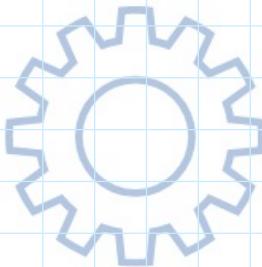


Indice cluster

L'indice è abbiano ad una memorizzazione in ordine lessicografico sulla base della chiave di ricerca del B-Tree.

Pot essere sparso solo se cluster.

Nel caso in cui ad ogni tupla siano associati più dati sparsi nelle pagine è possibile far partire i nodi foglia a tabella che indicizzano le pagine associate. Ne consegue che è dunque poi possibile



APPUNTI DI INGEGNERIA INFORMATICA

GAIA BERTOLINO

Il modello relazionale è fortemente strutturato ovvero i dati vanno codificati in una struttura ben precisa quale la tabella e ha inoltre una strutturazione dei dati tipicamente con variabile in quanto i campi sono sempre gli stessi. Un'altra caratteristica del modello relazionale è che lo schema e i dati sono ben distinti e lo schema è più piccolo dei dati.

Modello di dati semistrutturato

Rispetto a quello strutturato, la struttura può essere flessiva o assoluta ovvero con tutti i dati associati ad un concetto hanno la stessa forma. Tuttavia, la struttura di un dato può varcare nel tempo e lo schema può essere codificato nei dati ovvero associ ad una serie di dati uno schema e ad altri altri schemi ovvero crea schemi locali ai gruppi di dati. In aggiunta, uno schema può avere dimensione paragonabile ai dati.

Lo standard per il linguaggio semistrutturato è l'**XHTML** (Extensible Markup Language o linguaggio di delimitatori estendibili) e ha come simile il codice HTML che hanno in comune l'uso di delimitatori.

Per **DELIMITATORE** si intendono dei comandi che definiscono aree di proprietà o informazioni degli oggetti.

I file XML sono due documenti testuali dove i delimitatori servono a specificare il significato delle strutture specificate.

es. <biblioteca> TAG o DELIMITATORE

<libro anno = '1980' prezzo = '29€' ATTRIBUTI

<titolo> Divina commedia </titolo>

<autore> Dante </autore>

</libro>

<libro> ELEMENTI

I tag possono anche essere messi come attributi ma troppi attributi diminuiscono la leggibilità

```

< ISBN > 188 19 25 07 </ ISBN >
< autori >
    < autore > Gutter </ autore >
    < autore > Locentini </ autore >
< / autori >
< / biblioteca >

```

IL Linguaggio è detto extensible perché, rispetto a HTML, i markup non sono definiti nativamente nel linguaggio ma sono creati ad hoc in base alle caratteristiche dell'oggetto.

La peculiarità del linguaggio è che essendo scritta in linguaggio naturale esso è versatile e facilmente diffondibile e comprensibile senza difficoltà. Inoltre, anche lo schema, i dati e i Tag possono cambiare

Un file XML è spesso accompagnato da una guida che specifica ad una macchina come dover leggere il file e spesso ciò viene fatto tramite un DTD (Data Type Definition). Essa è definita **guida dei dati**.

es. <!ELEMENT biblioteca (libro*)>

GATA PER BESO INO OR

<!ELEMENT libro (ISBN?, titolo, autori | autore, casa editrice?)>

<!ELEMENT autori (autore+)>

<!ELEMENT ISBN #PCDATA> **parcele character data**

<!ATTLIST libro identificatore ID Univoco per gli oggetti
nuove attributo possibile

prezzo CDATA stringa di caratteri (che non è sottoposta a parsing)

curatrici IDREF > chiave esterna (riguarda ad un altro tag)

↳ IDREFS → più curatrici

Gli elementi definiscono l'ordine di presenza nello schema

Inoltre, un file XML può essere visto come un albero:

es. schema:

<biblioteca>

```
< libro ID = 'L1' CasaEd = "Ce1">
  < titolo > TL1 </titolo>
  < autori > < AL1 </autori>
</ libro >

< libro ID = 'L2' CasaEd = "Ce1">
  < titolo > TL2 </titolo>
  < autori >
    < autore > AL2 </autore>
    < autore > AL3 </autore>
  </ autori >
  < anno Ed > 2000 </anno Ed >
</ libro >

<CasaEd ID = "Ce1">
  nome CasaEd:
</CasaEd >
```

