Controllo delle Query Distribuite

Parte III

Indice

1 Introduzione **2**
- 1.1 Join sovrani . . . 2
- 1.2 Access patterns . . . 3
- 1.3 Autorizzazioni basate su viste . . . 3
- 1.4 Coalition networks . . . 4
  - 1.4.1 Broker join . . . 5
  - 1.4.2 Peer-join . . . 5
  - 1.4.3 Semi-join . . . 5
  - 1.4.4 Split-join . . . 5
- 1.5 Preferenze in ottimizzazione delle query . . . 7

2 Valutazione di query distribuite sotto requisiti di protezione **8**
- 2.1 Permessi . . . 9
- 2.2 Profilo della relazione . . . 10
- 2.3 Vista autorizzata . . . 11
- 2.4 Rilasci autorizzati . . . 11
- 2.5 Algoritmo . . . 12
- 2.6 Sintassi vs Semantica . . . 13

3 Composizione di autorizzazioni **14**
- 3.1 *Schema graph* . . . 14
- 3.2 *Views* . . . 15
- 3.3 *View Graph* . . . 15

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

$$\text{Insurance}^{oi}(\underline{\text{holder}},\text{plan})$$
$$\text{Hospital}^{oioo}(\underline{\text{patient}},\text{YoB},\text{disease},\text{physician})$$
$$\text{Nat\_registry}^{ioo}(\underline{\text{citizen}},\text{YoB},\text{healthaid})$$

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`

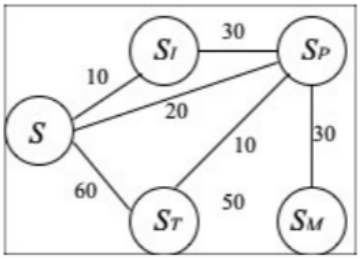
Verifico se una query può essere eseguita sulla base delle autorizzazioni che ho definito; il client scrive la sua query, e il server cerca di rielaborarla sulla base delle viste che sono state definite.
Nel caso in cui una query non possa essere eseguita, ci sono due scenari possibili:

- *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.

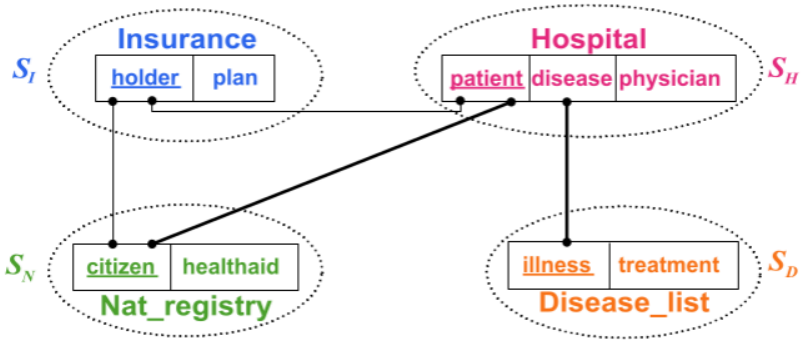
Capitolo 2

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*

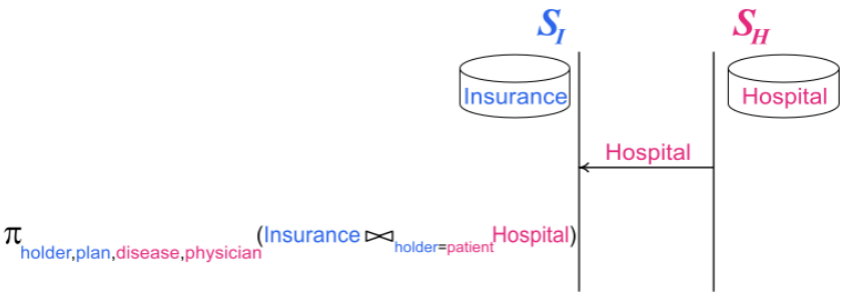
- Examples
  - S_D requires R :
    SELECT illness
    FROM Disease_list JOIN Hospital ON illness=disease
    WHERE treatment='antihistamine'
  - Relation profile: [(illness),(⟨D.illness,H.disease⟩),(treatment)]
  - Authorization
    [(illness,treatment),(⟨D.illness,H.disease⟩)] $\rightarrow S_D$
    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:

