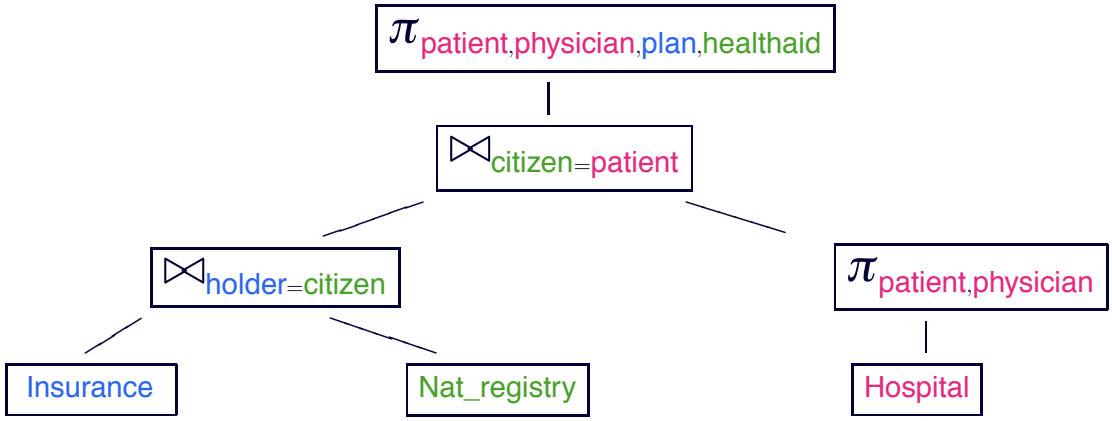
# Example of executor assignment

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

| | |
|---|---|
| Insurance(holder,plan) | Hospital(patient,disease,physician) |
| Nat_registry(citizen,healthaid) | Disease_list(illness,treatment) |

# Example of executor assignment

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

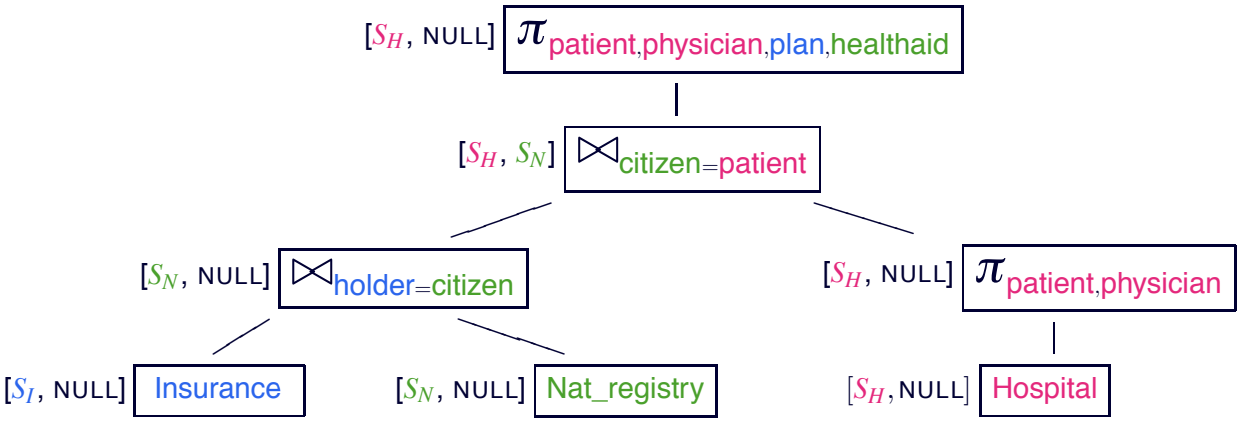



Insurance(holder,plan) Hospital(patient,disease,physician)
Nat_registry(citizen,healthaid) Disease_list(illness,treatment)



# Example of executor assignment

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




Insurance(holder,plan) Hospital(patient,disease,physician)
Nat_registry(citizen,healthaid) Disease_list(illness,treatment)

# 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 $\exists$ [*Attributes, Join Path*]$\rightarrow S$: $R^{\pi} \cup R^{\sigma} \subseteq$ *Attributes* and $R^{\bowtie} =$ *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:
  [(holder,plan),_] $\rightarrow S_I$
  [(patient, disease),_] $\rightarrow S_I$
- $S_I$ requires $R$:
  SELECT holder, disease
  FROM Insurance JOIN Hospital ON holder=patient
- Profile: [(holder,disease),($\langle$I.holder,H.patient$\rangle$),_]

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

| | |
|---|---|
| Insurance(holder,plan) | Hospital(patient,disease,physician) |
| Nat_registry(citizen,healthaid) | Disease_list(illness,treatment) |



# Authorization Composition

S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, P. Samarati, "Assessing Query Privileges via Safe and Efficient Permission Composition," in *Proc. of the 15th ACM Conference Conference on Computer and Communications Security (CCS 2008),* Alexandria, Virginia, USA, October 27-31, 2008.

