Cosa serve orale

domenica 7 luglio 2024 17:08

Dip. funz
chiave relazione
3NF
Assiomi di Armstrong
chiusura di X+ (parte correttezza alg, no la parte X+ rispetto a G contenuto in Zfinale)
copertura minimale di F
Scomposizioni JSP (non serve proprietà di mrho(r) (join delle decomp.))
Alg. scomposizioni JSP (parte correttezza alg, no che se la tabella fin ha riga con solo "a" allora è JSP)
Copertura minimale
scomposizioni che preservano F (non si dimostra che aggiungendo schema con chiave si ottiene JSP)

org. fisica solo capire le caratteristiche delle strutture

teoria concorrenza solo th. 2 fasi, correttezza regole timestamp

Fanno eccezione ovviamente, CIOE' NON VERRANNO CHIESTE, quelle dimostrazioni che sono state esplicitamente escluse anche se compaiono sulle slide del corso e sulla dispensa. Quindi gli argomenti oggetto di esame sono:

TUTTI i teoremi della teoria relazionale TRANNE:

- parte se della dimostrazione di correttezza dell'algoritmo per il calcolo della chiusura di X rispetto a G (dove G è l'insieme di dipendenze che risulta da una decomposizione) **quindi NON si dimostra la parte la parte X+ rispetto a G contenuto in Zfinale**

- non si dimostrano le proprietà di mρ(r) ma bisogna AVER CAPITO la sua relazione con r

- parte se della dimostrazione di correttezza dell'algoritmo che verifica il join senza perdita, **quindi non si dimostra che se la tabella finale ha una riga di tutte allora il join è senza perdita**

- dimostrazione della parte aggiunta all'algoritmo di decomposizione, **quindi non si dimostra che aggiungendo uno schema con una chiave si ottiene un join senza perdita**

Per **l'organizzazione fisica** occorre dimostrare di aver compreso le caratteristiche delle varie strutture ed essere in grado di dimostrare i costi delle perazioni.

Per la **teoria della concorrenza** si dimostra SOLO il teorema del protocollo a 2 fasi, e la correttezza delle regole del timestamp (anche se non c'è un teorema)

OVVIAMENTE non ci sono solo i teoremi, **ma anche le definizioni e occorre dimostrare di aver capito i concetti di base** ... in pratica i concetti che trovate su slide e dispense TRA i teoremi

OVVIAMENTE se c'è il dubbio che uno studente non sappia trovare una chiave o calcolare una chiusura, gli può essere chiesto di farlo all'orale (ANCHE CON LO SCRITTO COMPLETO!). OVVIAMENTE il codice degli algoritmi E' OGGETTO di orale come le dimostrazioni di correttezza (teoremi) (ANCHE CON LO SCRITTO COMPLETO)

Dipendenza funzionale

Sia R uno schema con istanza r e X,Y ⊆ R
La **Dipendenza funzionale** tra X e Y, indicata come X→Y, è un **vincolo di integrità** in cui ad ogni coppia di tuple in r se si hanno valori uguali su X allora devono essere uguali anche su Y

3NF

Se lo schema è in 3NF, il num. di **Anomalie** e Ridondanza **dei dati** è estremamente ridotta

R è in 3NF se per ogni dip. X→Y∈F (K chiave di R):
- X è **superchiave** (è K o sottoinsieme di K)
- Y è **attr. primo** (sottoinsieme proprio di K)
- X→Y non è **dip. transitiva:**
  - Y non è **primo**
  - Per ogni K si ha che X non appartiene a K e la loro sottrazione non è insieme vuoto
- X→Y non è **dip. parziale:**
  - Y non è **primo**
  - Esiste K tc X è **sottoinsieme proprio** di K

Armstrong

Gli **Assiomi di Armstrong** sono regole usate per trovare dipendenze funzionali aggiuntive partendo da quelle disponibili in F. F^A è l'insieme di tutte le dip. funz. ottenibili applicando i **seguenti assiomi** a F:

- **Inclusione iniziale*:** X→Y∈F allora X→Y∈F^A
- **Riflessività**: Se Y ⊆ X ⊆ R allora X→Y → $Y=\{A,B\}$ $X=\{A,B,C\} \Rightarrow \overset{X}{\{A,B,C\}} \to \overset{Y}{\{A,B\}}$
- **Aumento**: se X→Y∈F^A e Z∈R allora XZ→YZ → $X=\{A\}$ $Y=\{B\}$ $Z=\{C\} \Rightarrow \{A,C\} \to \{B,C\}$
- **Transitività**: se X→Y∈F^A e Y→Z∈F^A allora X→Z → $X=\{A\}$ $Y=\{B\}$ $Z=\{C\}$ $A \to B \to C \Rightarrow A \to C$

Possiamo dedurre altri assiomi aggiuntivi:
- **Unione**: se X→Y e X→Z allora X→YZ
- **Pseudotransitività**: se X→Y e WY→Z allora XW→Z → se $\overset{X}{A} \to \overset{Y}{B}$ e $\underset{W}{C}\underset{Y}{B} \to \underset{Z}{D} \Rightarrow \underset{X}{A}\underset{W}{C} \to \underset{Z}{D}$
- **Decomposizione**: se X→Y e Z⊆Y allora X→Z → se $\underset{X}{A} \to \underbrace{\{\underset{Z}{B},\underset{Z}{C}\}}_{Y} \Rightarrow \underset{X}{A} \to \underset{Z}{B}$ e $\underset{X}{A} \to \underset{Z}{C}$

  *L'inclusione iniziale non è un vero e proprio assioma pero ogni dipendenza di F è comunque in F^A poiché F⊆F^A