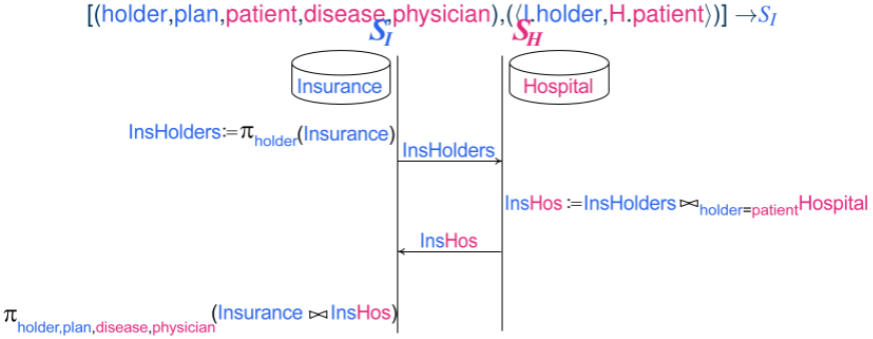
- **Unarie**; possono essere eseguite dal server S che tiene la relazione
  - proiezione: $\pi_X(R)$
  - selezione: $\sigma_X(R)$
- Operazioni di **join**; possono essere eseguite solo se implicano il rilascio di *viste autorizzate*. Si possono usare due strategie per eseguire il join soddisfando le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

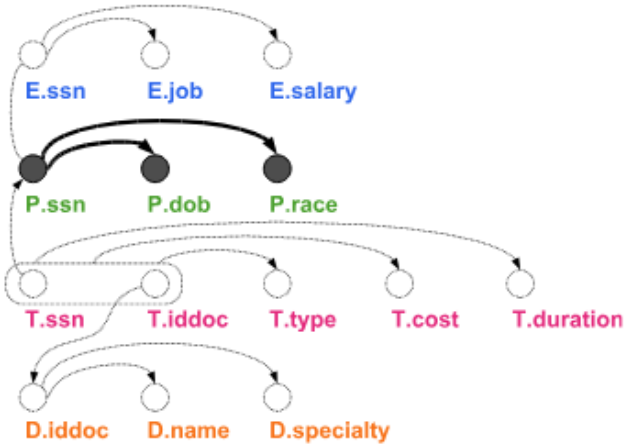
- lo schema è privo di cicli
- consideriamo solo join di tipo *naturale*; se c'è un informazione che è presente in più relazioni, allora supponiamo che usi sempre lo stesso nome (esempio SSN)
- Le composizioni vengono definite sempre per *stesso soggetto*, diventa una semplice coppia $[Attr, Rel]$
- 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*

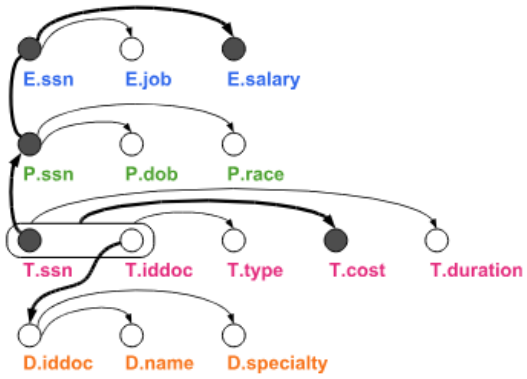
Permessi e query $[Attr, Rel]$ sono una **vista** su un set di relazioni R del mio sistema, dove il contenuto informativo di questa vista non cambia se viene estesa andando a chiuderla usando i vincoli di integrità referenziale; si definisce un *insieme di chiusura di relazione* ottenuto seguendo i vincoli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