</biblioteca >

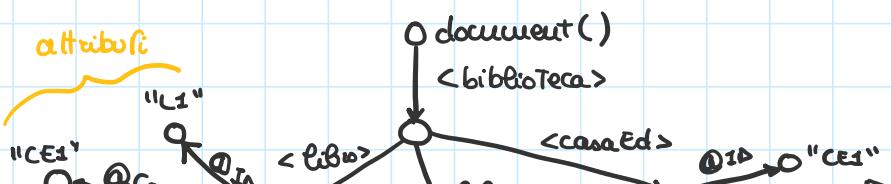
APPUNTI DI INGEGNERIA INFORMATICA

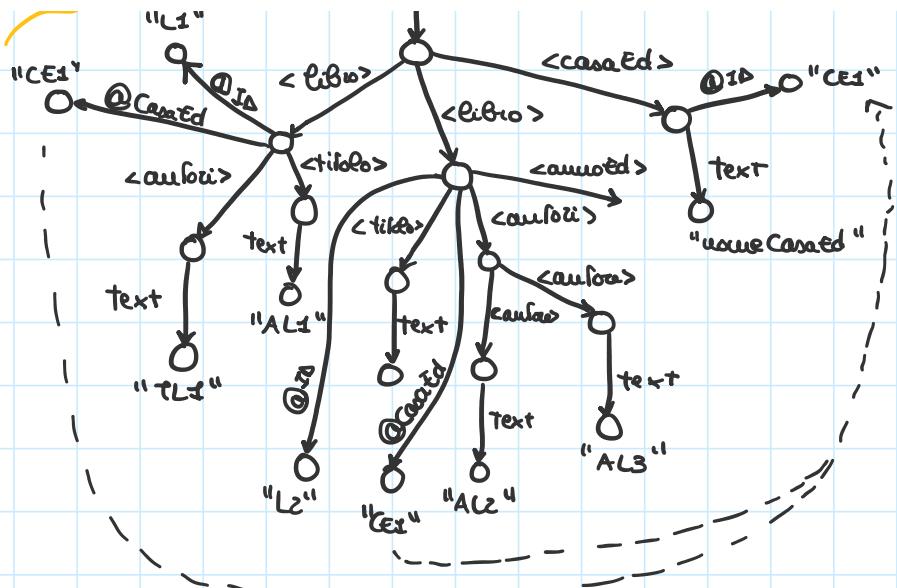
GAIA BERTOLINO

Guida dei dati:

```
<!ELEMENT biblioteca (libro+, casaEd+)>
<!ELEMENT libro (titolo, autori | autow, annoEd?)>
<!ELEMENT autori (autore+)>
<!ATTLIST libro ID ID
          CasaEd IDREF>
<!ELEMENT casaEd #PCDATA>
<!ATTLIST casaEd ID ID>
```

Albero:



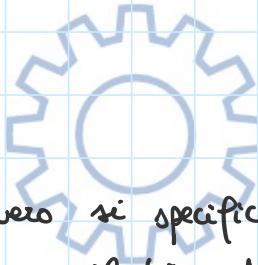


L'albero ottenuto è anche **diretto** perché è facilmente individuabile se nodo radice e i nodi figli.

LINGUAGGIO DI ESTRAZIONE

Linguaggio XPath

Esso esprime cammini sul quale si specifica il cammino che bisogna percorrere per leggere il dato di interesse e tali cammini sono espressi tramite i cosiddetti **assi navigazionali**.



INFORMATICA

unicaRadice ("unicaFile") / child::titolo

es. Ricercare i titoli dei libri:

[document ("unicaFile") / child::biblioteca / child::libro /
child::titolo]

Risultato

< titolo > TL1 </titolo >

< titolo > TL2 </titolo >

alternativa

document ("unicaFile") / descendant :: titolo

in questo caso funziona poiché il tag titolo fa riferimento solo ai libri.

solo ai libri.

Il comando descendant è comodo nel caso in cui i tag ricercati si trovano su livelli diversi.

↓ alternativa

document ("xmlxul") / descendant:: libro / child:: titolo

Riassunto dei comandi:

child / descendant / ancestor / parent / sibling

Non scrivere niente davanti: il tag sottointende il comando child. La doppia classe indica invece il comando descendant.

Specificare alla fine /text() restituisce il valore senza tag.

es. document () // libro / autore

<autore> AL1 </autore>

document () // libro / autore /text()

AL1

APPUNTI DI INGEGNERIA INFORMATICA

Per restituire un attributo si usa l'@.

es. document () // libro / @ id

L1

L2

Per accedere specificamente ad un dato si può utilizzare un **filtro di selezione** che si scrive attraverso delle parentesi quadre. Si possono applicare più filtri sovrapponendoli a cascata.

parte da dove sono arrivato

es. document () // libro [. / annoPub /text() = 2000]

filtro di selezione

<libro id = "L2" casatd = "LEI">

<titolo> TL2 </titolo>

< autori >

:

Tuttavia in questo modo si restituiscono tutti i dati relativi al libro. Per ottenere solo il tag di intestazione usa il seguente codice:

es. document() // libro [./annoPub /text() = 2000] /titolo



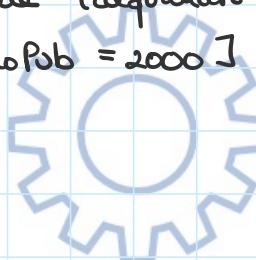
<titolo> TLC </titolo>

Il linguaggio XPath serve solo a trasformare valori.