$F^+ = F^A$

domenica 7 luglio 2024 18:41

F^+

Dato uno schema di relazione R e un insieme F di dip funz. su R
la chiusura di F (F^+) è l'insieme delle dip funz che sono soddisfatte da **ogni istanza legale** di R.
un'istanza r di R **soddisfa** la dipendenza funzionale X→Y se:
$\forall t1,t2 \in r,\ t1[X]=t2[X] \Rightarrow t1[Y]=t2[Y]$

Dato X,Y⊆R, si ha che:
$X \to Y \in F^A \Rightarrow Y \subseteq X^+$

Quindi F^+ contiene **tutte** le dipendenze possibili che **soddisfano** le **istanze legali** di R quindi contiene anche F **($F \subseteq F^+$)**

Dimostrazione $F^+ = F^A$:
Dobbiamo dimostrare sia che $F^+ \subseteq F^A$ e anche che $F^A \subseteq F^+$

Dimostriamo $F^A \subseteq F^+$ con il **principio di induzione** sul numero di assiomi applicati:

- **Caso Base** (n=0): Se $X \to Y \in F^A$ dopo aver applicato 0 assiomi allora la dipendenza è in F ($F \subseteq F^A$)
  quindi $X \to Y \in F$ allora $X \to Y \in F^+$
- **Ipotesi induttiva forte (k≤n):** ogni dip funz in F^A ottenuta applicando k≤n assiomi è anche in F^+
  $X \to Y \in F^A$ allora $X \to Y \in F^+$
- **Passo induttivo (n+1):** per dimostrare che dopo n+1 assiomi applicati $X \to Y \in F^A$ è anche in F^+ possiamo avere 3 casi in base all'assioma n+1 applicato:
  - Se è **riflessivo**: $X \to Y \in F^A$ implica che Y⊆X⊆R quindi per ogni **istanza legale r** di R si ha che:
    $\forall t1, t2 \in r$ (per ogni coppia di tuple di r), $t1[X]=t2[X] \Rightarrow$ che $t1[Y]=t2[Y]$ quindi $X \to Y \in F^+$
  - Se è **transitivo**: $\exists X \to Z, Z \to Y \in F^A$, ottenute con **k≤n assiomi** affinche $X \to Y \in F^A$
    poiché X→Z e Z→Y sono state ottenute con k≤n assiomi, per ipotesi induttiva, essi sono in F^+ quindi ogni istanza di R soddisfa X→Z e Z→Y quindi soddisfa anche X→Y e quindi $X \to Y \in F^+$
  - Se è **di aumento**:
    - $\exists V,W \subseteq R \mid \exists V \to W \in F^A$, ottenuta applicando k≤n assiomi (quindi
    - $\exists Z \subseteq R \mid X = VZ, Y = WZ$
      affinche si abbia che Z⊆R, $V \to W \Rightarrow VZ \to WZ = X \to Y \in F^A$

    poiché V→W è ottenuta con k assiomi essa è in F^+, e siccome Z→Z è sempre in F^+, allora ogni istanza di R soddisfa le dip. V→W e Z→Z in F^+, quindi soddisfa anche $VZ \to WZ \in F^+$
- in questo modo abbiamo dimostrato che per ogni possibile assioma applicato abbiamo che $F^A \subseteq F^+$

Dimostriamo ora che $F^+ \subseteq F^A$:
Supponiamo per assurdo che esista una dipendenza funzionale **$X \to Y \in F^+$** tale che **$X \to Y \notin F^A$**.
Useremo una particolare istanza legale di R per dimostrare che questa supposizione porta ad una **contraddizione**.

- Sia X⊆R e sia r un istanza dello schema R(X^+, R-X^+) tale che ha 2 tuple in questo modo
-

X^+			$R - X^+$		
A_1	...	A_i	A_j	...	A_n
1	...	1	1	...	1
1	...	1	0	...	0

- quindi:
  - $t1[X^+] = (1, \ldots, 1) = t2[X^+]$
  - $t1[R-X^+] = (1, \ldots, 1) \neq (0, \ldots, 0) = t2[R-X^+]$
- Per ogni $V \to W \in F$ si ha che:
  - se $V \subseteq (R-X^+)$ allora $t1[V] \neq t2[V]$ quindi la dip V→W è soddisfata per r
  - se $V \subseteq X^+$ allora $X \to V \in F^A$, siccome $X \to V \in F^A$ e $V \to W \in F^A$ per transitivita si ha che $X \to W \in F^A$ quindi $W \subseteq X^+$
    siccome $V,W \subseteq X^+$ si ha che $t1[V]=(1,\ldots,1)=t2[V]$ e $t1[W]=(1,\ldots,1)=t2[W]$ quindi r soddisfa $V \to W \in F$
- in entrambi i casi si ha che r soddisfa $V \to W \in F$ quindi r **è legale**

- qualsiasi dip $X \rightarrow Y \in F^+$ deve essere soddisfatta da qualsiasi istanza legale di R, inclusa r stessa
- poiché $X \subseteq X^+$, la dip X non può essere soddisfatta a vuoto quindi è soddisfatta da r solo se $Y \subseteq X^+$ cosi che t1[Y] = t2[Y]
- dunque si ha che $Y \subseteq X^+ \Rightarrow X \rightarrow Y \in F^A$ quindi $X \rightarrow Y \in F^+ \Rightarrow X \rightarrow Y \in F^A$, concludiamo che $F^+ \subseteq F^A$

Chiusura di X (X^+)

domenica 7 luglio 2024 20:15

Algoritmo 1: Algoritmo per la chiusura di un insieme di attributi

Sia R uno schema e sia F un insieme di dipendenze funzionali definite su R.

