Controllo delle Query Distribuite

Parte III

Indice

Capitolo 1

Introduzione

Torniamo a preoccuparci del problema di confidenzialità, nel contesto di computazione di query distribuite; l'assunzione è che non tutti siano autorizzati a vedere tutti i dati, ma ci sono dei vincoli di confidenzialità che devono essere rispettati.
Lo scenario di riferimento è quello in cui ci sono più sorgenti informative, dove da un lato ho l'esigenza di condivisione dei dati (per rispondere alle query), mentre dall'altro ho esigenza di confidenzialità perché non è detto che chiunque possa leggere i dati.

1.1 Join sovrani

Questo approccio sfrutta la presenza di un hardware fidato (nel senso che nessuno può vedere cosa fa), che può ricevere i dati per eseguire le computazioni.
Lo scenario è:

- si hanno due *data owner* che non si fidano l'uno dell'altro
- c'è una terza parte, che ha a disposizione dell'hardware fidato, che esegue la computazione

L'idea è che le parti criptano i dati e li mandano all'hardware, che si occupa di:

- decriptare i dati
- eseguire la computazione
- recriptare i dati e darli al client

Un osservatore potrebbe inferire sulla base del risultato qualcosa, come ad esempio sulle dimensioni del risultato o sul tempo richiesto ad eseguire la computazione.
$\Rightarrow$ l'output deve avere più o meno sempre la stessa dimensione e tempo di computazione, per cercare di ridurre l'inferenza

1.2 Access patterns

Cercano di specificare come le fonti informative devono essere accedute.
Definiamo un *access pattern* con un esempio:

- abbiamo 3 relazioni, ciascuna con un access pattern, ovvero dei vincoli di accesso
- si ha una lettera per ciasuno attributo delle relazione
  - *o* per output
  - *i* per input

  Insurance^{oi}(holder,plan)
  Hospital^{oioo}(patient,YoB,disease,physician)
  Nat_registry^{ioo}(citizen,YoB,healthaid)

  per accedere all'attributo "o" mi deve dare l'attributo "i"; si pongono dei vincoli, l'accesso non è libero

Questa tecnica presenta alcuna svantaggi:

- limitata espressione delle limitazioni
- tipicamente ci sono due entità, non un vero scenario distribuito
- può essere difficile da usare nella pratica

1.3 Autorizzazioni basate su viste

La peculiarità di questo approccio è che le restrizioni di accesso dipendono dal contenuto del dato.

- 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

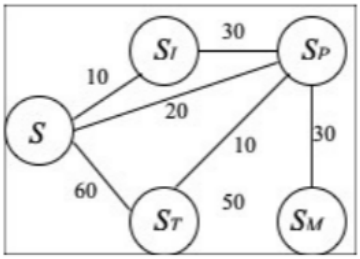
Verifico se una query può essere eseguita sulla base delle autorizzazioni che ho definito; il client scrive la sua query, e il server cerca di rielaborarla sulla base delle viste che sono state definite.
Nel caso in cui una query non possa essere eseguita, ci sono due scenari possibili:

- *truman:* ti restituisco un risultato parziale, che corrisponde non alla query che mi hai chiesto ma alla vista che è stata definita (facendotelo passare come completo)
- *non-truman:* non ti restituisco nulla e ti dico che non sei autorizzato ad accedere al risultato

1.4 Coalition networks

Ci sono diversi *providers* che si conoscono e che formano delle *coalizioni*; sono disposti a condividere le proprie informazioni per un obiettivo comune.
Ciascun provider ha:

- una o più relazioni
- uno o più server



I server formano una rete, possono comunicare tra di loro; una computazione vuole essere effettuata **minimizzando il costo** (ciascun canale a un costo associato) e **rispettando le restrizioni** sul flusso di informazioni (chi può vedere che cosa).
⇒ Si definisce un ***safe query plan***, ovvero un modo per soddsifare la query in modo sicuro e che minimizzi i costi:

- per le operazioni unarie non ci sono problemi, dato che non richiedono alcun trasferminento di dati
- per le operazioni di join, viene richiesto la cooperazione tra i due server:
  - uno funge da *master*, ha il compito di eseguire il join
  - uno funge da *slave*, aiuta il master

Dato che l'operazione più costosa e che implica un maggiore flusso di informazioni è il join, il *focus* è su come eseguirla in modo da rispettare le autorizzazioni; l'idea che sta alla base di questo modello è di supportare diverse operazioni di join in diversi modi, dove ognuno di questi implica flussi di informazioni diversi.

1.4.1 Broker join

Ci sono due relazioni su due server, su cui dobbiamo eseguire il join. Tipicamente, uno dei due server funge da *master* e l'altro da *slave*; se però sono state definite delle restrizioni che impediscono di accedere all'altra relazione, questa architettura non può esssere utilizzata.
Con il broker-join si usa (se esiste) un **terzo server che sia autorizzato ad accedere alle relazioni ed eseguire il join**; se ne esiste più di uno, seleziono quello con il costo minore.

1.4.2 Peer-join

Al contrario dello scenario precedente, almeno uno dei due server è autorizzato a leggere la relazione dell'altro; il join viene dunque eseguito da uno dei due server (viene scelto quello con il costo minore nel caso in cui tutti e due possano farlo).

1.4.3 Semi-join

Entrambi i server entrano in gioco per l'esecuzione dell'operazione di join:

- S_x fa da master, fa una proiezione della sua relazione sull'attributo di join, e la manda a S_y
- S_y unisce la relazione che ha ricevuto e fa il join con la sua relazione; manda il risultato a S_x
- S_x completa il risultato aggiungendo gli altri attributi che mancano (dato che inizialmente ha mandato solo l'attributo di join)

1.4.4 Split-join

- Supponiamo di avere un server S_x, con una relazione $r_x = r_{x1} \cup r_{x2}$
- Supponiamo, allo stesso modo, di avere un server S_y, con una relazione $r_y = r_{y1} \cup r_{y2}$
- Supponiamo che S_y possa accedere solo a una parte di r_x, ad esempio solo a r_{x1}
- Supponiamo, allo stesso modo, che S_x possa accedere solo a una parte di r_y, ad esempio solo a r_{y1}

$\Rightarrow$ Per rispettare i vincoli di confidenzialità il join viene eseguito in questo modo:

- Il server S_x fa il join tra r_x e r_{y1}, ovvero ciò a cui può accedere di r_y
- Il server S_y fa il join tra r_y e r_{x1}
- A questo punto manca il join tra r_{x2} e r_{y2}; nessuno dei due server può leggere questa parte della relazione, per cui viene coinvolta una terza parte che ha l'autorizzazione per farlo

L'operazione di join viene *splittata* in tre parti:

- un peer-join svolto da S_x
- un peer-join svolto da S_y
- un broker join

1.5 Preferenze in ottimizzazione delle query

L'aspetto che caratterizza questa classe di soluzioni è che fino ad adesso il *focus* è stato lato server; ora si cambia e diventa il client, che vuole eseguire una query, che si preoccupa di come la query viene eseguita (magari è sensibile e voglio decidere io come viene svolta).
Viene modificato il linguaggio che l'utente usa per esprimere la computazione, in modo che possa esprimere anche le restrizioni; ad esempio, vediamo due tipi di restrizioni:

- `REQUIRING condition HOLDS OVER` $\langle operation, parameters, master \rangle$
  - è un'autorizzazione **forte**, che deve per forza essere soddisfatta; la restrizione di accesso è `condition` applicata alle operazioni rappresentate dai nodi della terna, ovvero quelle che riguardano:
    - l'operazione *operation*
    - sugli attributi *parameters*
    - eseguita da *master*
- `PREFERRING condition HOLDS OVER` $\langle operation, parameters, master \rangle$
  - è un'autorizzazione **debole**, è una preferenza dell'utente; la restrizione applicata segue il ragionamento precedente

In questo modo gli utenti possono definire delle restrizioni su come vengono eseguite le computazioni.

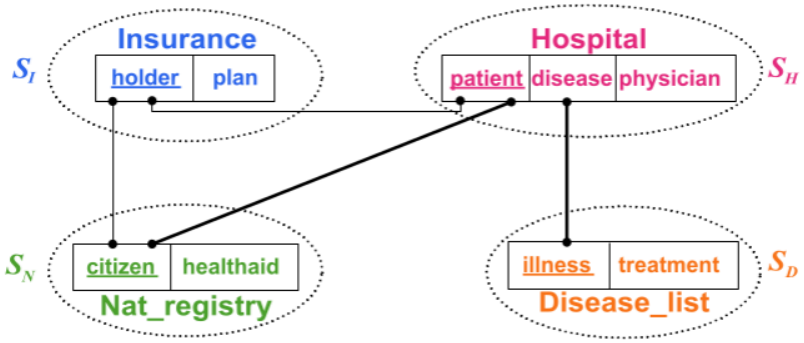
Capitolo 2

Valutazione di query distribuite sotto requisiti di protezione

In questo modello si valuta anche il **bagaglio informativo addizionale** per valutare se una informazione può essere trasferita da una parte ad un'altra (ad esempio, una relazione può essere il risultato di una computazione, quindi mi sta dando informazioni aggiuntive non esplicite).
In aggiunta, si usa un modello di autorizzazione che restringe non solo l'informazione che puoi vedere, ma anche **il modo in cui questa informazione può essere computata**.
⇒ Questi due aspetti sono un qualcosa che i modelli visti in precedenza non considerano

Scenario di riferimento

Ci sono diverse sorgenti informative; gli archi mostrano come le informazioni possono essere combinate tra di loro.



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

2.1 Permessi

I permessi vengono espressi come una coppia $[Attributes, JoinPath] \rightarrow Subject$
$\rightarrow$ *il soggetto è autorizzato ad accedere a tutti gli attributi listati, applicando il join path*

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

I join path aumentano il potere espressivo, e possono:

- Rappresentare **vincoli di connettività:** stabilisco come sono collegate relazioni diverse

  [(holder,treatment),(⟨I.holder,H.patient⟩,⟨H.disease,D.illness⟩)] $\rightarrow S_I$

  mi dicono come collegare le relazioni affinché questi attributi siano accessibili a un particolare soggetto

- Esprimere **restrizioni sulla quantità di informazioni** a cui un soggetto può accedere

$$[(\text{holder},\text{plan}),(\langle \text{I.holder},\text{H.patient}\rangle)] \rightarrow S_I$$

posso accedere solo alle tuple blu che si uniscono in join alla relazione rosa

Bisogna fare attenzione al fatto che:

- un rilascio di meno tuple (dovuto ad un join path restrittivo) non implica per forza un rilascio di meno informazion

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

- può essere inutile se coinvolge i vincoli di integrità referenziale

$$[(\text{patient},\text{disease},\text{physician}),\_] \rightarrow S_I$$
$$[(\text{patient},\text{disease},\text{physician}),(\langle \text{N.citizen},\text{H.patient}\rangle,\langle \text{H.disease},\text{D.illness}\rangle)] \rightarrow S_I$$

2.2 Profilo della relazione

È un concetto che cerca di catturare il concetto di informazione aggiuntiva non esplicita; il **profilo di una relazione** è una tripla $[R^\pi, R^\bowtie, R^\sigma]$, dove:

- R^π è la relazione esplicita
- $R^\bowtie$ è il join path eseguito per ottenere la relazione R
  - *come ho ottenuto questa relazione? è stata ottenuta da un join?*
- R^σ è il set di attributi che sono stati coinvolti in operazioni di selezioni applicate per ottenere la relazione R
  - *R deriva da qualche condizione applicata su attributi che non sono esplicitamente presenti nella relazione?*

Esempio

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

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

2.3 Vista autorizzata

Un soggetto S è autorizzato ad accedere ad una vista R sse:
$\exists[Attributes, JoinPath] \rightarrow S | R^{\pi} \cup R^{\sigma} \subseteq Attributes \wedge R^{\bowtie} = JoinPath$
$\Rightarrow$ *Un soggeto è autorizzato ad accedere ad una relazione quando:*

- *tutti gli attributi espliciti e quelli che usati per definere le condizioni che hanno ristretto le tuple che fanno parte della relazione, sono attributi a cui il soggetto può accedere*
- *il join nella componente R deve essere uguale al join path a cui il soggetto è autorizzato; devono essere ottenuti in quel modo, altrimenti sta accedendo ad informazioni a cui non ha diritto ad accedere*

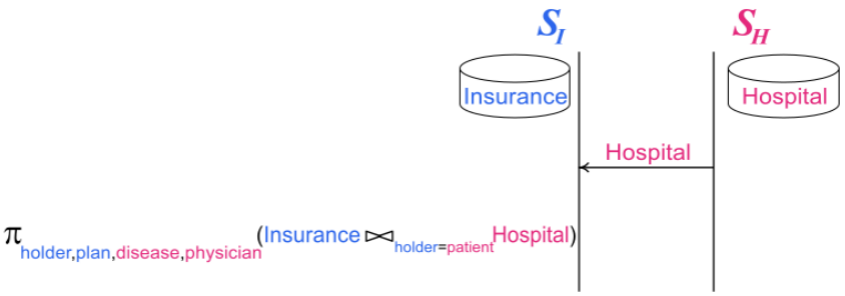
- Examples
  - S_D requires R :
    SELECT illness
    FROM Disease_list JOIN Hospital ON illness=disease
    WHERE treatment='antihistamine'
  - Relation profile: [(illness),(⟨D.illness,H.disease⟩),(treatment)]
  - Authorization
    [(illness,treatment),(⟨D.illness,H.disease⟩)] $\rightarrow S_D$
    authorizes the query
  - Authorization [(illness,treatment),_] $\rightarrow S_D$
    does not authorize the query

