Integrità delle Query

Parte II

Indice

Capitolo 1

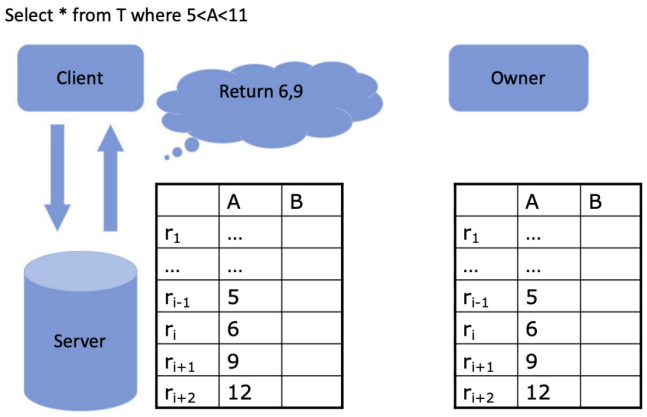
Integrità della computazione

Nel nostro scenario di riferimento potremo avere più *data owner*; queste sono entità di cui ci fidiamo.
Il problema è che questi dati potrebbero essere affidati a dei *cloud provider* esterni, e che possano essere soggetti a delle *computazioni*; questo potrebbe essere un problema sia in termini di confidenzialità che in termini di **integrità**: *"chi mi dice che la tua computazione sia integra?"*.

1.1 Esempi

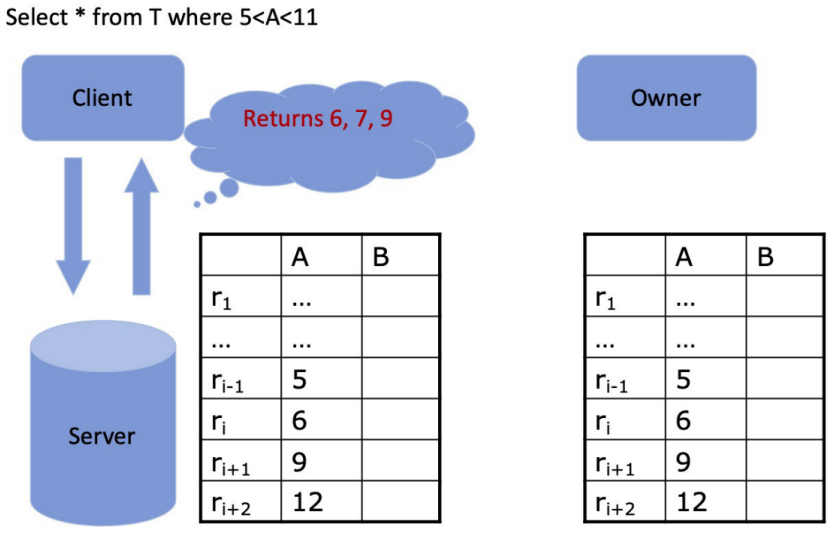
Esempio di una query

Abbiamo l'owner che affida i propri dati ad un provider esterno; abbiamo poi un client che effettua una query.