Dato un qualsiasi insieme di attributi $X \subseteq R$, è possibile calcolare tutti gli elementi appartenenti a X_F^+ tramite il seguente algoritmo:

```
def closureX(R: schema, F: set of dependencies, X: subset of R):
    Z := X
    S := { A | ∃ Y → V ∈ F, A ∈ V ⊆ R, Y ⊆ Z }
    while S ⊈ Z do:
        Z := X ∪ S
        S := { A | ∃ Y → V ∈ F, A ∈ V ⊆ R, Y ⊆ Z }
    X⁺ := Z
    return X⁺
```

trovo tutti gli attr A t.c. esista una dip Y→V in F per cui A∈V e Y⊆X

ripeto il ciclo finche S = Z

aggiorno Z unendo S a Z (qui dovrebbe essere Z U S non X U S)

cerco nuovamente tutti gli attr A t.c. esista una dip Y→V in F per cui A∈V e Y⊆X

Tale algoritmo viene eseguito in tempo polinomiale, ossia $O(n^k)$

Correttezza dell'algoritmo

Dimostriamo che dato un qualsiasi insieme di attr $X \subseteq R$, l'algoritmo restituisce X^+_F

- Indichiamo con Z_0 il val iniziale di Z ($Z_0 = X$) e con Z_i e S_i, $i \geq 1$, i val di Z e S **dopo** l'**i-esima** iterazione del ciclo while. è facile vedere che $Z_i \subseteq Z_{i+1}$ per ogni i
- Sia f tale che $S_f \subseteq Z_f$ (cioe Z_f è il val di Z quando **termina** il ciclo), proveremo che:
  $A \in Z_f$ se e solo se $A \in X^+$
- Dimostriamo per induzione che $Z_f \subseteq X^+$:
  - **Caso Base (i=0)**: $Z_0 = X$ e siccome $X \subseteq X^+$, si ha che $Z_0 = X^+$
  - **Ipotesi induttiva**: per ogni $i \geq 1$ si ha che $Z_i \subseteq X^+$
  - **Passo induttivo (i>0)**: Dato **$A \in Z_{i+1} - Z_i$ (quindi è stato aggiungo alla i+1 esima iterazione), $\exists Y \to V \in F$ tale che $Y \subseteq Z_i$, e $A \in V$.**
    Poiché $Y \subseteq Z_i$ per ipotesi induttiva si ha che $Z_i \subseteq X^+$ quindi $Y \subseteq X^+$ e quindi $X \to Y \in F^A$.
    Poiché $Y \to V$ e $X \to Y$ si ha che $X \to V \in F^A$, quindi $A \in V \subseteq X^+$ e quindi **$A \in X^+$**. Percio $Z_{i+1} \subseteq X^+$.
- Dimostriamo che $X^+ \subseteq Z_f$:
- Sia $A \in X^+$, dobbiamo dimostrare che $A \in Z_f$. Poiché $A \in X^+$, si ha $X \to A \in F^+$, quindi $X \to A$ deve **essere soddisfatta da ogni istanza legale di R**. Consideriamo la seguente istanza r di R:
-

Z_f			$R - Z_f$		
A_1	$\cdots$	A_i	A_j	$\cdots$	A_n
1	...	1	1	...	1
1	...	1	0	...	0

- Dimostriamo prima che r sia un istanza legale. Per ogni dip $V \to W \in F$:
  - Se $V \subseteq (R - Z_f)$ allora la dip è soddisfatta
  - Se $V \subseteq Z_f$ allora i valori delle tuple sono uguali. Se le due tuple avessero valori diversi su W (quindi la dipendenza non fosse soddisfatta), si avrebbe che Z_f non è il valore finale di Z.

Abbiamo assunto che Z_f è il val finale di Z quindi all'iterazione f non si può aggiungere più nulla di nuovo. Pero se avessi $V \subseteq Z_f$ e valori diversi su W **avrei qualche elemento di W che non è ancora in Z_f.** Quindi applicando l'iterazione f+1 potrei raccogliere in S i **nuovi elementi** tramite V→W e poi inserirli in Z_{f+1}. Pero facendo cosi vediamo che l'iterazione f non è quella finale bensi **l'iter f+1** e quindi Z_f non è il valore finale di Z ma questo è in **contraddizione** con la nostra costruzione dell'istanza.
Quindi V→W sarebbe soddisfatta anche quando le due tuple hanno valori uguali in V, quindi l'istanza è legale.

Chiusura di X^+ su G

lunedì 8 luglio 2024 00:27

Algoritmo 3: Calcolo di X_G^+ tramite F

Dato uno schema R con decomposizione $\rho = R_1, \ldots, R_k$, dato un insieme F di dipendenze funzionali su R e posto:

$$G := \bigcup_{i=0}^{k} \pi_{R_i}(F)$$

preso $X \subseteq R$, il seguente algoritmo calcola X_G^+ tramite F:

```
def X_G^+_with_F(R: schema, F: set of dependencies, X: set of attributes):
    Z := X
    S := ∅
    for i in range(1, k):
        S := S ∪ ((Z ∩ R_i)_F^+ ∩ R_i)
    while S ⊄ Z:
        Z := Z ∪ S
        for i in range(1, k):
            S := S ∪ ((Z ∩ R_i)_F^+ ∩ R_i)
    X_G^+ := Z
    return X_G^+
```

per ogni sottoschema in ρ aggiungiamo a S l'intersezione tra il sottoschema e la chiusura dell'intersezione di X e il sottoschema

finche S non è uguale a Z

aggiungiamo S a Z

e ricalcoliamo S e l'intersezione tra il sottoschema e la chiusura di Z e il sottoschema per ogni sottoschema

99mj

Chiusura di X^+ su G

Partendo da un insieme X, stiamo "ricostruendo" la sua **chiusura** dalle proiezioni di F che **compongono G**, e da lì iterando (ciclo) dall'insieme **G^+** (tramite la transitività). Attraverso l'intersezione con **R_i** ci assicuriamo che la dip. sia valida nel singolo sottoschema.