Anche una query senza text() o altre funzioni di trasformazione funziona poiché ogni nodo viene trasformato secondo il proprio tipo.

es. document() // libro [./annoPub = 2000] /titolo

<titolo> TLC </titolo>



Infatti, quando vi è un valore di confronto avviene un'automatica conversione nei rispettivi tipi.

Nel caso in cui siano presenti più nodi a sinistra della confrontazione allora essa è verificata se esiste almeno uno dei nodi che fa verifica, qualsiasi sia la condizione.

es. [10 50 60] != [10] TRUE

[10 50 60] = [10] TRUE

[6 7 8] = [8 9 10] TRUE

Per prendere tutti i discendenti è invece possibile usare l'asterisco prima dell'elemento. Esso è detto simbolo di wildcard.

es. document() // * [. /annoPub] /titolo

Linguaggio Xquery

Linguaggio Xquery

DTD del documento

```
<!ELEMENT bib (book*)>
<!ELEMENT book (title, (author+ | editor+), publisher, price)>
<!ATTLIST book year CDATA #REQUIRED>
<!ELEMENT author (last, first)>
<!ELEMENT editor (last, first, affiliation)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT last (#PCDATA)>
<!ELEMENT first (#PCDATA)>
<!ELEMENT affiliation (#PCDATA)>
<!ELEMENT publisher (#PCDATA)>
<!ELEMENT price (#PCDATA)>
```

Documento di esempio

```
<bib>
  <book year="1994">
    <title>TCP/IP Illustrated</title>
    <author><last>Stevens</last><first>W.</first></author>
    <publisher>Addison-Wesley</publisher>
    <price>65.95</price>
  </book>

  <book year="1992">
    <title>Advanced Programming in the Unix environment</title>
    <author><last>Stevens</last><first>W.</first></author>
    <publisher>Addison-Wesley</publisher>
    <price>65.95</price>
  </book>

  <book year="2000">
    <title>Data on the Web</title>
    <author><last>Abiteboul</last><first>Serge</first></author>
    <author><last>Buneman</last><first>Peter</first></author>
    <author><last>Suciu</last><first>Dan</first></author>
    <publisher>Morgan Kaufmann Publishers</publisher>
    <price>39.95</price>
  </book>

  <book year="1999">
    <title>The Economics of Technology and Content for Digital TV</title>
    <editor>
      <last>Gerbarg</last><first>Darcy</first>
      <affiliation>CITI</affiliation>
    </editor>
    <publisher>Kluwer Academic Publishers</publisher>
    <price>129.95</price>
  </book>
</bib>
```

GAIA BERTOLINO

Retorno → • ritorna espansione Xpath

es. ritorna document(...)//publisher

↓
esiste di elementi publisher

<publisher>...</publisher>

⋮
<publisher>...</publisher>

es. ritorna <risultato> {document(...)//publisher} ↓

↓
elemento risultato con all'interno una lista di publisher

<risultato>

<publisher>...</publisher>

⋮
<publisher>...</publisher>

</risultato>

es. ritorna <risultato> document(...)//publisher

</risultato >

es. return <risultato> document(...) //publisher

↓
elemento xml con all'interno una stringa

<risultato>

document(...) //publisher

</risultato> viene tolto come stringa

Foc →

- for \$p in espressione xpath ritorna

es. for \$p in document(...)//publisher

ritorna <casaEditrice> { \$p/text() }

↓ el. xml con lista dei tag publisher

es. <risultato>

{ for \$p in document(...)//publisher

return <casaEditrice>

{ \$p/text() }

</casaEditrice>

}

</risultato>

↓

<risultato>

APPUNTI DI INGEGNERIA

INFORMATICA

GAIA BERTOLINO

<casaEditrice>

:

</risultato>

Where → Verifica una condizione

es. Risituire i titoli dei libri pubblicati

dopo il 2000:

<risultato>

{ for \$b in document(...)//book

where \$b/@year > "2000"

return <titolo> { \$b/title/text() }

</titolo> }

```
return <titolo> {$b /title/text()}  
</titolo>  
</risultato>
```



```
<risultato>  
<titolo> ...  
:  
</risultato>
```

La where è traducibile tramite il filtro xpath:

```
<risultato>  
{ for $b in document (...) // book [@year > 2000]  
  return <titolo> {$b /title/text()} </titolo> }  
</risultato>
```

Let → • Describe un assegnamento new (lexico)

es. for \$b in document (...) // book

let \$a := \$b // autori

return <libro>

GAIA BERTOLINO

```
<titolo> {$b /title/text()} </titolo>  
<autori> {$a </autori> }
```

</libro>



<libro>

<titolo> T1 </titolo>

<autori>

<autori> ...