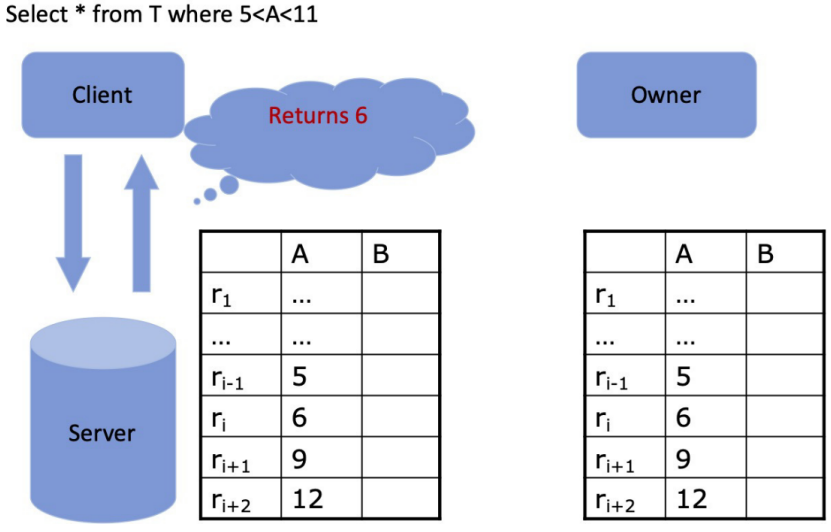


Esempio di query: iniezione

Viene iniettata un'informazione fasulla; *magari mi conviene dirti una cosa piuttosto che un'altra, le tue azioni dipendono da quello che ti dico...*

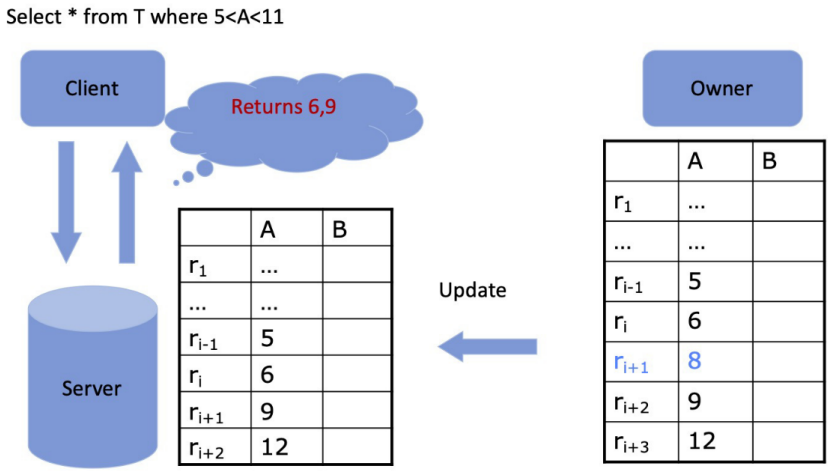


Esempio di query: drop



Esempio di query: omissione

I dati potrebbero essere dinamici, dunque potrebbero essere richieste delle operazioni di update.



1.2 Integrità di storage e computazione

Il data owner e gli utenti necessitano di meccanismi che assicurino l'integrità dei risultati delle query. Una query è integra se rispetta:

- **Correttezza:** il risultato viene calcolato sui dati veri dell'owner (primo esempio)
- **Completezza:** il risultato calcolati su tutti i dati (secondo esempio)
- **Freschezza:** il risultato è calcolato sull'ultima versione dei dati che l'owner ha dato (terzo esempio)

Ci sono due diversi tipi di approcci per rispondere al problema di integrità, ciascuno con i suoi vantaggi e svantaggi:

- **Deterministico:** *se il risultato di una computazione è integro, sono sicuro al 100% che sia integro*

  Queste tecniche vengono implementate in modo che l'owner dà al provider, oltre ai dati da gestire, anche delle strutture ausiliarie che vengono sfruttate per verificare l'integrità della computazione

- **Probabilistico:** *ti dico sempre se è integro o no, ma non con certezza assoluta ma con una certa probabilità; c'è della probabilità di fare degli errori*

  Perché si usano questi approcci? Sono tecniche che hanno lo svantaggio di non avere la certezza assoluta ma che hanno altri vantaggi (che vedremo

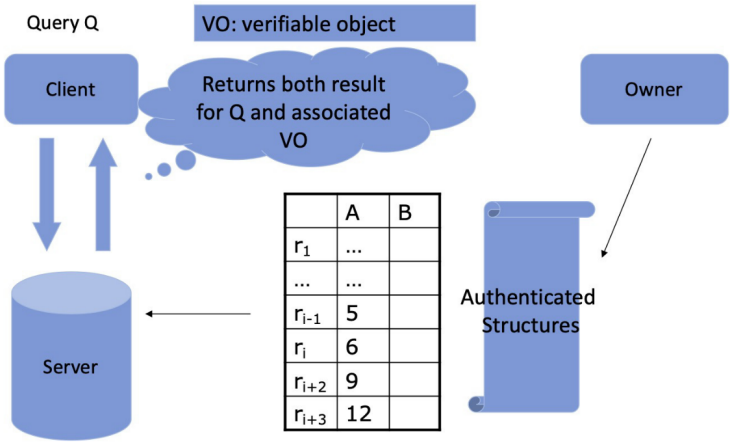
più avanti); il fatto che non avere certezza sia un problema dipende da caso a caso.