Consideriamo a turno ogni sottoschema in ρ e prendiamo le dipendenze nella chiusura F^+ ($(Z \cap R_i) \to A \in F^+$) perche **le dip nell'insieme G che ci interessa** sono incluse nelle proiezioni di F sui sottoschemi, che includono dip in F^+.
Quindi avremo gli attr che **dipendono funzionalmente da X** anche se appartengono a sottoschemi in cui X **non è incluso** perche dipendono da attr che sono nel sottoschema di X e dipendono da X, ma stanno in altri sottoschemi.

Dimostrazione. Indichiamo con Z_0 il val iniziale di Z ($Z_0 = X$) e con Z_i, $i >= 1$, il val di Z dopo la i-esima esecuzione dell'assegnazione Z=ZUS; vediamo che $Z_i \subseteq Z_{i+1}$ per ogni i. Sia Z_f il val finale di Z quando termina l'algoritmo, proveremo:

$A \in Z_f$ se e solo se $A \in X^+_G$

Dimostrazione per induzione che $Z_{i+1} \subseteq X^+_G$:

- **Caso Base (i=0):** $Z_0 = X$ e $X \subseteq X^+$, si ha che $Z_0 \subseteq X^+_G$
- **Ipotesi induttiva:** $Z_i \subseteq X^+_G$
- **Passo induttivo (i>0):** Sia $A \in Z_{i+1} - Z_i$ (quindi A è stata aggiunta nella i+1 esima iterazione, non è in Z_i)
  Allora deve esistere un indice j tale che **$A \in ((Z_i \cap R_j)^+_F \cap R_j)$**. Perche $A \in (Z_i \cap R_j)^+_F$ si ha $(Z_i \cap R_j)^+_F \to A \in F^+$.
  Poiché $(Z_i \cap R_j)^+_F \to A \in F^+$, $A \in R_j$ e $(Z_i \cap R_j) \subseteq R_j$ si ha, **per la definizione di G**, che $(Z_i \cap R_j)^+_F \to A \in G$.
  Poiché per **ipotesi induttiva** si ha che $X \to Z_i \in G^+$, per la regola della **decomposizione** si ha che $X \to (Z_i \cap R_j) \in G^+$ e quindi per **transitività** si ha che $X \to A \in G^+$, cioe $A \in X^+_G$, quindi $Z_{i+1} \subseteq X^+_G$.

La dimostrazione che $X^+_G \subseteq Z_{i+1}$ non è richiesta all'orale

Buona decomposizione

lunedì 8 luglio 2024 00:28

Buona Decomposizione

Sia R uno schema con decomposizione $\rho = R_1,\dots,R_k$ e F un insieme di dip. funz. su R
Definiamo ρ una **buona decomposizione** di R se:

- Ogni sottoschema $R_1,\dots,R_k$ in ρ è in **3NF**
- **ρ preserva F:** ρ permette di mantenere soddisfatte le dipendenze di F per ogni istanza legale r di R ricostruita con un **join naturale** tra tutte le istanze $r_1,\dots,r_k$ dei sottoschemi
- **Join senza Perdita**: ρ permette di ricostruire, **senza perdita di informazioni**, le tuple di ogni istanza legale r di R ricostruita con un **join naturale** tra tutte le istanze $r_1,\dots,r_k$ dei sottoschemi

Equivalenza tra due insiemi di dipendenze

Siano F e G due insiemi di dip. funz. F e G sono **equivalenti se $F^+= G^+$** (F e G non contengono le stesse dipendenze ma le loro **chiusure si**)
Poiché verificare l'uguaglianza di F^+ e G^+, cioè $F^+\subseteq G^+$ e $G^+\subseteq F^+$, richiede troppo tempo useremmo il seguente lemma per diminuire il tempo per la verifica delle chiusure.

Lemma: **Se $F\subseteq G^+$ allora $F^+\subseteq G^+$**. Dimostrazione che $f\in F^+ - F$ (è una dip. in F^+ che **non compare** in F):

- Denotiamo come $G \xrightarrow{A} F$ la derivazione di F partendo da G tramite gli assiomi di Armstrong
- Poiché $F\subseteq G^+$ abbiamo che $\forall X\rightarrow Y\in F$ si ha che $X\rightarrow Y\in G^+ = G^A$ quindi $G \xrightarrow{A} F$
- Siccome si ha che $F^A=F^+$ si ha che

$$F\subseteq G^+ \Rightarrow G \xrightarrow{A} F \xrightarrow{A} F^A = F^+ \Rightarrow F^+ \subseteq G^+$$

Join senza perdita

lunedì 8 luglio 2024 11:17

JSP

Sia R uno schema con decomposizione ρ = $R_1,\dots,R_k$ e F un insieme di dip. funz su R e r un istanza legale di R.

per ogni decomposizione R_i avremmo la decomposizione dell'istanza r_i che corrisponde ad una **proiezione di r** sugli attributi di R_i:

$r_i = \pi_{R_i}(r), i \in [1, k]$

Le singole proiezioni eliminano i duplicati che potrebbero essere generati da due tuple distinte aventi una porzione comune n el sottoschema R_i.