# 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 [*Attr, Join Path*]$\rightarrow$*Subject* can be viewed as [*Attr*,*Rel*], where *Rel* are the relations involved in *Join Path*
- Relation profile $[R^{\pi}, R^{\bowtie}, R^{\sigma}]$ can be viewed as [*Attr*,*Rel*]:
  - *Attr* $= R^{\pi} \cup R^{\sigma}$
  - *Rel* are the relations involved in join path $R^{\bowtie}$

# Schema graph

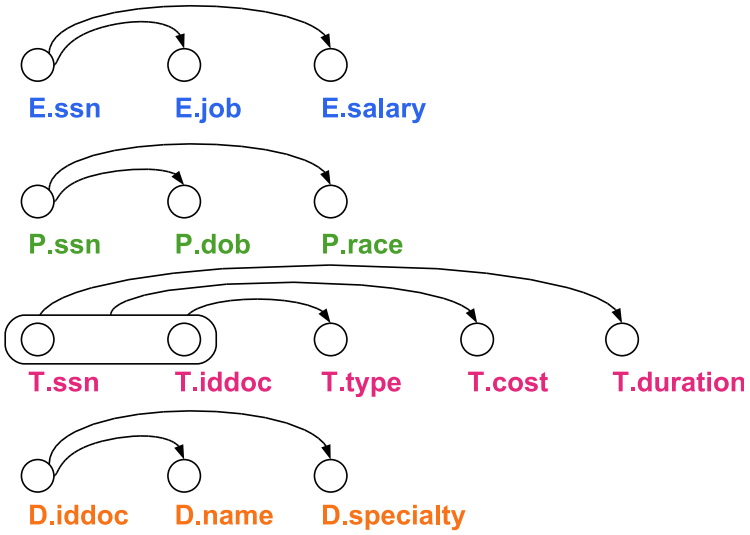
Schema graph of a set $\mathscr{R}$ of relations is a mixed graph $G(\mathtt{N}, \mathtt{E})$

- $\mathtt{N} = \{R_i.* \mid R_i \in \mathscr{R}\}$ corresponds to the set of attributes
- $\mathtt{E} = \mathtt{J} \cup \mathtt{I} \cup \{(R_i.K, R_i.A) \mid R_i \in \mathscr{R} \wedge A \notin K\}$ corresponds to:
  - natural joins (non-oriented)
  - referential integrity constraints (oriented)
  - functional dependencies (oriented)



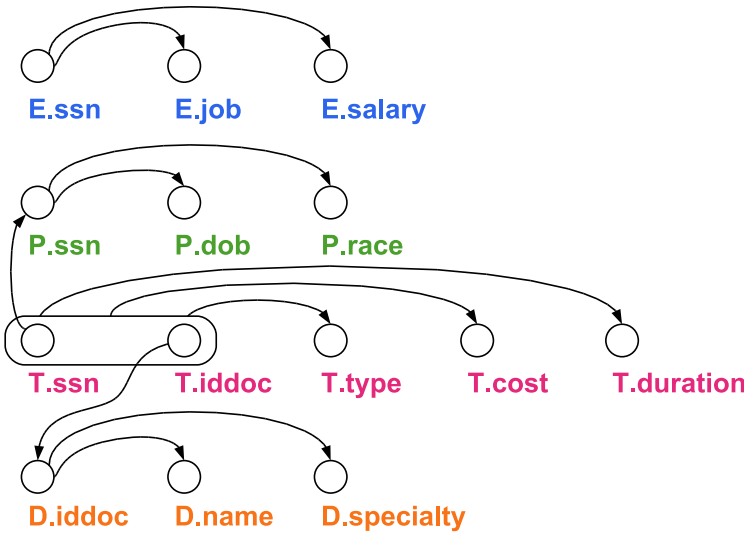
# Schema graph – Example

| $\mathscr{R}$ | | | |
|---|---|---|---|
| | Employee(ssn,job,salary)<br>Patient(ssn,dob,race)<br>Treatment(ssn,iddoc,type,cost,duration)<br>Doctor(iddoc,name,specialty) | I | |
| | | J | |