:</autori>

</libro>

es. for \$b in document (...) //book

let \$a := \$b // autori

return <libro>

<titolo> {\$b / title / text()} </titolo>

<autori>

{for \$x in \$a}

return <cognome Autore> {\$x / last / text()}

</cognome Autore>

</autori>

</libro>



<libro>

<titolo> T1 </titolo>

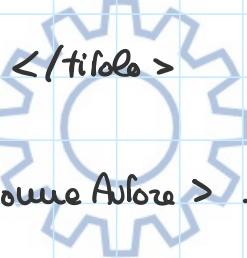
<autori>

<cognome Autore> ...

:

</autori>

</libro>



GAIA BERTOLINO

es. for \$b in document (...) //book

let \$a := \$b // autori

where count (\$a) >= 2

return <libro>

<titolo> {\$b / title / text()} </titolo>

</libro>



<libro>

<titolo> T1 </titolo>

:

</libro>

Esempi:

- 1) Restituire un documento la cui radice ha un attributo numTitoli che contiene il numero di elementi <libro> presenti nel documento. La radice contiene inoltre una lista di elementi titolo, uno per ogni libro.

```
<!ELEMENT risultato (titolo+)>
<!ATTLIST risultato numTitoli CDATA>
<!ELEMENT titolo (#PCDATA)>

<risultato numTitoli="{count( document(...) //book) }">
{
    for $t in document(...) //title
    return <titolo>
        { $t/text() }
    </titolo>
}
</risultato>

<risultato numTitoli="3">
    <titolo> ... </titolo>
    <titolo> ... </titolo>
    <titolo> ... </titolo>
</risultato>
```



- 2) Restituire i titoli dei libri scritti da Dan Suciu:

```
<result>
{
    for $b in document(...) //book
    where $b/author/last="Suciu" and $b/author/first="Dan"
    return $b/title
}
</result>
```

↳ es. risultato sbagliato se esiste un libro scritto da
Pasqualino Suciu e Dan Peresce

```
<result>
{
    for $b in document(...) //book[ author[last=Suciu][first=Dan] ]
    return $b/title
}
</result>
```

Oppure

```

<result>
{   for $b in document(...) //book
    for $a in $b/author
      where $a/first="Dan" and $a/last="suciu"
      return $b/title
}
</result>

```

3)

RISULTATO:

```

<!ELEMENT result (publisher+)>
<!ELEMENT publisher (name, listOfBooks)>
<!ELEMENT listOfBooks (title+)>
<!ELEMENT title (#PCDATA )>
<!ELEMENT name (#PCDATA )>

<result>
{
  for $p in document()//publisher
  return <publisher>
    <name> {$p/text()} </name>
    <listOfBooks>
      { for $t in document(...) //book[publisher=$p]/title
        return $t
      }
    </listOfBooks>
  </publisher>
}
</result>

```

APPUNTI DI INGEGNERIA INFORMATICA

oppure

```

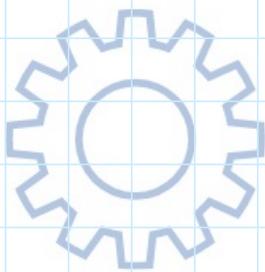
<result>
{
  for $p in document()//publisher
  return <publisher>
    <name> {$p/text()} </name>
    <listOfBooks>
      { document(...) //book[publisher=$p]/title }
    </listOfBooks>
  </publisher>
}
</result>

<result>
{
  for $p in distinct-values(document()//publisher)
  return <publisher>
    <name> {$p} </name>
    <listOfBooks>
      { document(...) //book[publisher=$p]/title }
    </listOfBooks>
  </publisher>
}
</result>

```

```
}
```

```
</result>
```



APPUNTI DI INGEGNERIA INFORMATICA

GAIA BERTOLINO

Gestione delle transazioni

domenica 30 gennaio 2022 22:53

Transazione → programma in esecuzione che accede alla base di dati per accedere ai dati o modificarli.

La differenza coi singoli comandi di query o simili è che una transazione è un insieme (blocco di operazioni) che necessariamente devono essere eseguite insieme.

Un esempio è l'inserimento di una fattura di un oggetto che non esiste nel database; in questo caso bisogna che sia possibile inserire sia oggetto che fattura e se uno dei due inserimenti fallisce allora non riesce nemmeno il resto e avviene un tipistico alla situazione precedente alle operazioni (**ROLL BACK**).

L'istruzione di **COMMIT** è invece una operazione di definizione finale delle modifiche apportate al database.

I sistemi che gestiscono le transazioni sono dette **transazionali** e hanno quattro proprietà descritte dalla sigla ACID:

- Atomicità → l'insieme di statement è eseguito solo se tutti gli statement possono essere singolarmente eseguiti.
- Consistenza → una transazione agisce su uno stato consistente e lo lascia tale ovvero i vincoli non vengono violati / modificati
- Isolamento → se vi sono più transazioni concorrenti, esse vengono eseguite in maniera isolata ovvero viene gestita la concorrenza delle modifiche per evitare condizioni di conflitto.
↓
Venne garantito tramite la serializzabilità

La modalità più semplice di alternanza è la banale esecuzione delle operazioni in serie detta **esecuzione seriale** o **scheduling seriale**.