$m_\rho(r)$ è il join tra tutte le proiezioni di r sulle decomposizioni in ρ. quindi $m_\rho(r) = \pi_{R1}(r) \bowtie \dots \bowtie \pi_{Rk}(r)$, per ogni istanza legale di R.
posta questa istanza $m_\rho(r)$ si ha:

1) $r \subseteq m_\rho(r)$
2) $\pi_{Ri}(m_\rho(r)) = \pi_{Ri}(r), \forall R_i \in \rho$
3) $m_\rho(m_\rho(r)) = m_\rho(r)$

La dimostrazione di ciò non è chiesta all'orale pero si trova a pag. 67 dell'exyss e alla lezione 15 del materiale di Maria d e Marsico

Algoritmo 4: Controllo presenza del join senza perdita

Dato uno schema $R = A_1,\dots,A_n$ con decomposizione $\rho = R_1,\dots,R_k$ e un insieme F di dipendenze funzionali su R, presa l'istanza legale di r di R dove $\forall i \in [1,k], \forall j \in [1,n]$ si ha:

$$r_{i,j} = \begin{cases} "a" & \text{se } A_j \in R_i \\ "b_i" & \text{se } A_j \notin R_i \end{cases}$$

il seguente algoritmo determina se ρ presenta un join senza perdita.

```
def has_lossless_join(R: schema, F: set of dep., ρ:  decomposition):
    unchanged := False
    while not unchanged:          il ciclo finisce quando non è stata fatta più nessuna modifica nella tabella
        unchanged := True
        for X → Y ∈ F:            per ogni dipendenza in F
            for t1 in r:
                for t2 in r:      per ogni coppia di tuple nella tabella
                    if t1[X] == t2[X] && t1[Y] != t2[Y]:   se il determinante nelle due righe è
                                                           uguale ma il determinato no
                        unchanged = False
                        for Aj ∈ Y:          per ogni attributo nel determinante
                            if t1[Aj] == "a":      se almeno un val in quell'attributo nelle righe
                                                   di X ha il valore "a" cambio gli altri valori nelle
                                t2[Aj] := t1[Aj]   altre righe di X in "a"
                            else:
                                t1[Aj] := t2[Aj]   altrimenti se nessun attributo è "a" imposto
                                                   tutti gli elementi "bk" allo stesso pedice
    if ∃ t ∈ r | t = ("a", ..., "a"):
        return True           se alla fine almeno una riga nella tabella ha solo elementi
                              "a" possiamo dire che la decomposizione ha un join
    else:  return False       senza perdita
```

Algoritmo di Verifica di un JSP

Dato uno schema R = $A_1,\dots,A_n$ un insieme F di dip. funz su R e una decomposizione ρ = $R_1,\dots,R_k$ di R.
Costruiamo una tabella r che ha |R| colonne e |ρ| righe e ad ogni i∈[1,k] (righe, decomposizione in ρ) e j∈[1, n] (colonne, attr di R)
Ad ogni r_{ij} (quindi ogni casella della tabella) mettiamo:

- "a" se l'attributo $A_j \in R_i$ (quindi se l'attributo in quella colonna appartiene alla decomposizione in quella riga)
- "b_i" altrimenti (mettere il pedice i distingue gli elementi di una decomposizione per ogni decomposizione/riga)

Note sull'algoritmo (X = determinante, Y = determinato):

- Ogni volta che si trovano due tuple con lo stesso valore in X ma valori differenti in Y, quest'ultimo viene modificato, in modo che essi siano tutti uguali
- "a" **è prioritario**. "a" non può mai diventare "b" ma "b" può diventare "a"
- Se due tuple hanno stessi val in X ma differenti val in Y, se almeno una tupla ha "a" in Y, viene cambiato il "b" nell'altra tupla in "a"
- Invece se hanno stessi val in X ma differenti val in Y e nessuna tupla ha "a" in Y, viene cambiato il "b" in una delle due tuple in modo che abbiano lo stesso val "b" (quindi stesso pedice)
- Due valori sono considerati **uguali** se sono entrambi "a" o se hanno una "b" con lo stesso pedice ("a" = "a", "b_i" = "b_i", "a" ≠ "b_i", "b_i" ≠ "b_j")
- L'algoritmo termina quando tutte le coppie di tuple soddisfano le dipendenze di F
- Infine r diventa una **istanza legale** di R

- Terminato l'algoritmo se in r troviamo una tupla con solo valori "a" allora ρ presenta un **join senza perdita**, altrimenti no

Dimostrazione correttezza Algoritmo
Occorre dimostrare che ρ ha un join senza perdita (quindi $m_\rho(r) = r$ **per ogni** r legale) **se e solo se** quando l'algoritmo termina la tabella r ha una tupla con solo "a"
Dimostrazione **parte solo se:**

- Supponiamo **per assurdo che ρ abbia un join senza perdita** ($m_\rho(r) = r$) e che l'algoritmo termina con r che **non ha** una tupla con solo "a".
- La tabella r è un **istanza legale di R** (basta sostituire a "a" e "b" i valori presi dai domini dei corrispondenti attributi in modo che ad uno stesso simbolo venga sostituito lo stesso valore) in quanto l'algoritmo termina quando **non ci sono più violazioni delle dipendenze in F**.
- Poiché **ogni simbolo "a"** che compare nella **tabella iniziale non viene mai modificato** dall'algoritmo, per ogni $i \in [1, k]$, $\pi_{R_i}(r)$ contiene (**fin dall'inizio)** una tupla con tutte "a" (ottenuta **proiettando l'istanza r nella riga corrispondente al sottoschema R_i**)
- Pertanto $m_\rho(r)$ **contiene sicuramente una tupla con tutte "a"** e quindi $m_\rho(r) \neq r$ **(contraddizione)**

La dimostrazione per assurdo inizia assumendo che una proiezione ρ abbia un join senza perdita, ovvero che $m_\rho(r)=r$, e che l'algoritmo termini con r priva di tuple contenenti solo "a". La tabella r è un'istanza legale dello schema R, senza violazioni delle dipendenze in F, poiché il simbolo "a" non viene modificato dall'algoritmo. Pertanto, ogni proiezione $\pi_{R1}(r)$ contiene inizialmente una tupla con tutte "a". Di conseguenza, $m_\rho(r)$ deve contenere una tupla con tutte "a", implicando che $m_\rho(r) \neq r$, contraddicendo l'assunzione iniziale e dimostrando che r deve contenere una tupla con solo "a".