# Schema graph – Example

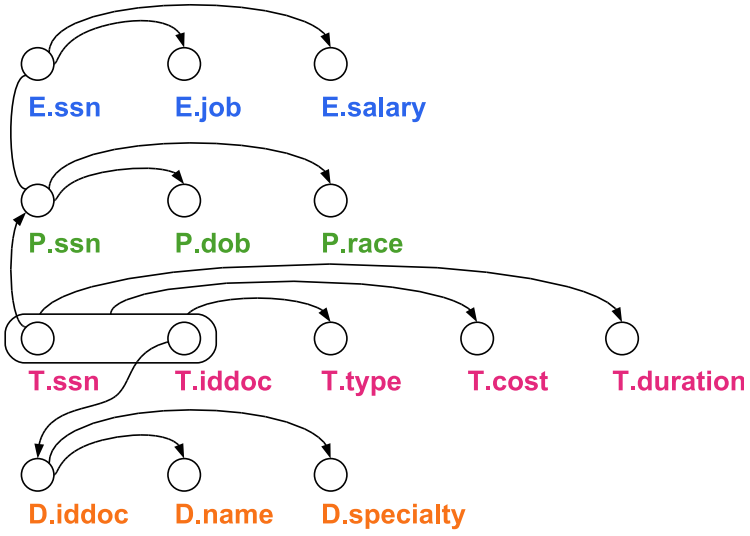
| $\mathscr{R}$ | Employee(ssn,job,salary)<br>Patient(ssn,dob,race)<br>Treatment(ssn,iddoc,type,cost,duration)<br>Doctor(iddoc,name,specialty) | I | ⟨Treatment.ssn,Patient.ssn⟩<br>⟨Treatment.iddoc,Doctor.iddoc⟩ |
|---|---|---|---|
| | | J | |






# Schema graph – Example

| $\mathscr{R}$ | Employee(ssn,job,salary)<br>Patient(ssn,dob,race)<br>Treatment(ssn,iddoc,type,cost,duration)<br>Doctor(iddoc,name,specialty) | I | ⟨Treatment.ssn,Patient.ssn⟩<br>⟨Treatment.iddoc,Doctor.iddoc⟩ |
|---|---|---|---|
| | | J | ⟨Employee.ssn,Patient.ssn⟩ |



# Views

Permissions and queries $[Attr, Rel]$ are views over set of relations $\mathscr{R}$

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

Example

- *Rel* = {Treatment}
- $Rel^*$ = {Treatment,Patient,Doctor}

| | |
|---|---|
| Employee(ssn,job,salary) | Treatment(ssn,iddoc,type,cost,duration) |
| Patient(ssn,dob,race) | Doctor(iddoc,name,specialty) |

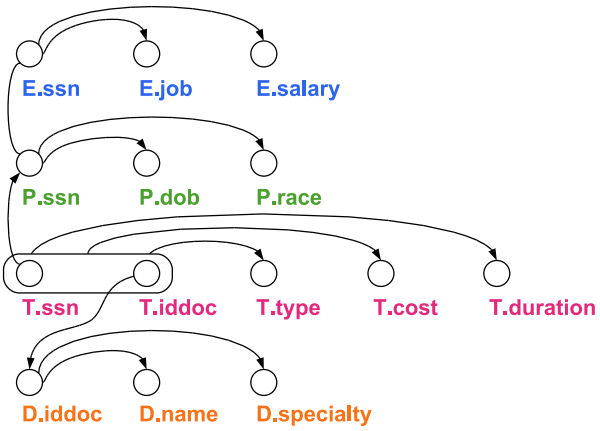


# View graph

$G_V(\mathtt{N}, \mathtt{E}, \lambda_V)$ 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

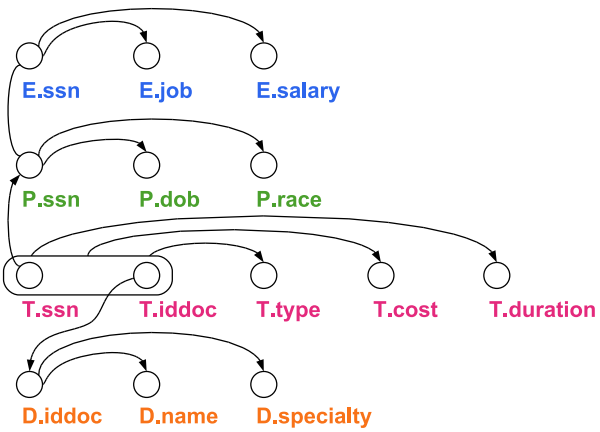
# View graph – 1 (Examples)