2.4 Rilasci autorizzati

Devo trovare un modo per eseguire la computazione in modo che rispetti i vincoli del sistema. Le operazioni si dividono in:

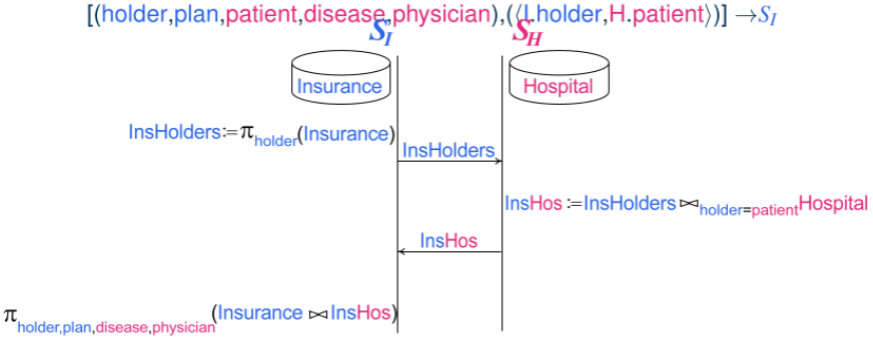
- **Unarie**; possono essere eseguite dal server S che tiene la relazione
  - proiezione: $\pi_X(R)$
  - selezione: $\sigma_X(R)$
- Operazioni di **join**; possono essere eseguite solo se implicano il rilascio di *viste autorizzate*. Si possono usare due strategie per eseguire il join soddisfando le autorizzazioni:
  - Regular join (master e slave)

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



  - Semi-join

- 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⟩)] $\rightarrow S_I$



2.5 Algoritmo

Data la computazione che voglio eseguire e dato l'insieme di autorizzazioni, l'obiettivo è quello di calcolare come eseguire la computazione in modo da rispettare le autorizzazioni.
L'idea è di fare l'assegnamento in due passi:

1. Cerco tutti i soggetti che sono potenzialmente autorizzati ad eseguire una operazione
   - faccio una visita post-order dell'albero (L, R, root; in pratica dalle foglie risalgo)
2. Faccio una visita in pre-order dell'albero e ne scelgo uno; può essere fatta in diversi modi in base a qual è il parametro di voglio ottimizzare

2.6 Sintassi vs Semantica

- 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),(⟨I.holder,H.patient⟩),_]

Per vedere se il server è abilitato ad accedere al risultato della query, devo vedere qual è il contenuto informativo associato alla query, ovvero il suo profilo.
Devo vedere se le autorizzazioni del sistema coprono il profilo della query: quello che vediamo è che non c'è una autorizzazione esplicita che permette di effetuare il join e di accedere a quella particolare query; tuttavia, ha più permessi che composti tra loro permettono di accedere al medesimo risultato.
Se mi va bene avere più autorizzazioni che composte tra loro permettono di accedere allo stesso risultato, è un tipo di approccio **semantico**.
Se voglio che ci sia una autorizzazione esplicita fatta in modo preciso è un tipo di approccio **sintattico**.

Capitolo 3

Composizione di autorizzazioni

Facciamo diverse assunzioni:

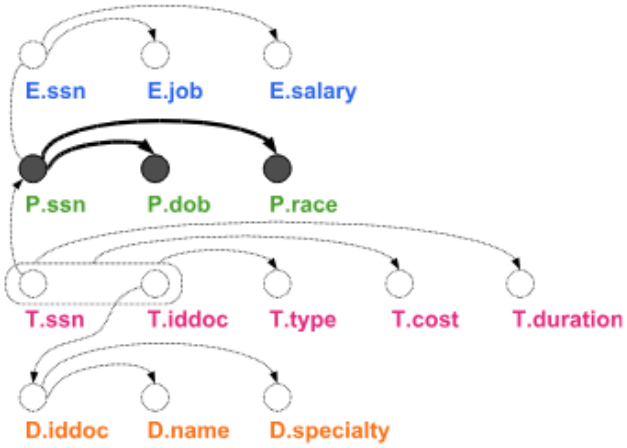
- lo schema è privo di cicli
- consideriamo solo join di tipo *naturale*; se c'è un informazione che è presente in più relazioni, allora supponiamo che usi sempre lo stesso nome (esempio SSN)
- Le composizioni vengono definite sempre per *stesso soggetto*, diventa una semplice coppia $[Attr, Rel]$
- Il profilo della relazione $[R^{\pi}, R^{\bowtie}, R^{\sigma}]$ viene semplificato come $[Attr, Rel]$ dove:
  - $Attr = R^{\pi} \cup R^{\sigma}$
  - Rel sono le relazioni coinvolte nel join path $R^{\bowtie}$ (mi basta specificare che il join lega due relazioni, senza specificare su quali il attributo perché abbiamo supposto di avere *join naturali*)

3.1 *Schema graph*

Conviene rappresentare attraverso un grafo lo schema per capire meglio quando è *safe* comporre delle autorizzazioni.
Un *grafo di schema* a partire da un set di relazioni è un grafo misto dove:

- ci sono tanti nodi quanti sono gli **attributi** delle relazioni
- ci sono degli **archi orientati** che rappresentano le **dipendenze funzionali** da una chiave verso tutti gli altri attributi della medesima relazione

- ci sono degli **archi orientati** che rappresentano i vincoli di **integrità referenziale**
- ci sono **archi non orientati** che rappresentano le operazioni di **join**



3.2 *Views*

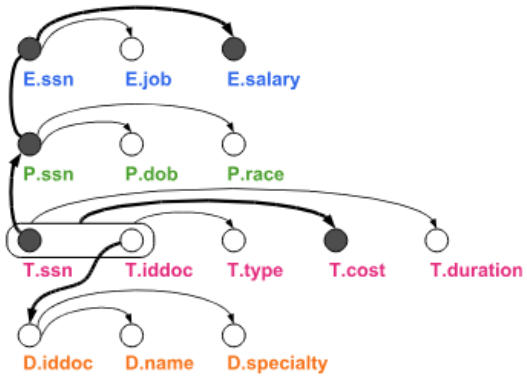
Permessi e query $[Attr, Rel]$ sono una **vista** su un set di relazioni R del mio sistema, dove il contenuto informativo di questa vista non cambia se viene estesa andando a chiuderla usando i vincoli di integrità referenziale; si definisce un *insieme di chiusura di relazione* ottenuto seguendo i vincoli di integrità referenziale a partire da una relazione.