La dimostrazione della **parte se** non è richiesta all'orale pero si può trovare nelle dispense del corso

Algoritmo di decomposizione

lunedì 8 luglio 2024 13:11

Algoritmo 6: Algoritmo di decomposizione

Dato uno schema R e un insieme F una **copertura minimale** di dipendenze funzionali su R, il seguente algoritmo calcola in tempo polinomiale, dunque in $O(n^k)$, una decomposizione ρ di R tale che ogni sottoschema è in 3NF e ρ preserva F:

```
def decompose_R(R: set of attributes, F: minimal cover of dependencies):
    S, ρ := ∅
    for A ∈ R | ∄ X → B ∈ F, A ∈ X ∨ A = B:
        S := S ∪ A
    if S ≠ ∅:
        R := R - S
        ρ := ρ ∪ {S}
    if ∃ X → A ∈ F | X ∪ A = R:
        ρ := ρ ∪ {R}
    else:
        for X → A ∈ F:
            ρ := ρ ∪ {XA}
    return ρ
```

se trova attributi in R non coinvolti in alcuna dip li aggiunge a S

Se ha trovato attributi non coinvolti in alcuna dip allora toglie questi attributi da R e li aggiunge a ρ

se ci sono attributi che coprono completamente R allora li aggiunge a R e termina l'algoritmo

altrimenti cicla ogni dip in F e aggiunge la coppia determinante e determinato a ρ come sottoschema

Algoritmo di decomposizione

Sia R uno schema, F una **copertura minimale** di dip funz su R. L'algoritmo di decomposizione calcola una decomposizione ρ di R tale che:

- ogni schema in ρ è in **3NF**
- ρ **preserva F**

Dimostrazione:

Dimostriamo separatamente le due proprietà della decomposizione

ρ preserva F:

- Sia G l'unione tra le proiezioni di F sui sottoschemi in ρ.
- Quindi $\pi R_i(F) = \{X \rightarrow Y \mid X \rightarrow Y \in F^+ \text{ AND } XY \in R_i\}$
- $G := \bigcup_{i=0}^{k} \pi_{R_i}(F)$
- Si hanno due possibilità:
- Che **R-A→A∈F** quindi **R∈ρ** (R stesso è sottoschema)
- Per ogni dip X→A∈F si ha che **XA∈ρ** e quindi si ha che questa dip X→A **sarà sicuramente in G**
- Quindi $F \subseteq G$ e quindi $F^+ \subseteq G^+$.
- Poi per definizione $G \subseteq F^+$, quindi l'inclusione $G^+ \subseteq F^+$ è verificata
- Quindi F è equivalente a G

Ogni schema in ρ è in 3NF:

Analizziamo i diversi casi che si possono presentare:

1) Se S∈ρ, ogni attributo in S fa parte della chiave quindi S è in 3NF (perche se un attributo non è in alcuna dipendenza in F allora è in ogni chiave di R)
2) SE R∈ρ, **esiste** una dip in F **che coinvolge tutti gli attributi in R**.
F è una **copertura minimale** tale dipendenza avrà la forma **R-A→A.**
Poi siccome F è una **copertura minimale non esiste** una dip X→A in F^+ tale che **X è sottoinsieme unico di R-A**, quindi **R-A è chiave** di R e quindi anche **superchiave**.
Sia **Y→B** una qualsiasi dip in F:
    - se B=A allora, poiché F è copertura minimale, Y=R-A (cioè, Y è **superchiave**)
    - se B≠A allora **B∈R-A** e quindi B è **primo**
3) Se XA∈ρ, **poiché F è una copertura minimale**, **non esiste** una dip **X'→A in F^+ tale che X' è sottoinsieme di X**.
Quindi **X è chiave in XA**.
Sia **Y→B** una qualsiasi dip in F tale che **YB⊆XA**:
    - se B=A allora, poiché F è **copertura minimale**, Y=X (Y è **superchiave**)
    - se B≠A allora, B∈X e quindi B è **primo**

Nota: possiamo avere come risultato solo gli step **1+2** (R residuo) o **1+3** o solo **3**

Per avere anche un **join senza perdita**, **basta aggiungere un sottoschema contenente una chiave** a ρ.
Il teorema di ciò non è richiesto all'orale

Copertura minimale di F

lunedì 8 luglio 2024 12:13

Copertura Minimale

Sia F un insieme di dip. funz. Una **copertura minimale** di F è un insieme di dip. funz. G **equivalente** a F tale che:

- Per ogni dip X→A∈G, A è un singolo attributo (**singleton)**
- Per ogni dip X→A∈G, **non esiste** X' sottoinsieme unico di X tale che G è equivalente a **(G - {X→A}) U {X'→A}** ossia non è possibile determinare funzionalmente A tramite un **sottoinsieme** proprio di X (**rimozione determinanti ridondanti**)
- **Non esiste dip** X→A∈G tale che G è equivalente a **G - {X→A}**, ossia non è possibile determinare funzionalmente A tramite altre dipendenze (**rimozione dipendenze ridondanti**)

In sintesi la **copertura minimale** di F è un insieme equivalente G che soddisfa le seguenti condizioni:

1) Ogni dip. ha un **singolo attributo** come determinato
2) **Non ci sono determinanti ridondanti** (non si può ridurre un determinante senza perdita di informazioni)
3) **Non ci sono dipendenze ridondanti** (ogni dip è essenziale)

Si può ottenere una **copertura minimale di F** in 3 passi:

1) **Decomposizione di F:** Usando la regola della **decomposizione**, i determinati di ogni dip vanno ridotti a **singleton** (Es. AB→CD diventa AB→C, AB→D)
2) **Rimozione determinanti ridondanti:** Ogni dip $A_1A_2 \ldots A_{i-1}A_iA_{i+1} \ldots A_n \rightarrow A$ in G tale che G è equivalente a G - $\{A_1A_2 \ldots \mathbf{A_{i-1}A_iA_{i+1}} \ldots A_n\}$ U $\{A_1A_2 \ldots \mathbf{A_{i-1}A_{i+1}} \ldots A_n\}$ viene sostituita appunto da $A_1A_2 \ldots \mathbf{A_{i-1}A_{i+1}} \ldots A_n \rightarrow A$. Il passo termina quando **nessuna** dip può essere **ridotta**.
   Quindi controlliamo che per ogni dip esiste un sottoinsieme unico X' del determinante tale che la chiusura di X' su G determina Y e sia valida, quindi sostituiamo X→Y con X'→Y. (quindi per ogni dipendenza X→Y dobbiamo verificare che Y è in una chiusura di qualche insieme proprio di X)
   Es. AB→C: se A^+_G **=** ACD allora C **fa parte** della chiusura di A su G quindi possiamo già ridurre la dip a A→C
   ES. AB→C: se A^+_G = AD e B^+_G = B allora C **non fa parte delle chiusure** di alcun sottoinsieme unico di AB quindi non possiamo ridurre la dipendenza
3) **Rimozione dipendenze ridondanti:** Ogni dip X→A in G tale che G è equivalente a G - {X→A} (cioè che senza la dipendenza riusciamo comunque a determinare A) è **rimossa**. Per fare ciò possiamo controllare che per dip X→Y in G se $Y \in (X)^+_{G-\{X \rightarrow Y\}}$ allora la dipendenza è **rimossa**.
   Es. AB→C: se $(AB)^+_{G-\{AB \rightarrow C\}}$ = ABC allora C **appartiene alla chiusu**ra e possiamo **rimuovere** la dip AB→C
   Es. AB→C: se $(AB)^+_{G-\{AB \rightarrow C\}}$ = AB allora C **non appartiene alla chiusura** e **manteniamo** la dip AB→C

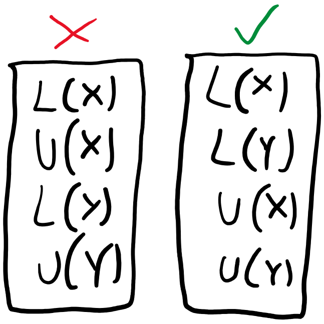
Le slide della professoressa spiegano questi passaggi in maniera più dettagliata e lunga quindi ho riassunto il processo

Protocollo a 2 Fasi

lunedì 8 luglio 2024 14:32

Per garantire la **serializzabilità** degli schedule esistono diversi **metodi:**
- Imporre dei **protocolli**, quindi delle **regole**, alle transazioni per garantire la serializzabilità di ogni schedule
- Usare i **timestamp** delle transazioni, cioè degli **identificatori** delle transazioni che vengono generate dal sistema per dare un valore alla durata della transazione e in base ai quali le operazioni delle transazioni possono essere **ordinate** per garantire la serializzabilità.

Un protocollo utile per garantire la serializzabilità di uno schedule è tramite l'implementazione del **protocollo di locking a due fasi**



Locking a 2 Fasi

Una transazione soddisfa il protocollo di locking a **2 fasi** se:
- Prima effettua **tutte le operazioni di lock** necessarie (fase di locking)
- **Poi tutte le operazioni di unlock** (fase di unlocking)

Se **ogni** transazione in uno schedule è **a 2 fasi** allora lo schedule è **serializzabile**.
Il protocollo del locking a **2 fasi** risolve anche il problema dell'**aggregato non corretto**, perché blocca tutti gli item necessari non permettendo alle altre transazioni di usarli finché non ha finito i calcoli necessari.

Praticamente poiché tutte le transazioni effettuano il lock su ogni item necessario allora lo schedule eseguirà le transazioni in modo **seriale**.

Time	T1	T2
t_0	BEGIN TRANSACTION	BEGIN TRANSACTION
t_1	Get Write Lock for A	
t_2	Get Write Lock for B	
t_3	Update A = A + 1	
t_4	Update B = B + 1	
t_5	UNLOCK A	
t_6	UNLOCK B	
t_7		Get Write Lock for A
t_8		Get Write Lock for B
t_9		Update A = A * 2
t_{10}		Update B = B * 2
t_{11}		UNLOCK A
t_{12}		UNLOCK B
t_{13}	COMMIT TRANSACTION	COMMIT TRANSACTION

Dati sporchi: dati scritti dalla transazione sulla BD prima che abbia raggiunto il punto di **commit**.

Per risolvere il problema della lettura di **dati sporchi** occorre che le transazioni obbediscano a regole più **restrittive del locking a 2 fasi.**

Locking a 2 Fasi Stretto

Una transazione soddisfa il protocollo di **locking a 2 fasi stretto** se:
1) **non scrive** sulla BD **fino a quando** non ha raggiunto il **punto di commit.**
   Se una transazione è **abortita** allora **non ha modificato** nessun item nella BD.
2) **non rilascia un lock finché non ha finito di scrivere** sulla BD.
   Se una transazione legge un item scritto da un'altra transazione, quest'ultima può essere abortita.