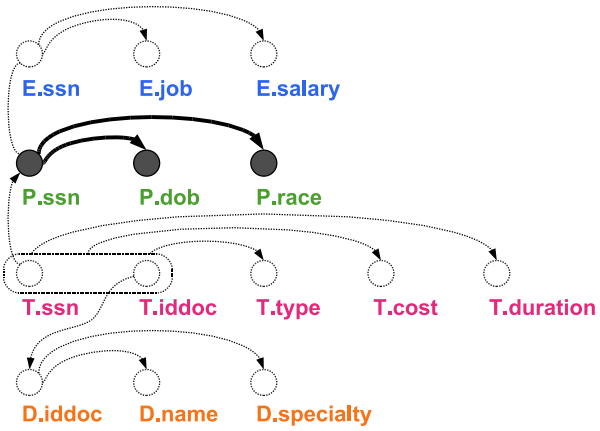

# View graph – 1 (Examples)

[(ssn,dob,race),(Patient)]→Alice



# View graph – 1 (Examples)

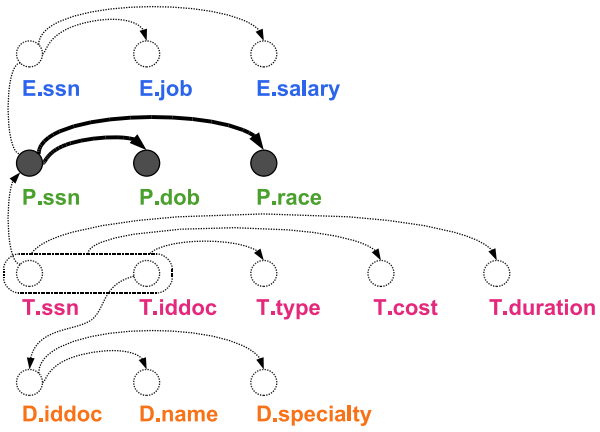
[(ssn,dob,race),(Patient)]→Alice

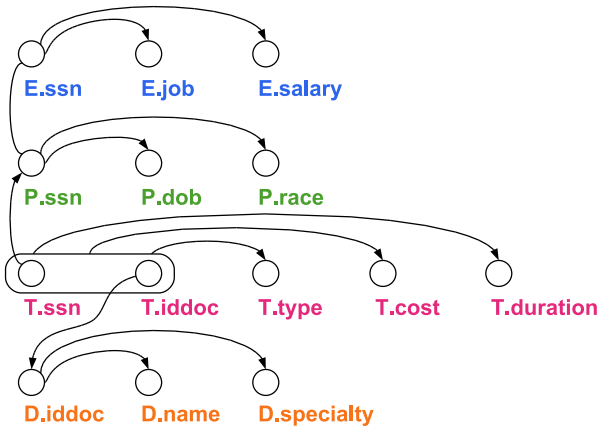





# View graph – 1 (Examples)

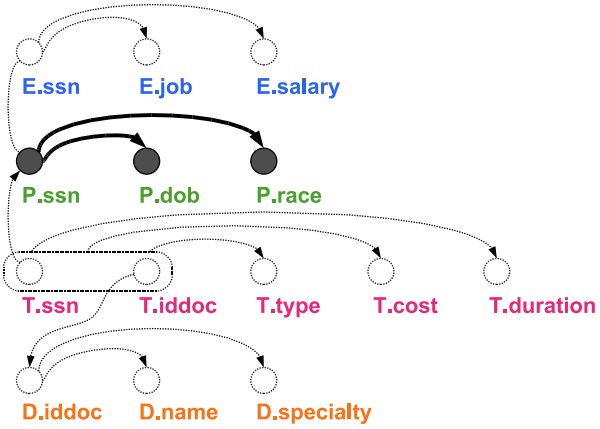
[(ssn,dob,race),(Patient)]→Alice



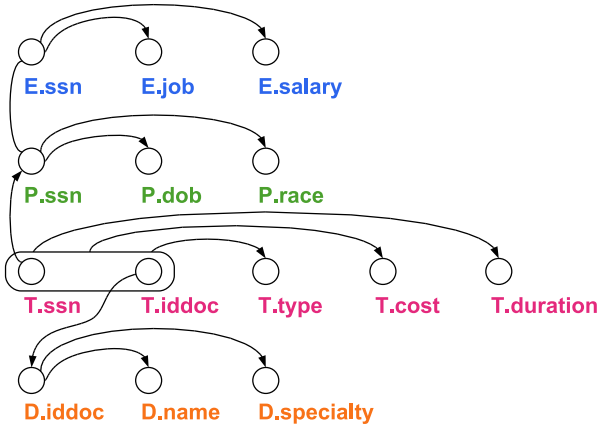



# View graph – 1 (Examples)

[(ssn,dob,race),(Patient)]→Alice



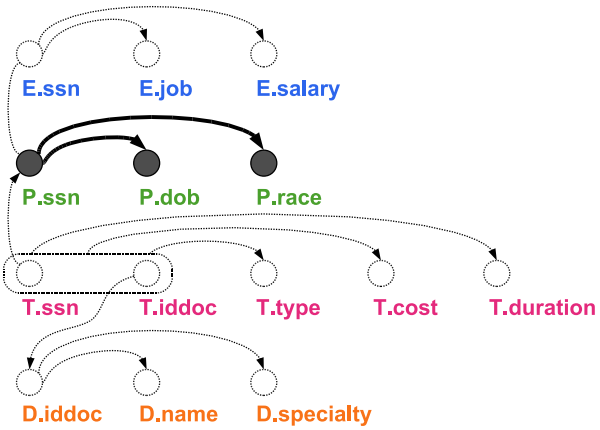

[(ssn,type,cost,duration),(Treatment)]→Alice



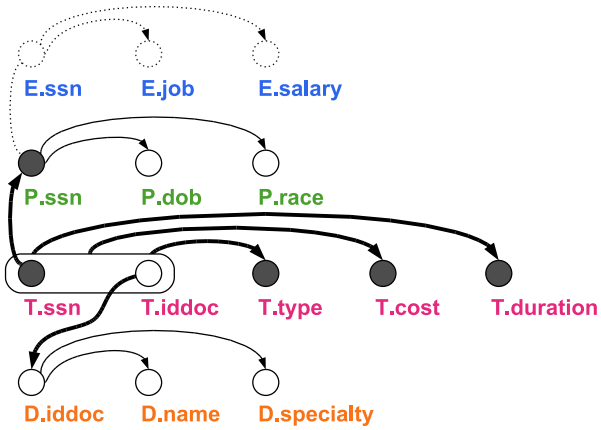



# View graph – 1 (Examples)

[(ssn,dob,race),(Patient)]→Alice