3.3 *View Graph*

Una vista può essere rappresentata graficamente attraverso una colorazione dello *schema graph*; viene colorata la porzione del grafo che corrisponde di una vista $[Attr, Rel]$ nel seguente modo:

- di nero gli attributi che compaiono esplicitamente in $Attr$
- di nero tutti gli archi che corrispondono al join path definito sull'insieme di chiusura di Rel, oppure gli archi che partono da una chiave e vanno versi attributi neri
- di bianco tutti gli attributi che non sono neri della chiusura di Rel e gli archi che li connettono alla chiave primaria
- *clear* per tutti gli altri attributi e archi (un terzo colore)

```
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
```



In figura la vista del risultato è $[(ssn, salary, cost), (Employee, Patient, Treatment)]$

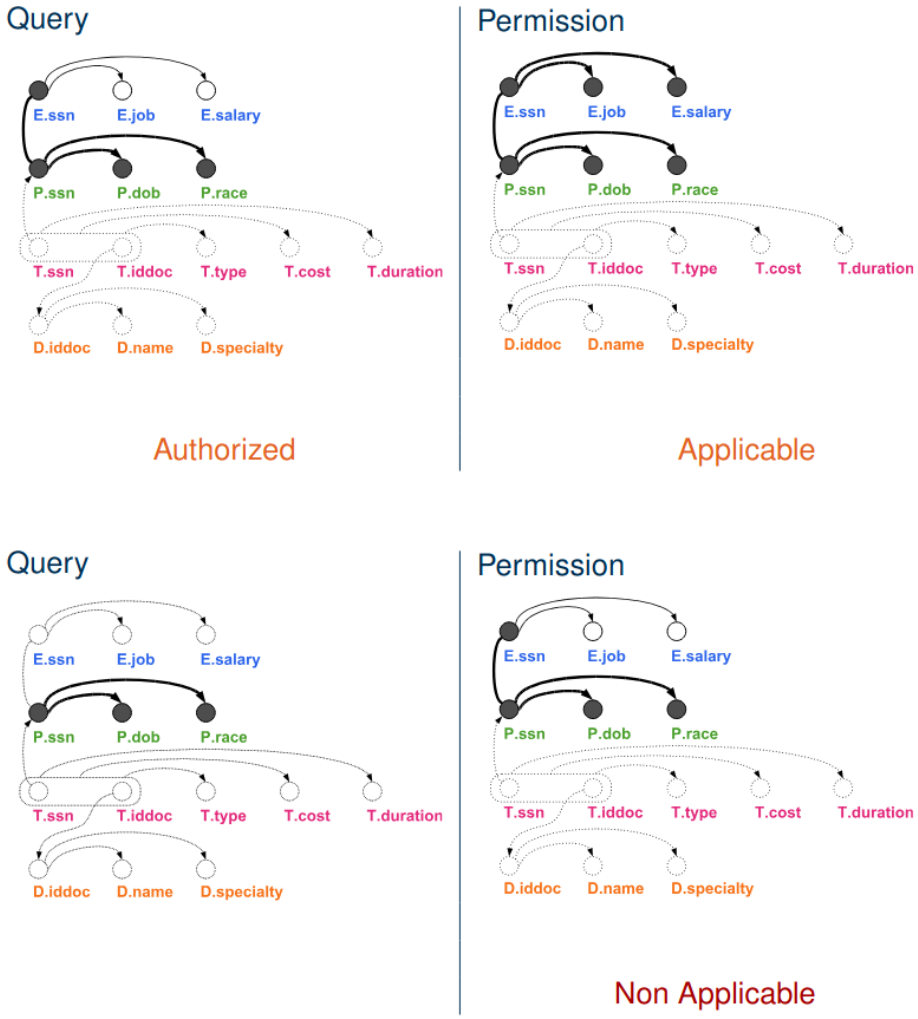
3.4 Autorizzare le query

Un soggetto vuole eseguire una query; il sistema deve controllare se il soggetto è autorizzato ad accedere al risultato della query; se non può accedervi, la query non viene neanche eseguita.
Ho da una parte il profilo della query e dall'altra la vista dei permessi; devo prima vedere quando una autorizzazioni si applica alla query (quando riguarda le stesse informazioni che ti sto chiedendo), e vedo se mi danno il permesso di accedere al risultato:

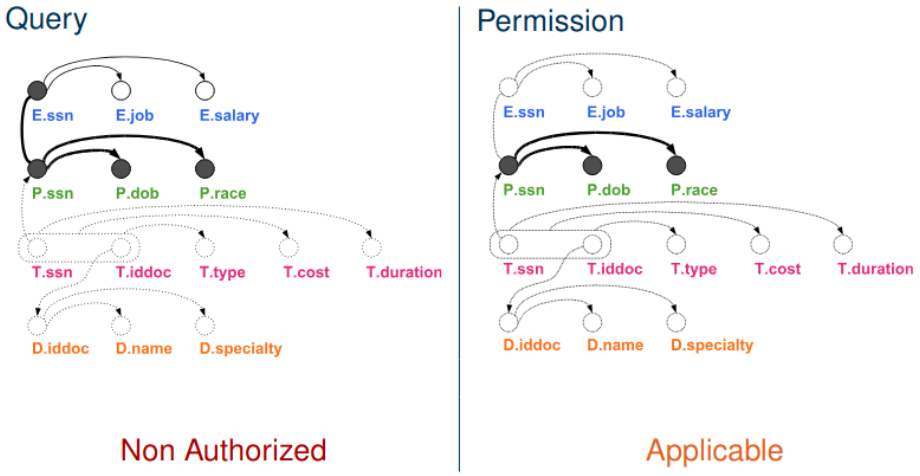
- $p = [Attr_p, Rel_p]$ si **applica** a $q = [Attr_q, Rel_q]$ sse $Rel_p^* \subseteq Rel_q^*$
- $p = [Attr_p, Rel_p]$ **autorizza** a $q = [Attr_q, Rel_q]$ sse:
  - p si applica a q
  - G_q e G_p hanno gli **stesso** archi neri di integrità referenziale e join
  - tutti i nodi che sono neri G_q lo sono anche in G_p

  $\rightarrow$ devo sostanzialmente vedere che tutto ciò che è nero nella query lo è anche nei miei permessi, sia archi che nodi