Quindi l'ordine in cui eseguire le operazioni è:
1) **Lock**
2) **Operazioni**
3) **Commit**
4) **Write**
5) **Unlock**

Punto di commit: il punto in cui la transazione ha:
- ottenuto **tutti i lock** necessari
- effettuato **tutti i calcoli** nell'area di lavoro

quindi la transazione non può essere più abortita nel caso in cui:
- La transazione esegue un op. **non corretta**
- Lo **scheduler** rileva un **deadlock**
- Lo **scheduler** fa abortire la transazione per garantire la serializzabilità (**timestamp**)

Deadlock: Uno stallo si verifica quando:
- **Ogni transazione** in T è in **attesa** di ottenere un lock su un item sul quale un'altra transazione in T mantiene il lock quindi
- Rimane **bloccata** e non **rilascia il lock**
- Può bloccare transazioni che non sono in T

Timestamp

lunedì 8 luglio 2024 15:08

Timestamp

Il **timestamp** identifica una transazione ed è assegnato ad essa dallo **scheduler** quando la transazione ha inizio
Esso può essere:
- il valore di un **contatore**
- l'**ora di inizio della transazione**

Il **timestamp di una transazione n è indicato come TS(Tn)**

Se il TS(T1) < TS(T2) allora vuol dire che la transazione T1 **inizia prima** della transazione T2 quindi, se le transazioni fossero eseguite in modo seriale, T1 viene **eseguita prima** di T2.

Uno schedule è **serializzabile** se è **equivalente** (quindi per **ogni dato modificato** producono **valori uguali**) **allo schedule seriale** in cui le transazioni **sono ordinate in base al TS.**
Quindi uno schedule è **serializzabile** se:
per ciascun item acceduto da più di una transazione, l'**ordine** con cui le transazioni **accedono** all'item è quello **imposto dai TS.**

Oltre al timestamp della transazione, ad **ogni item** X vengono associati **due timestamp:**
- **Read Timestamp di X** (Read_TS(X) o RTS(X)). Indica il **più grande** tra i timestamp di transizioni che hanno **letto** X con successo. (il massimo RTS(X))
- **Write Timestamp di X** (Write_TS(X) o WTS(X)). Indica il **più grande** tra i timestamp di transizioni che hanno **scritto** X con successo. (il massimo WTS(X))

Ogni volta che una T esegue un **write(X)/read(X)**, si confronta il **TS(T)** con il **WTS(X)/RTS(X) precedenti** per assicurarsi che l'ordine basato sui TS **non è violato.**

Se T cerca di eseguire **write(X)**:
1) Se **RTS(X) > TS(T)**, T viene **Rolledback**
2) Se **WTS(X) > TS(T)**, l'operazione di scrittura **non viene eseguita**
3) Altrimenti write(X) è **eseguita e WTS(X) = TS(T)**

Se T cerca di eseguire **read(X)**:
1) Se **WTS(X) > TS(T)**, T viene **Rolledback**
2) Se **WTS(X) <= TS(T)**, allora read(X) è **eseguita e se RTS(X) < TS(T) ⇒ RTS(X) = TS(T)**

Rollback: è un operazione eseguita su una transizione per risolvere il **deadlock** e altri motivi. Il rollback di T prevede:
1) La transazione è **abortita**
2) I suoi **effetti** sulla BD vengono **annullati**, **ripristina** i valori dei dati precedenti l'inizio della sua esecuzione
3) **Tutti i lock** mantenuti dalla transazione vengono **rilasciati**

T_1	T_2
read(X)	read(X)
X:=X+10	X:=X+5
write(X)	write(X)

TS(T1)=110
TS(T2)=100

TS(T2) < TS(T1)
ORDINE SERIALE
T2 T1

Seriale

T_1	T_2
	read(X)
	X:=X+5
	write(X)
read(X)	
X:=X+10	
write(X)	

T_1 legge X scritto da T_2

Schedule es.

T_1	T_2
	read(X)
read(X)	
	X:=X+5
X:=X+10	
write(X)	
	write(X)

Lo schedule è **serializzabile** solo se **non viene eseguita** la scrittura di X da parte di T_2

T_1 legge X prima che T_2 l'abbia scritto
non serializzabile

Esempio 2

SAPIENZA

Passo	Transazione	Operazione	RTS(X)	RTS(Y)	WTS(X)	WTS(Y)	X	Y
1	T3 TS(T3)=100	READ(Y)	0	100	0	0	x0	y0
2	T2 TS(T2)=110	READ(X)	110	100	0	0	x0	y0
3	T1 TS(T1)=120	READ(Y)	110	120	0	0	x0	y0
4	T1 TS(T1)=120	Y=Y-50	110	120	0	0	x0	y0
5	T3 TS(T3)=100	Y=Y-20	110	120	0	0	x0	y0
6	T2 TS(T2)=110	X=X+50	110	120	0	0	x0	y0
7	T1 TS(T1)=120	READ(X)	120	120	0	0	x0	y0
8	T1 TS(T1)=120	X = X +Y	120	120	0	0	x0	y0
9	T2 TS(T2)=110	[illegible]	120	120	0	0	x0	y0
10	T1 TS(T1)=120	WRITE(Y)	120	120	0	120	x0	y0-50
11	T3 TS(T3)=100	[illegible]	120	120	0	120	x0	y0-50
12	T1 TS(T1)=120	WRITE(X)	120	120	120	120	x0+ y0-50	y0-50

Al passo 9 la transazione T2 viene abortita. Dovrebbe eseguire la scrittura dell'item X, ma il suo timestamp è **minore** del timestamp della transazione **più giovane (con timestamp più alto)** che ha **letto** l'item X (RTS(X) = 120 > TS(T2) =110). Ciò significa che una transazione che **ha iniziato le** proprie operazioni **dopo T2 ha già letto** il valore dell'item X, mentre **secondo l'ordine di esecuzione** avrebbe dovuto leggere il valore di X **già modificato da T2**. Da qui la necessità di eseguire il roll back della transazione T2. Al passo 11 la transazione T3 viene abortita per lo stesso motivo. Dovrebbe eseguire la scrittura dell'item Y, ma il suo timestamp è **minore** del timestamp della transazione **più giovane (con timestamp più alto)** che ha **letto** l'item X (RTS(X) = 120 > TS(T3) =100).