[(ssn,type,cost,duration),(Treatment)]→Alice



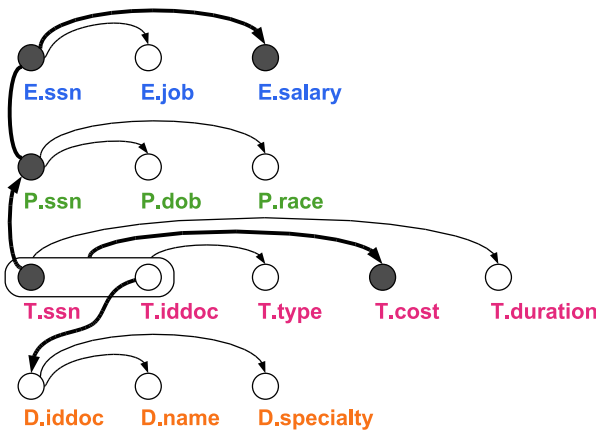
# View graph – 2 (Examples)

```
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
```



# View graph – 2 (Examples)

```
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
```




©Pierangela Samarati 44/55

# Authorizing permissions

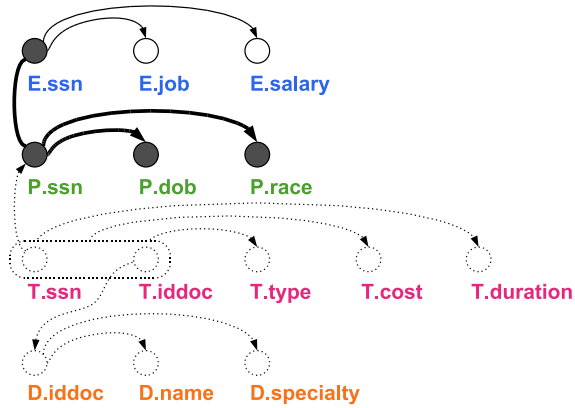
Permission *p* authorizes query *q* iff the direct or indirect information released by *q* is a subset of the information authorized by *p*

- $p=[Attr_p,Rel_p]$ 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 $G_p$



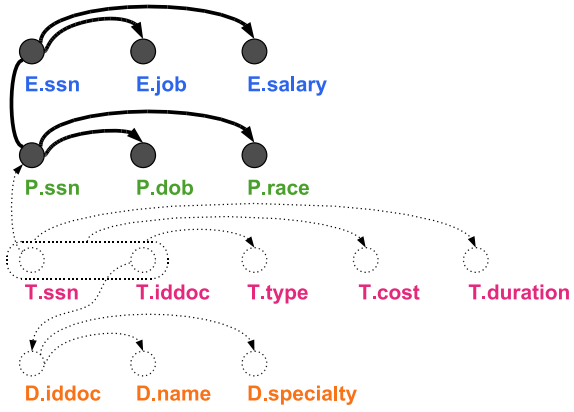
# Applicable, authorizing permission – Example