Tuttavia si usa una interruzione con alteranza dei processi che permette di avere del "tempo" per parallelizzare operazioni che non entreranno in conflitto.

Uno scheduler è un componente del DBMS che genera schedule.

che permette di avere del "tempo" per parallelizzare operazioni che non entreranno in conflitto.

SCHEDULE → assegnazione di un numero ordinale ad ogni operazione di un set di operazioni. Il componente che si occupa dello scheduling è detto scheduler.

Per scheduler serializzabile si intende scheduler equivalenti a quelli seriali ossia portano il DBMS allo stesso stato da cui sarebbe stato portato dallo scheduler seriali.

Uno scheduler è serializzabile se esiste una S' alle stesse transazioni di S tale che $S \equiv S'$

- Durabilità → è modifiche, dopo un commit, sono definitive

Gli scheduler sono seriali se per ogni transazione le operazioni sono interruzze da altre operazioni. È serializzabilità se è equivalente ad uno serial.



a. serial

T ₁	T ₂
read(A)	read(B)
write(A)	write(B)
read(B)	read(A)
commit	commit

una serial una serializzabile

T ₁	T ₂
read(A)	
	read(A)
	write(B)
	read(B)
	read(A)

Gli scheduler solitamente generano degli schedule definiti **conflict-serializable** ovvero degli schedule ottenuti da uno **serializable** invertendo operazioni non in conflitto.

Due operazioni sono in conflitto quando agiscono sulla stessa risorsa e almeno una di loro è una write.

→ Formalmente:

Una schedula S è conflict-serializable se è conflict-equivalent ad una schedula seriale S' ($S \equiv S'$).

Una schedula $S \equiv S'$ è conflict-equivalent se esiste una sequenza di scambi di operazioni von in conflitto tramite la quale si ottiene S' da S .

Una schedula conflict-serializable è serializzabile.

Utilizzando una rappresentazione a grafo con archi che indicano i passaggi fra transazioni, ogni arco rappresenta un'operazione in conflitto e una schedula sarà conflict-serializable se non presenta cicli.

Ordinamento Topologico → è l'algoritmo di individuazione di una schedula conflict-serializable tale che ad ogni iterazione si riduce il grado di ingresso zero del grafo che rappresenta le operazioni in conflitto.

APPUNTI DI INGEGNERIA

INFORMATICA

GAIA BERTOLINO

Una schedula S è VS se è view-equivalent ad una schedula seriale S' ovvero ($S \equiv S'$)

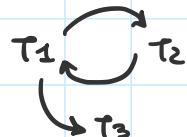
Due schedule S e S' sono view equivalent se

- 1) partono allo stesso modo ovvero leggono gli stessi dati:
A risorsa X , le transazioni che leggono il valore iniziale di X in S leggono tale valore anche in S' e viceversa
- 2) "scorrono" allo stesso modo:
A risorsa X , se T_i legge il valore di X scritto da T_j in S allora T_i legge il valore di X scritto da T_j in S' e viceversa
- 3) terminano allo stesso modo:
A risorsa X , la transazione (se esiste) che esprime il valore

A risorsa X , la transazione (se esiste) che esprime il valore finale di X in S scrive il valore finale di X in S' e viceversa.

es.	T_1	T_2	T_3
	read(A)		
		write(A)	
	write(A)		
		write(A)	
			write(A)

Non è conflict-serializable



Scrittura creca

è view-equivalent a T_1, T_2, T_3 (che è serializzabile)
e quindi è anche view-serializable

Attenzione! CS C VS C S

es.	T_1	T_2	T_3
	read(A)	write(A)	
	write(A)		
		write(A)	
			write(A)

Non è view-equivalent
a T_1, T_2, T_3 poiché viola
grado la prima proprietà di
lettura

Problema della recuperabilità (recoverability):

Tale problema riguarda alla possibilità di riuscire a stabilire sempre uno stato consistente (vor per forza quello iniziale) una volta che compie azioni rollback possibili.

I problemi di rollback possono essere evitati tramite una condizione sufficiente:

per ogni coppia di transazioni tale che una legge un valore scritto da un'altra, deve valere che la seconda di chi scrive deve precedere quella di chi legge.

→ tali regole non dicono nulla su conflict-serializability

procedere quella di chi legge.

→ tale proprietà è applicabile ai conflict-serializable

e..	T ₁	T ₂	T ₃
	read(A) write(A)		
		read(A) write(A)	
			read(A) write(A)
	committ		
		committ	
			committ

La condizione è verificata

ma si crea un rollback a cascata.

Per evitare abbandon a cascata

si cerca di avere delle

situazioni di **cascadeless**

Per realizzare una cascadeless situation devo fare in modo
che T₁ committi le sue operazioni prima di T₂ ovvero:
per ogni coppia di transazioni tale che la seconda legge un
valore scritto dalla prima, la prima deve eseguire la
committ prima della seconda.