3.4.1 Esempi



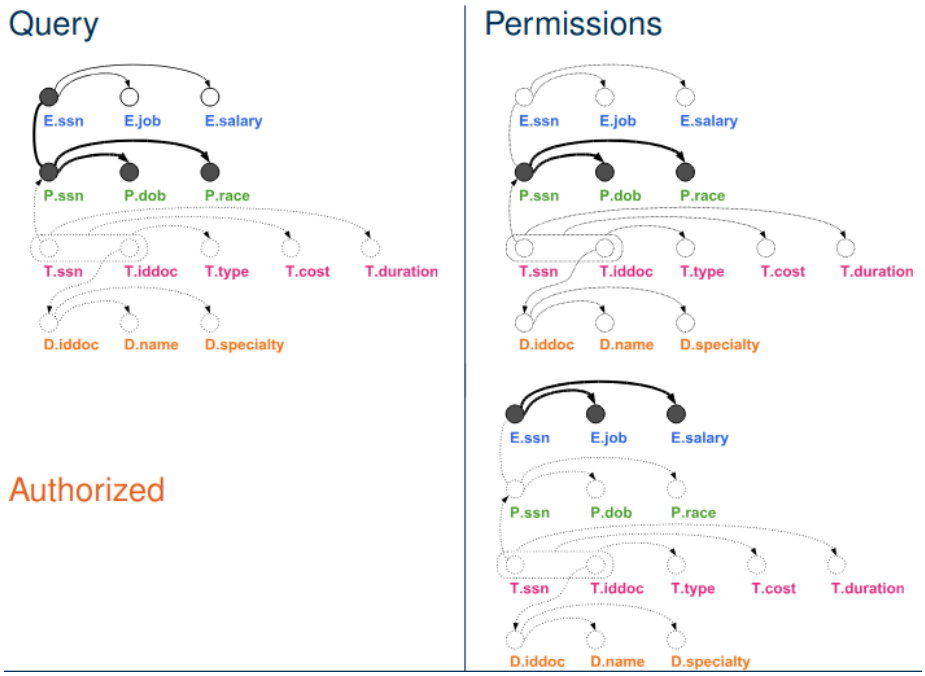
$Rel_p^* = \{E, P\}$, mentre $Rel_q^* = \{P\}$.
Non posso applicare questo permesso perché c'è il join, mi porta un contenuto informativo maggiore che non c'è nella query. Il permesso mi dice *tutti i pazienti che sono anche impiegati*; la query mi dice *tutti i pazienti*. Me ne accorgo perché ci sono delle operazioni di join che nella query non ci sono, che mi danno delle informazioni in più (e allo stesso mi toglie).

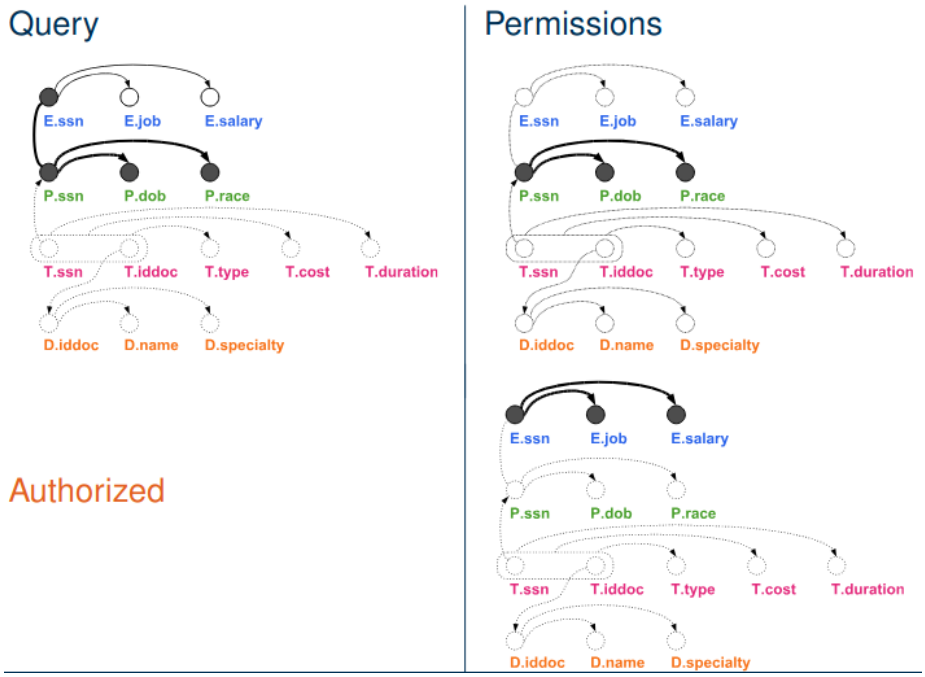


3.5 Composizione di permessi

Questo modello mi permette di applicare un approccio semantico; posso avere più permessi che mi vanno a coprire la query che mi permettono di accedere al risultato.
Una query può essere eseguita se il soggetto ha i permessi per vedere il contenuto informativo della query; una query dovrebbe essere autorizzata se il soggetto ha i permessi per computare in modo indipendente il risultato.
Un singolo permesso potrebbe non essere sufficiente, dunque si usa la **composizione** di permessi. Bisogna fare attenzione al rischio di *disclosure indiretta*.

3.5.1 Esempi


Query
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
Authorized
Permissions
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
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



Potremmo fare il ragionamento di *sovrapporre* i permessi: avremmo tutto ciò che ci serve colorato di nero, e potremmo pensare di essere autorizzati ad accedere alla query.
Tuttavia, questa composizione non può essere fatta perché manca l'associazione tra SSN (un paziente specifico) e *specialty*; anche eseguendo singolarmente le computazioni non rieco a ricostruire il risultato della query.