Query




Authorized

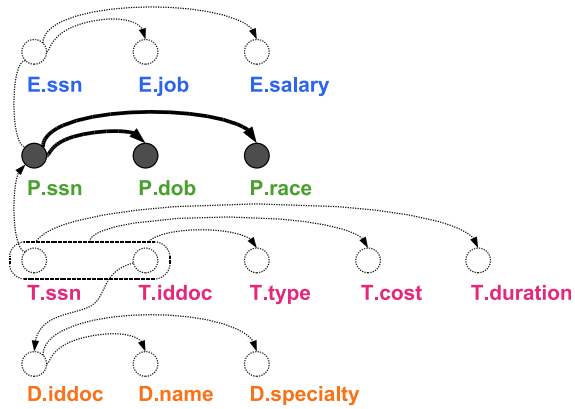
Permission



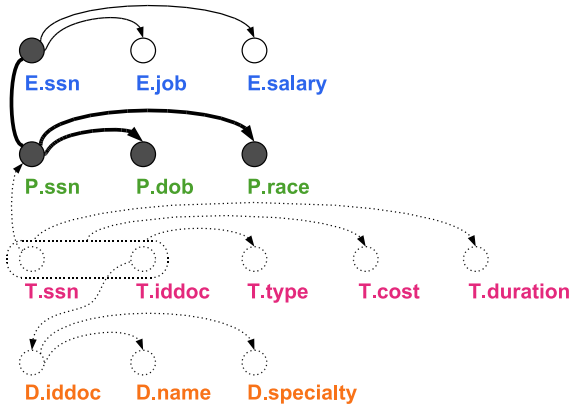

Applicable

# Non applicable permission – Example

## Query




## Permission

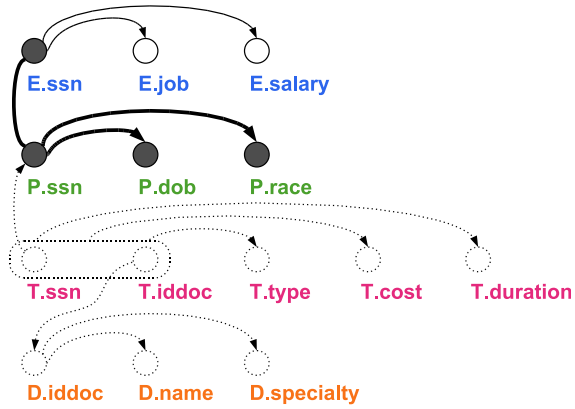



Non Applicable



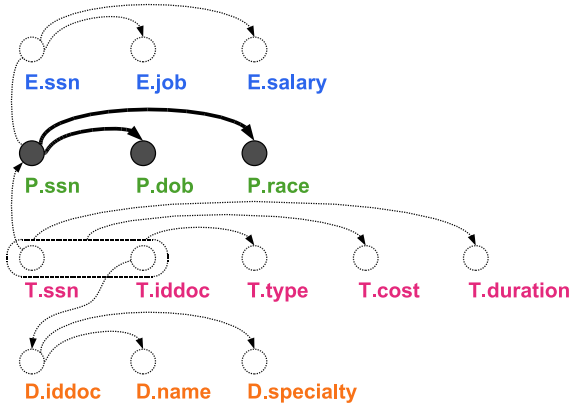
# Applicable, non authorizing permission – Example

## Query




Non Authorized

## Permission




Applicable

# Composition of permissions

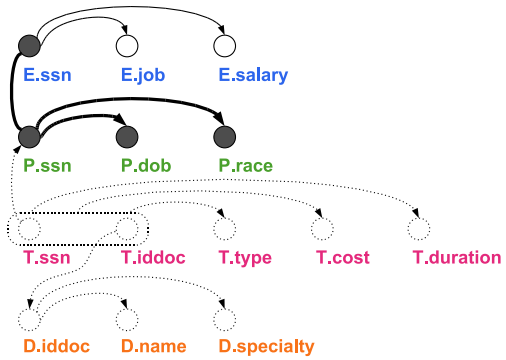
A query can be executed if the subject has permissions to view the information content carried by the query

- 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 $\Longrightarrow$ composition of permissions
- Need to be careful not to enact indirect disclosure



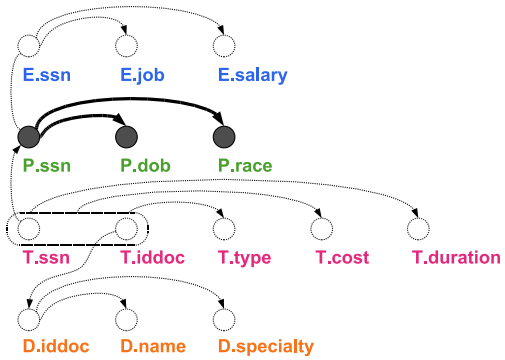
# Composition of permissions – 1 (Examples)