APPUNTI DI INGEGNERIA INFORMATICA

GAIA BERTOLINO

Riassunto:

L'isolamento viene garantito gestendo la serializzabilità. Purtroppo il caso generale di serializzabilità, ossia generare schedule che siano equivalenti ad uno schdule serial, è complesso e dunque nei DBMS si garantiscono delle condizioni sufficienti di serializzabilità; ad esempio: DBMS usano il concetto di conflict serializable per cui potrebbero esistere degli schedule che non vengono mai presi in considerazione perché ottenibili richiedendo politiche di scheduling troppo onerose.

Oltre alla serializzabilità vi sono altri aspetti da garantire quali la recuperabilità ovvero garantire che in caso di errori sia sempre possibile recuperare uno stato consistente e l'assenza di cascade che significa garantire che nel caso di errori si limiti il più possibile il numero di transazioni che vengono fatte abbandonare.

Per verificare che una schedula sia serializzabile basta prendere il graf delle precedenze e verificare se è circolare o meno.

APPUNTI DI INGEGNERIA

La generazione di schedule fino ad uno conflict-serializable è onerosa. Dunque si prediligono lock di lettura e scrittura realizzati tramite:

- Lock-X (exclusive) → Lock di accesso esclusivo fra risorse (ottimo per la scrittura).

Quando è attivo tutto le richieste di Lock-X e di Lock-S vengono messe in attesa.

- Lock-S (shared) → Lock di accesso condiviso fra risorse (ottimo per la lettura).

Quando è attivo tutto le richieste di Lock-X vengono messe in attesa mentre tutte quelle di Lock-S vengono accettate (granted).

Dunque in questo caso non si parla di true sharing

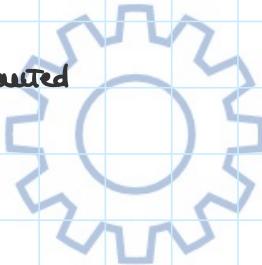
es.

T_1	T_2
read(A)	
$A = A + 10$	
	read(A)
	$A = A + 20$
write(A)	
	write(A)

Non è né serializzabile
né conflict-serializable.

↓

T_1	T_2
lock-x(A)	→ granted
read(A)	
$A = A + 10$	
	lock-x(A) → not granted = wait
write(A)	
unlock-x(A)	
	lock-x(A) → granted
	read(A)
	$A = A + 20$
	write(A)
	unlock-x(A)



Protocollo 2PL (2 phase locking)

Usando i lucchetti non si ottiene sempre l'isolamento.

Per risolvere tale problema, anziché bloccare tutte le variabili utilizzate da una transazione, faccio in modo che non rilasci lucchetti se prima non richiede e ottiene il lucchetto di altre variabili che deve utilizzare.

es.

T_1	T_2
lock-x(A)	
read(A)	
$A = A + 20$	
write(A)	
unlock(A)	
	lock(A)
	:
	lock(s)
	lock(B)
	:

Il protocollo individua due fasi:

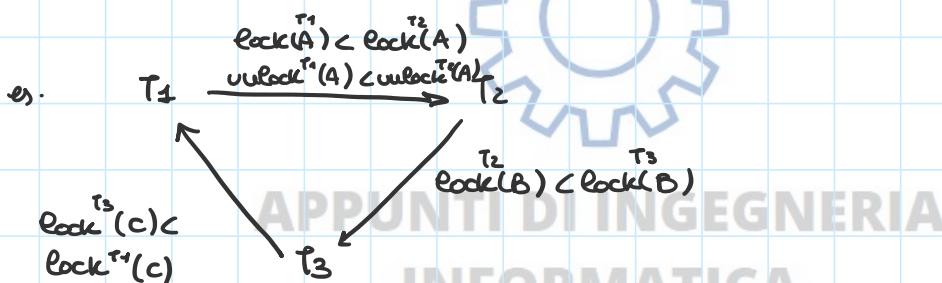
- Growing → crescita del numero di lock richiesti
- Shrinking → contrazione ovvero cessione dei lock

→ dopo la prima unlock le transizioni non possono fare più lock ma solo unlock

Questo protocollo fa in modo che gli scheduli che lo seguono siano sempre conflict-serializzabili.

Dimostrazione:

Supponendo di avere T_1 , T_2 e T_3 aderenti alla 2PL si ragiona per assurdo ovvero che S non sia conflict-serializzabile. Allora, il grafo delle precedenze deve essere circolare; tuttavia la presenza di cicli è incompatibile dell'acquisizione con precedenza dei lock con successivo unlock!



Ne consegue che $unlock^{T1}(A) < lock^{T3}(A)$ che è una CONTRADDIZIONE

Tuttavia il 2PL non garantisce la recuperabilità poiché non dà indicazioni sulla commit.