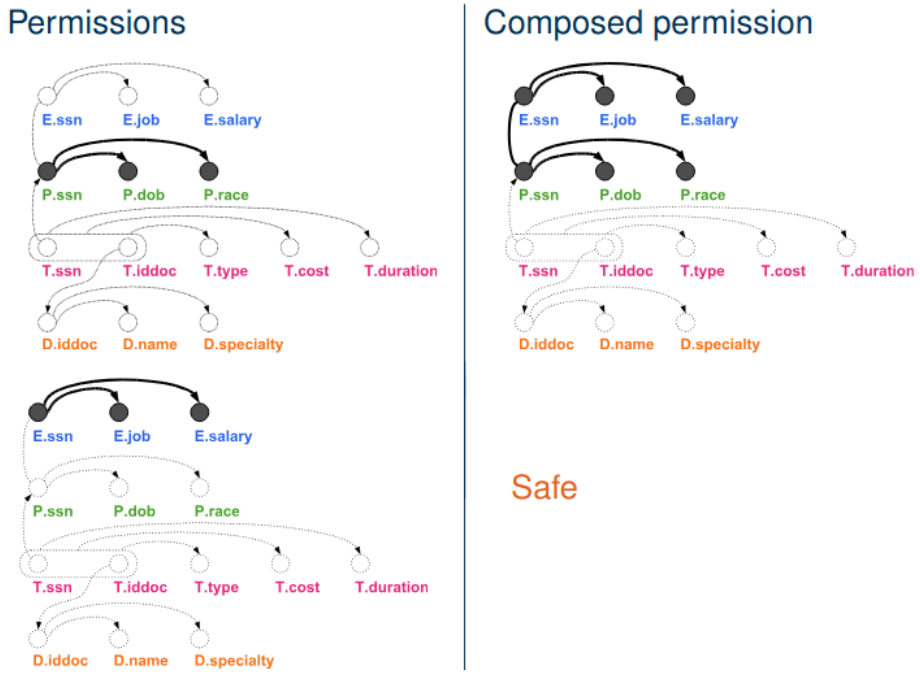
3.6 Composizione sicura

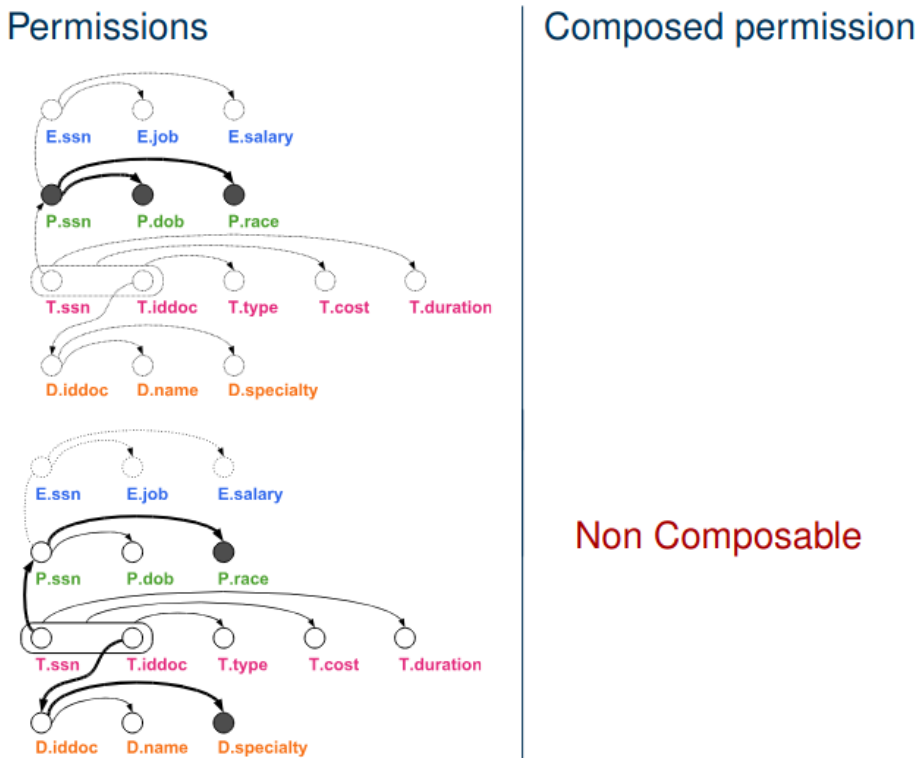
Due permessi possono essere composti in maniera **sicura** sse la loro composizione **non aggiunge informazione**.

- $p_i \to p_j$: $p_j = [Attr_j, Rel_l]$ dipende da $p_i = [Attr_i, Rel_i]$ sse l'intersezione degli insiemi di attributi $Attr_i$ e $Attr_j$ è un insiemi di attributi tutti neri in p_j, e a partire da questi attributi ci deve essere un cammino (in uno dei due grafi) che mi permette di raggiungere tutti gli altri nodi neri attraversando solo archi neri

  $\Rightarrow$ se è verificato posso comporre i permessi

3.6.1 Esempi


Permissions
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
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
Composed permission
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
Safe



3.7 Esecuzione di Access Control

Data una query, come faccio a vedere se è può essere autorizzata oppure no? Il processo richiede tre passi:

1. determino tutti i permessi applicabili alla query
2. determino tutti i permessi che ottengo attraverso il processo di composizione (fino a a quando termina; quando compongo un permesso allo stesso tempo uno viene tolto dall'insieme, per cui a una certa il processo termina)
3. guardo se esiste un permesso (tra quelli esistenti e quelli che ho ottenuto con la composizione) che copre la query che voglio eseguire

L'efficienza garantita dallo sfruttamento dell'implicazione dei permessi:

- Se $p_j \rightarrow p_i$, $\forall p_k \in P$:
  - $p_j \rightarrow p_k \Rightarrow (p_i \otimes p_j) \rightarrow p_k$
  - $p_k \rightarrow p_j \Rightarrow p_k \rightarrow (p_i \otimes p_j)$
    $\rightarrow$ aggiungendo $p_i \otimes p_j, p_j$ può essere rimosso da P
- Non c'è necessità di computare tutte le $2^n - 1$ composizioni di n permessi

- La complessità computazionale rimane polinomiale ossia $O(n^3)$

Spieghino della prof: Supponiamo di avere tre permessi, p_i, p_j e p_k quello che osseravo è che p_i e p_j sono due autorizzazioni che possono essere composte tra di loro perchè p_i dipende da p_j. Se p_i e p_j sono componibili tar di loro e se esiste un'altra autorizzazione p_k tale per cui p_k dpende da p_j allora in pratica p_j si può buttar via, tanto p_k dipende dal risultato della composizione tra p_i e p_j. Questo è vero non solo quando p_k dipende da p_j ma anche quando p_j dipende da p_k, in ogni caso io p_j posso buttarlo via senza perdere nulla.
N.B. : La prof ha fatto degli esercizi su sta cosa

Capitolo 4

Un modello di autorizzazione per query multi-provider

Normalmente un modello di autorizzazione è binario, nel senso che puoi accedere all'informazione oppure no; la peculiarità del modello che adesso vediamo è che aggiunge un *terzo livello di visibilità*:

- puoi accedere ai dati
- non puoi accedere ai dati
- puoi accedere ai dati criptati

Il vantaggio è che potrebbe essere conveniente in termini economici, usando magari server poco fidati.

4.1 Definizione del problema

- **INPUT:**
  - Query
  - Set di cloud providers, ciascuno con le sue informazioni che sono messe a disposizione per poter eseguire le query in modo collaborativo
  - Specifica di autorizzazioni
- **OUTPUT:**
  - Assegnare le operazioni ai soggetti in modo da soddisfare le autorizzazioni (che ciascun soggetto ha definito sui propri dati) e minimizzare i costi

L'idea è di dare l'autorizzazione a un soggetto di eseguire delle operazioni no sui dati in chiaro ma sui dati cifrati, può essere conveniente quando mi fido di un soggetto ma non troppo.

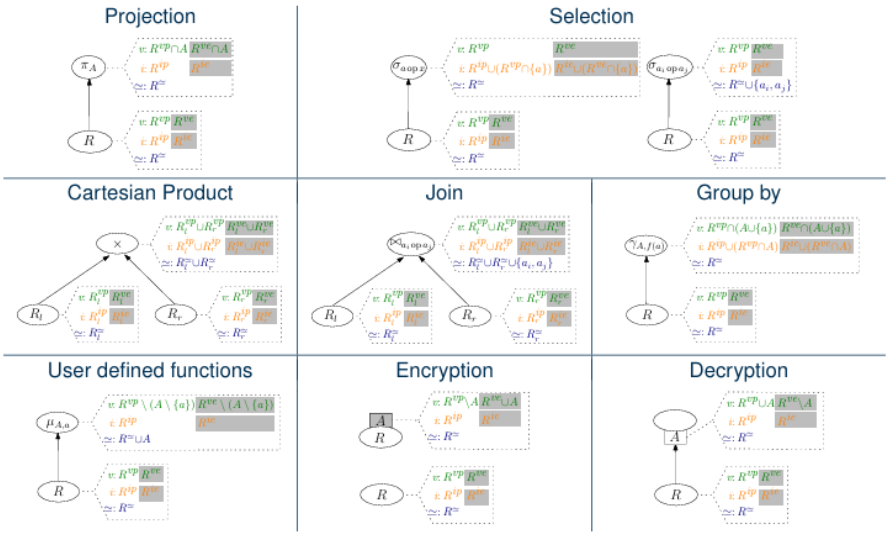
4.2 Profilo della relazione

È il bagaglio informativo della relazione;no solo inteso come attributi della relazione ma anche altre informazioni che non appaiono in modo esplicito, deve essere adattato a questo modello ed include tre componenti:

- v: attributi espliciti dello schema della relazione, sia in chiaro che in forma criptata (devo distinguere queste due situazioni)
  - prima erano R^{π}
- i: attributi impliciti (per costruire la relazione ho usato delle condizioni su attributi in chiaro o criptati), sia in chiaro che in forma criptata (devo distinguere queste due situazioni), le informazioni che io vedo dipendono in qualche modo da questi attributi su cui io ho applicato dterminate condizioni.
  - **Selection:** faccio una selezione dove scelgo in base alla peculiarità di un determinato altro attributo, creo un collegamento con l'attributo peculiare anche se non fa parte dello schema
  - **Grouping:** selezione facendo raggruppamento
  - prima erano R^{σ}
- $\simeq$: ho gli attributi coinvolti in operazioni di join (si suppone join naturali, prima erano $R^{\bowtie}$)
  - **Comparing** attributes: facciamo un confronto ad esempio $S = C$, se io confronto questi attributi e mettiamo caso che per l'utente lattributo S debba essere criptato, essendo un'equivalenza basta poter vedere C per conoscere anche S che però non ho l'autorizzazione per conoscere.

    Per ovviare a questo problema se si deve dare questa relazione a un soggetto questo soggetto può ricevere la relazione solo se è abilitato a vedere S e C allo stesso modo.

4.3 profili risultanti dalle operazioni



Tutti gli esempi sono organizzati nello stesso modo: a sx il modo generale a dx un'esempio.

4.3.1 Proiezione

Se io parto da una relazione R che ha questi profilo ed eseguo una proiezione, la relazione risultato ha lo stesso profilo, l'unica cosa che cambia sono gli attributi **visibili**.

- Esempio: nella relazione la proieizione viene fatta su B e su P butto via D e T.

4.3.2 Selezione - 1

Selezione = io vado ad eseguire una condizione su determinati attributi. Viene fatta una distinzione su qual è la forma della condizione: se la condizione è *attributo* = *valore* mi devo ricordare che è stata eseguita quella condizoone su quell'attributo, quindi quell'attributo deve finire nella componente implicita del profilo della mia relazione.

- Esempio: Se io faccio "D=stroke" allora poi l'attributo D dovrà finire nella componente implicita

4.3.3 Selection - 2

In questo caso nella selezione vengono comparati due attributi (e non attributo con valore), ha effetto sul campo delle equivalenze

- Esempio: SC viene aggiunto al campo equivalenze

4.3.4 Prodotto cartesiano

Quando viene fatto il prodotto cartesiano tra due relazioni devo unire i profili delle relazioni

- Esempio: si uniscono tutti i campi delle due relazioni

4.3.5 Join

Vuol dire ancora una volta che devo fare l'unione dei due profili, è simile al prodotto cartesiano

- Esempio: in questo caso mi devo ricordare che oltre a unire i profili ho anche confrontato gli attributi D e C e questo va aggiunto nella componente delle equivalenze

4.3.6 Group by

Raggruppo le tuple della mia relazione in base a uno o più attributi,e su questi gruppi posso eseguire delle funzioni di aggregazione.

- Esempio: in questo caso average(AVG) viene applicato alla relazione, io ottengo una nuova relazione che ha come schema l'attributo T e quello che ho come secondo attributo non è proprio l'attributo P, ma è il risultato di una funzione che io ho applicato all'attributo P. Devo ricordarmi di aggiungere l'attributo T negli attributi impliciti della mia relazione, a livello di equivalenza invece non cambia nulla.

4.3.7 Funzioni definite dall'utente

Sono delle funziomi che io posso avere all'interno delle query SQL tipo machine learning, io non so di preciso questa funzione.

- Esempio: cambia lo schema degli attributi visibili, è una funzione UDF che in una coppa (S, B) prende solo l'attributo S. Fondamentalmente sparisce l'attributo B, nella classe delle equivalenze viene fatta l'unione tra SC già presenti e SB confrontati ora e ottengo SBC.

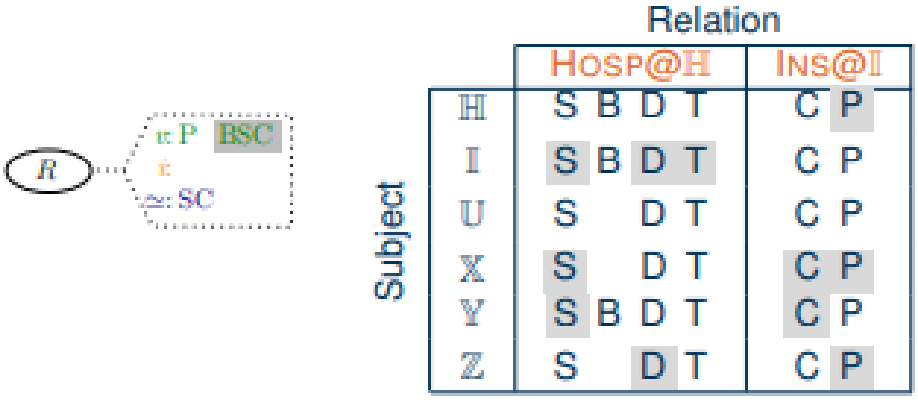
4.3.8 Encryption

Se cripto un attributo in una relazione, quell'attributo sarà ancora parte della relazione però criptato.

4.3.9 Decryption

Se decripto un attributo, poi lo vedo in chiaro.

4.4 Visibilità autorizzata



Introduciamo il concetto con un esempio: Ummaginiamo di avere sempre le nostre due relazinoi Hospital e Insurance. Nella tabella vediamo rappresentate le autorizzazioni definite su queste due relazioni per i vari soggetti all'interno del sistema.
H sta per Hospital ed è il proprietario della relazione Hospital, I sta per Insurance ed il proprietario della relazione insurance, poi abbiamo altri soggetti, in questo esempio U è l'utente che richiede la computazione.
A sx R è la relazione che è il risultato di una query.
Data questa relazione e questa tabella andiamo a vedere chi è autorizzato ad accedere alla relazione R: Intuitivamente un soggetto è autorizzato ad accedere alla relazione R se può accedere in chairo a tutti gli attributi che sono in chairo nella relazione e può accedere agli attributi che appaiono in forma criptata o in chairo o in modalità criptata (questo per quanto rguarda gli attributi nei campi v e i). Per wuanto riguarda il campo delle equivalenze un soggetto deve avere una visibilità uniforme su questi attributi.