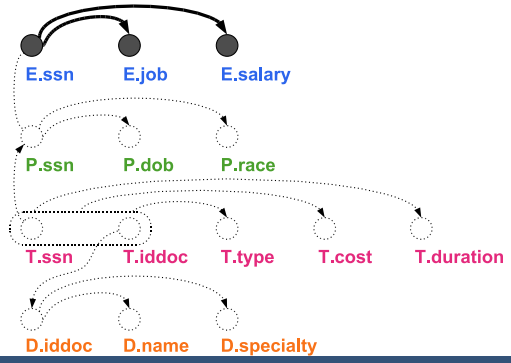
## Query




Authorized

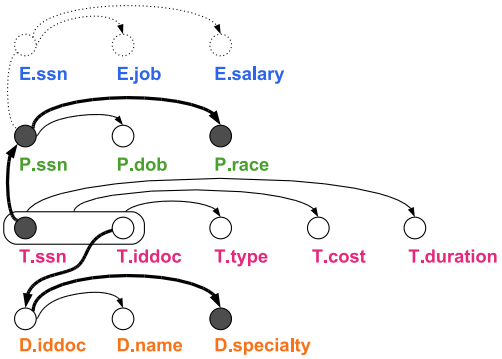
## Permissions



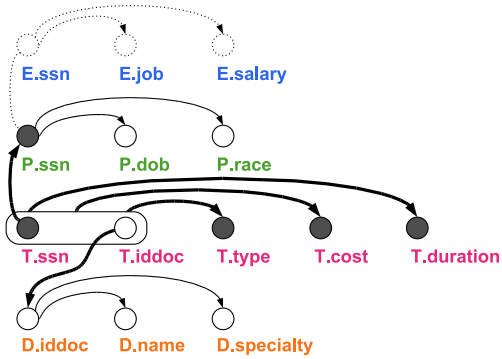
# Composition of permissions – 2 (Examples)

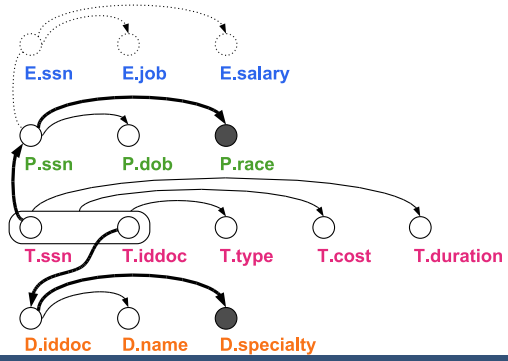
Query




Non Authorized

Permissions





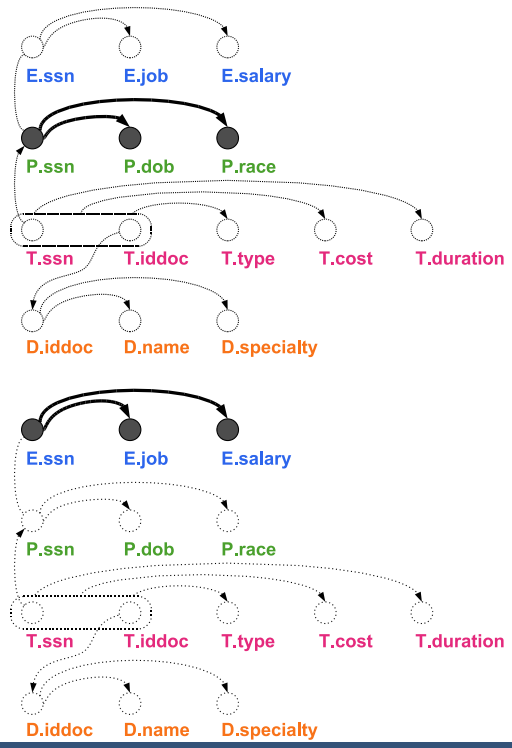

# Safe composition

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

- $p_i \rightarrow p_j$: $p_j$=[$Attr_j$,$Rel_j$] depends on $p_i$=[$Attr_i$,$Rel_i$] iff ∃ a path from nodes corresponding to attributes in $Attr_i \cap Attr_j$ 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 $p_i \otimes p_j$=[$Attr_i \cup Attr_j$,$Rel_i \cup Rel_j$]
  - graph coloring of composed permission derived from colors of operands