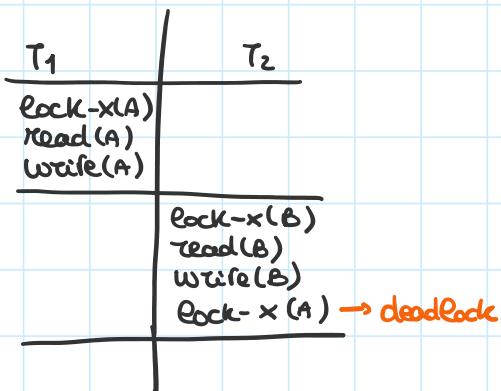
Per evitare a tale problema si utilizza la variante **Strict 2PL** in cui non vengono mai rilasciati i lucchetti esclusivi se non da una commit.

Nello **strong strict 2PL** tutti i lock vengono rilasciati con una commit (non sono cioè vengono fatte delle unlock).

In questi protocolli si può verificare **starvation** nel caso in cui vi sia una carenza di acquisizioni di lock di lettura. Ciò si può risolvere con meccanismi di monitor.

caccata di acquisizioni di lock di lettura. Ciò si può risolvere con meccanismi di aging.

Inoltre, si può verificare **deadlock**:



Una transazione che è bloccata temporalmente da troppo viene fatta volentieri fallire e viene fatto rollback.

Il comando **set isolation** serve a definire qual è il livello di isolamento ovvero di complessità delle operazioni eseguite per mantenere l'isolamento:

- serializable
- repeatable read
- read committed
- read uncommitted → messo in calo

APPUNTI DI INGEGNERIA INFORMATICA GAIA BERTOLINO

Esercitazione 2

domenica 30 gennaio 2022 23:01

21 Ottobre 2021

Corso di Laurea in Ingegneria Informatica (DM 270) Esame di Basi di Dati 4 settembre 2018

Esercizio 1: Progettazione concettuale e logica di una Base di Dati (40 minuti)

a) Si progetti una base di dati di parziale supporto alla gestione di un'impresa di pulizie. Occorre tenere traccia dei clienti dell'impresa, i quali sono identificati dalla partita IVA e sono anche caratterizzati dal nome, dalla residenza, e dalla lista dei contratti di opere di pulizia con essi stipulati. Ciascun contratto, oltre che dal cliente, è caratterizzato da un numero progressivo e dalla data di stipula. Il numero di un contratto lo distingue da quelli stipulati nella stessa data. L'insieme dei contratti è partitionato nei due sottoinsiemi formati da contratti relativi a singole prestazioni e quelli relativi a prestazioni periodiche (in seguito indicati come contratti singoli e periodici). Per ciascun contratto periodico occorre indicare la periodicità dell'intervento di pulizia e la lista dei locali presso i quali gli interventi vanno svolti. Ogni contratto singolo si riferisce invece ad un unico locale, di cui occorre tenere traccia.

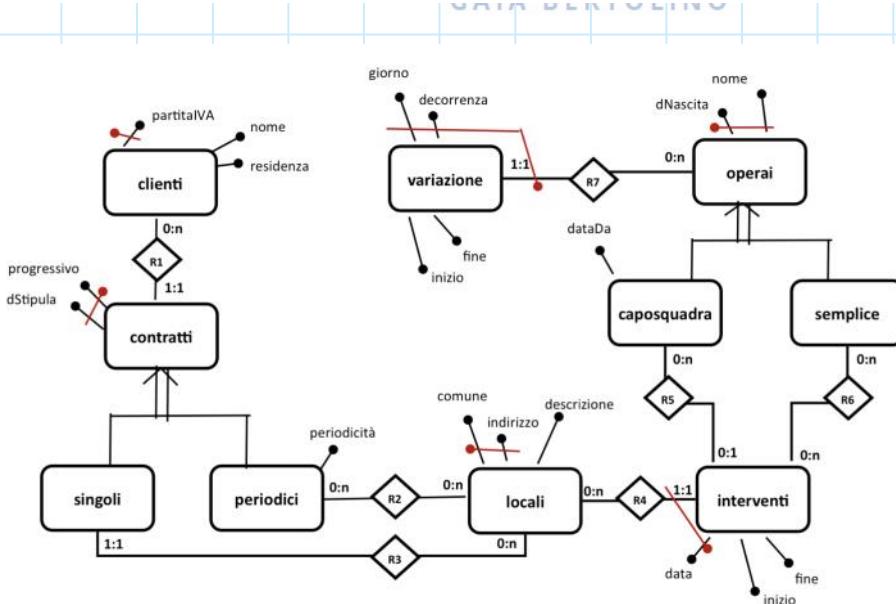
I locali sono caratterizzati dal comune in cui si trovano, dall'indirizzo, e da una descrizione. Non esistono più locali ubicati allo stesso indirizzo dello stesso comune. Si noti che un locale può essere associato ad un numero qualunque di contratti periodici e ad un numero qualunque di contratti singoli. Per ogni locale va inoltre indicata la lista degli interventi di pulizia presso di esso effettuati. Ogni intervento, oltre che dal locale presso il quale è avvenuto, è caratterizzato dalla data di effettuazione, dall'ora di inizio e da quella di fine. In una stessa data, esiste al più un intervento effettuato presso ciascun locale.