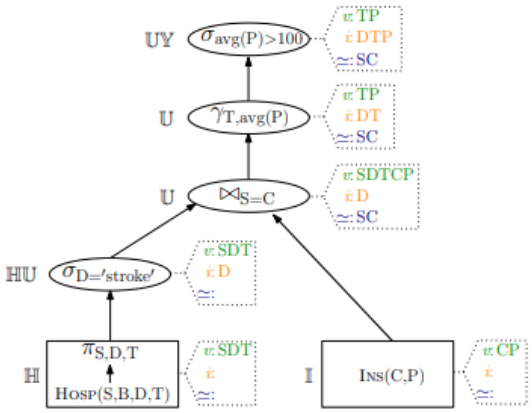
- **Soggetto H:** il soggetto H ha una visibilità criptata sull'attributo P quindi non può accedere alla relazione
- **Soggetto I:** il soggetto I è okay per gli attributi visibili ma non ha visibilità uniforme sulle equivalenze poichè S è criptato e C in chiaro, quindi non è autorizzato ad accedere
- **Soggeto U:** non ha accesso all'attributo B, quini non è autorizzato
- **Soggetto X:** visibilità criptata su P quini no accesso

- **Soggetto Y:** può accedere.
- **Soggetto Z:** visibilità criptata su P quini no accesso

In conclusione:
Un soggetto S è autorizzato ad accedere alla relazione R se:

- visibilità del testo in chiaro su attributi in chiaro (visibili o impliciti)
- visibilità in chairo o criptata su attributi criptati (visibili o impliciti)
- visibilità uniforme (in chiaro o criptata) su attributi equivalenti

4.5 Calcolo delle assegnazioni

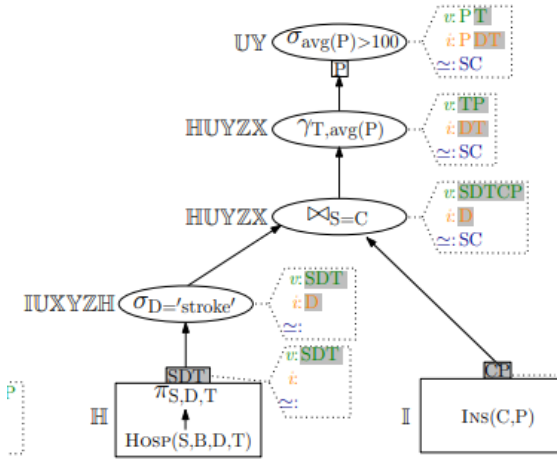


Parto da una situazione dove tutto è in chairo e guardo chi può eseguire le varie operazioni di cui la mia query si compone. Partiamo dalle foglie e andiamo verso l'alto, abbiamo sempre le nostre due relazioni Hospital e Insurance, di base queste due relazioni possono essere gestite da H per Hospital e I per Insurance.

- Analizzando la query è visibile come nella relazione Hospital l'attributo B non è mai coinvolto all'interno della computazione, quindi viene fatta una **proiezione**, rappresentata con π, non le rappresento con un nodo dell'albero poichè rientra tra le cose che può fare chi ha in mano la relazione.
- Andiamo sul nodo sopra con l'operazione di **selezione** su D, visto che l'idea è lasciare tutto in chiaro vado a vedere chi è autorizzato a vedere in chiaro la relazione H che contiene al suo interno gli attributi S , D e T. Guardando la tabella notiamo che solo gli utenti H e U sono autorizzati a vedere questi attributi in chairo.

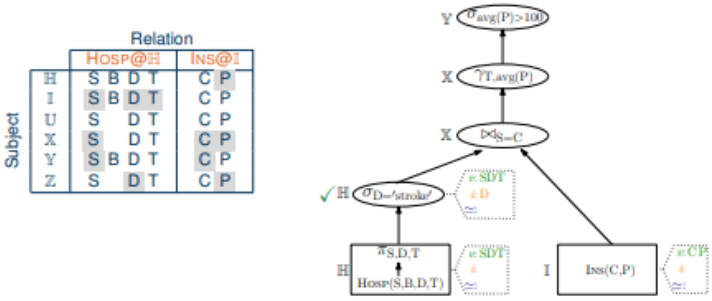
- Arrivo al nodo di join, l'unico soggetto che può vedere la relazione che da una parte ha gli attributi SDT in chairo e gli attributi CP dell'altra relazione in chairo è solo il soggetto U
- Questo ragionamento è valido anche per tutti gli altri nodi, notiamo che però se lasciamo tutto in chairo ogni nodo è caratterizzato da pochi candidati.

Faccio l'opposto e parto dall'assunzione che sia **tutto criptato** (guardare immagine sotto)



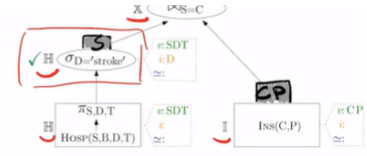
Supporre di avere tutto criptato aumenta notevollmente il numero di candidati, infatti nella realtà non è detto che sia tutto criptato bensì è solo una strategia che applico per trovare i candidati. In questa query c'è un assunzione particolare: la P che si trova sotto l'ultimo nodo dell'albero (quello più in alto) indica l'**operazione di decriptazione**, viene fatto poichè determinate operazioni vogliono essere eseguite sugli attributi in chairo. Il valore che dipende dall'attributo P deve essere decriptato.

4.6 Piano di query minimamente esteso



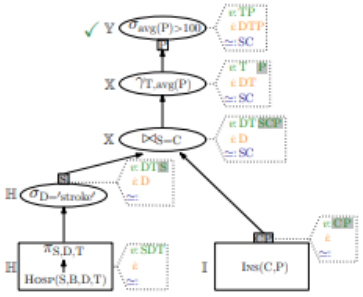
In qualche modo abbiamo applicato un qualche criterio e questi sono i soggetti che abbiamo scelto.

- Il soggetto H deve compiere l'operazione di selezione su D, e al fine che la possa compiere io devo lasciare tutto così com'è e non criptare nulla.
- Per l'operazione di Join abbiamo scelto il soggetto X, però può accedere gli attributi solo in modalità criptata. Quindi quando vado ad eseguire la query devo iniettare delle operazioni di **criptazione** (tutto questo dipende dal sogetto che scelgo per fare una determinata operazione)



- L'operazione successiva la svolge sempre X che consegna il risultato al soggetto Y, l'operazione che Y deve fare va eseguita in chiaro sull'attributo P che però il soggetto X ha criptato, quindi decripta l'attributo P prima di svolgere l'operazione.

Scherma finale risultante:



Problema della gestione delle chiavi di criptazione: chiavi che devono essere condivisi tra più soggetti, quelli che eseguiranno l'operazione.