# Safe composition – Example

## Permissions

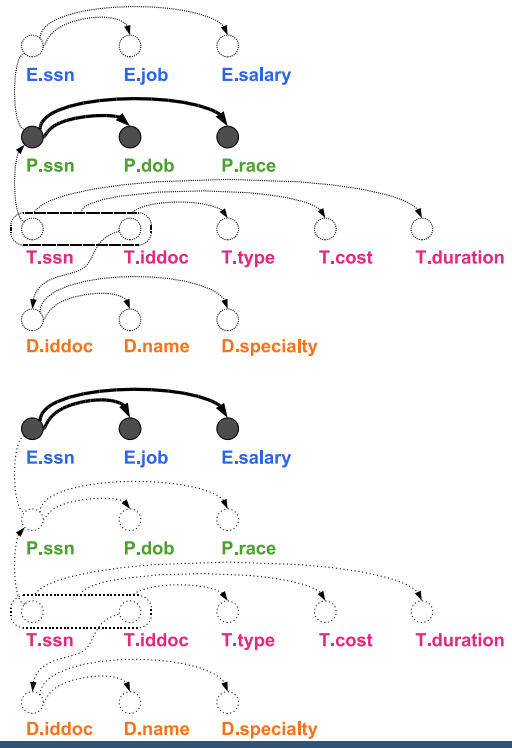



## Composed permission

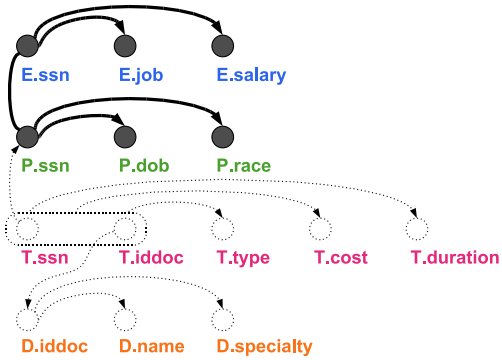


# Safe composition – Example

## Permissions



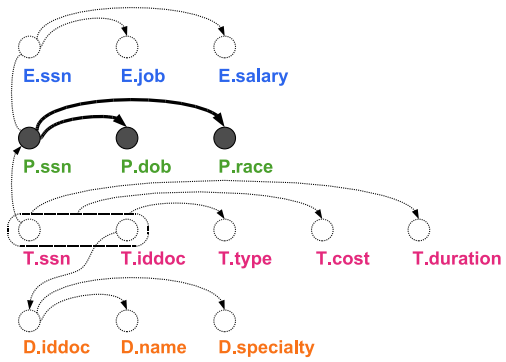

## Composed permission

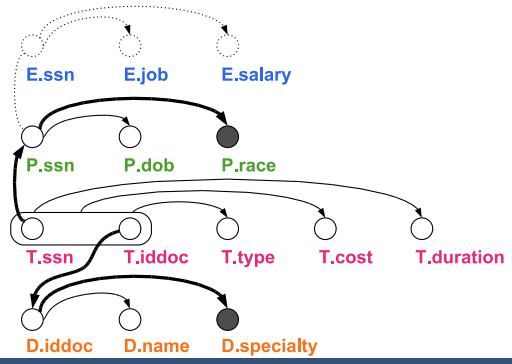



Safe

# Safe composition – Example

## Permissions



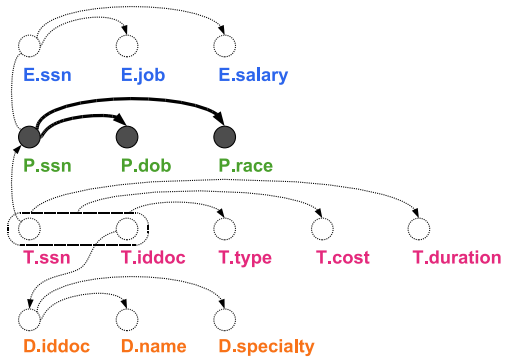



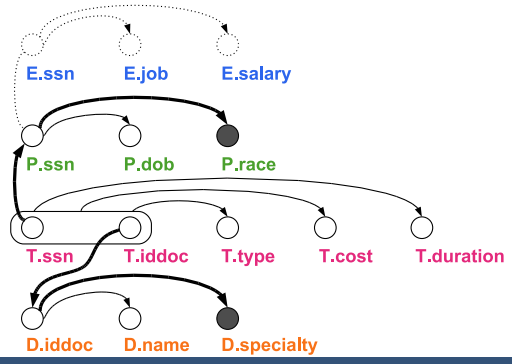

## Composed permission



# Safe composition – Example

## Permissions




## Composed permission

Non Composable

# Access control enforcement

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

Efficiency guaranteed by exploiting permission implication

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

  $\Longrightarrow$ when adding $p_i \otimes p_j$, $p_j$ can be removed from $\mathtt{P}$
- 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