In queste tecniche il *qualcosa di auisiliario* sono dei "dati finti" (marcatori) che "aggiungo" ai dati veri; dalla presenza o meno capisco se la query è integra o no (se prima c'erano e poi non ci sono più, probabilmente c'è un errore)

Capitolo 2

Approcci deterministici

L'idea è che il proprietario *dà fuori* i dati e una struttura da lui calcolata. Quando il client vuole fare una computazione, restituisce oltre al risultato anche un *qualcosa in più* usando la struttura dati; questo prende il nome di **verification object**: è ciò che permette di verificare se il risultato della query è integro.



2.1 Approccio basato su firma

Questa tecnica si preoccupa di verificare l'integrità solo per una tipologia particolare di query, ovvero quelle che coinvolgono un solo attributo della relazione; ad esempio $x = 5, 4 < x < 5, \ldots$ l'idea è:

- ordinare le tuple rispetto al valore dell'attribuo preso in considerazione
- applicare una firma alle tuple, non singolarmente ma in coppie tra loro consecutive

  $(t_1, s_1), (t_2, s_2) \ldots (t_n, s_n), cons_i = \epsilon(t_i | t_{i+1})$

- oltre ai dati, vengono date al provider anche le firme

A questo punto quando un client vuole eseguire una computazione, ad esempio $a < x < b$:

- vengono restituite le tuple (e le firme associate) $[a-1, b+1] \rightarrow$ voglio anche la tupla immediatamente precedente ed immediatamente successiva
- le *cose aggiunte* al risultato vero e proprio per verificare l'integrità sono:
  - tuple precedente e successiva
  - firme associate alle tuple

$\Rightarrow$ l'idea è che il client tramite le firme può verificare se il risultato è integro.
Questo metodo non è molto utilizzato perché:

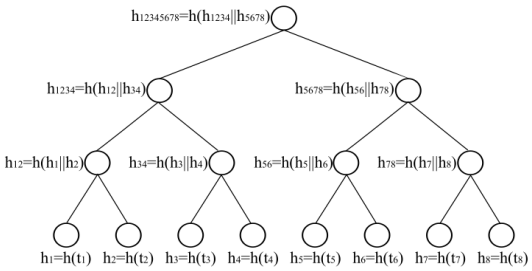
- limitazione sulle query
- costosa sia in termini di computazione delle firme, sia nei termini di informazioni aggiuntive che ti devo dare (lineare rispetto al risultato)

2.2 Merkle hash tree

Questa tecnica può essere utilizzata per risolvere lo stesso tipo di query viste nella sezione precedente, ma in maniera più efficiente.

Patients

	SSN	Name	Disease
t_1	123-45-6789	Alice	Asthma
t_2	234-56-7891	Bob	Asthma
t_3	345-67-8912	Carol	Asthma
t_4	456-78-9123	David	Bronchitis
t_5	567-89-1234	Eva	Bronchitis
t_6	678-91-2345	Frank	Gastritis
t_7	789-12-3456	Gary	Gastritis
t_8	891-23-4567	Hilary	Diabetes



Merkle hash tree over attribute SSN

L'idea è:

- ordinare i valori dell'attributo preso in considerazione
- si applica una funzione di hash alle tuple (foglie dell'albero)
  - nel livello delle foglie ci sono 2^L elementi
  - i nodi intermedi vengono calcolati applicando la stessa funzione di hash alla concatenazione degli hash dei figli
    $\rightarrow$ l'idea è che l'hash di un nodo dipende dall'hash dei figli

- se l'albero non è completo, tipicamente si aggiungono delle tuple *null* per renderlo completo

$\Rightarrow$ la quantità di informazioni aggiuntive non è più lineare rispetto al risultato ma è **logaritmica**.