Oltre che di quanto detto sopra, è necessario tenere traccia degli operai dell'impresa, i quali sono identificati dalla coppia <nome, data di nascita> e sono di esattamente uno dei seguenti tipi: operaio semplice e caposquadra. Per i capisquadra, occorre indicare la data a partire dalla quale hanno rivestito tale incarico. Sia gli operai semplici che i capisquadra partecipano agli interventi di pulizia, e di tale partecipazione occorre tenere traccia, secondo quanto indicato nel seguente diagramma.

In particolare, un capisquadra può partecipare ad ogni intervento, mentre un operaio semplice può partecipare ad un numero qualunque di interventi, ed al più un intervento per ogni capisquadra.

Per ogni operaio (indipendentemente dal tipo) va inoltre rappresentata la lista delle variazioni dei turni di lavoro. Ogni variazione è caratterizzata dall'operaio a cui si riferisce, dal giorno della settimana del turno variato, dalle nuove ore di inizio e di fine del turno, e dalla data di decorrenza di validità della variazione. Per ogni operaio, non possono esistere più variazioni di turno relative allo stesso giorno della settimana e aventi la stessa data di inizio di decorrenza.

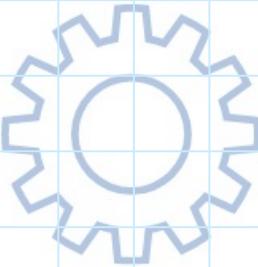
b) Si definisca uno schema relazionale corrispondente allo schema E-R.



Generalizzazione

- Clienti (partitaIVA, nome, residenza)
- Contratti (progressivo, dStipula, cliente)
- Singoli (progressivo, dStipula, loc_comune, loc_indirizzo)
- Periodici (progressivo, dStipula, periodicità)
- Locali (comune, indirizzo, descrizione)
- R2 (per_progressivo, per_dStipula, loc_comune, loc_indirizzo)
- Interventi (data, inizio, fine, loc_comune, loc_indirizzo)
- Operaio (nome, dNascita, dataDa^{NULL})
- Variazione (....)

- Singoli [progressivo, dStipula] \subseteq_{FK} Contratti [progressivo, dStipula]
- Periodici [progressivo, dStipula] \subseteq_{FK} Contratti [progressivo, dStipula]
- Contratti [cliente] \subseteq_{FK} Clienti [partitaIVA]
- Singoli [loc_comune, loc_indirizzo] \subseteq_{FK} Locali [comune, indirizzo]
- R2 [per_progressivo, per_dStipula] \subseteq_{FK} Periodici [progressivo, dStipula]
- R2 [loc_comune, loc_indirizzo] \subseteq_{FK} Locali [comune, indirizzo]
- Interventi [loc_comune, loc_indirizzo] \subseteq_{FK} Locali [comune, indirizzo]



APPUNTI DI INGEGNERIA INFORMATICA

GAIA BERTOLINO

Esercitazione 3

domenica 30 gennaio 2022 23:09



6 novembre 2021

esercit2810

Corso di Laurea in Ingegneria Informatica (DM 270) Esame di Basi di Dati 8 luglio 2019

Esercizio 4: Interrogazioni su un database relazionale (50 minuti)

Sia dato il seguente schema di database:

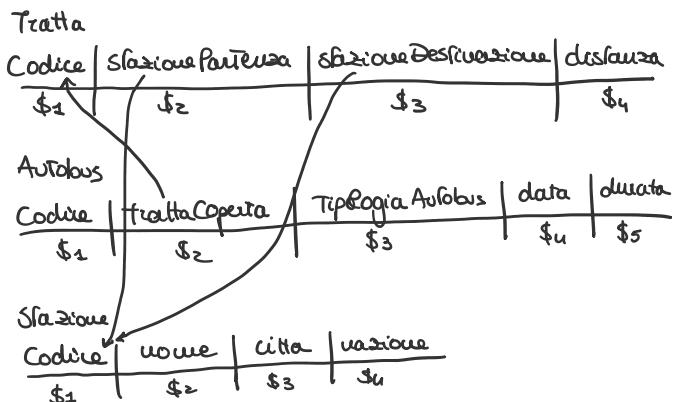
Tratta (codice, stazionePartenza, stazioneDestinazione, distanza)
Autobus (codice, trattaCoperta, tipologiaAutobus, data, durata)
Stazione (codice, nome, città, nazione)

con i seguenti vincoli di chiave esterna:

Tratta[stazionePartenza] ⊆_{FK} *Stazione*[codice];
Tratta[stazioneDestinazione] ⊆_{FK} *Stazione*[codice];
Autobus[trattaCoperta] ⊆_{FK} *Tratta*[codice].

Lo studente formuli in SQL e, dove possibile, in Algebra Relazionale le seguenti interrogazioni:

1. I nomi delle stazioni che sono il punto di partenza di almeno due tratte, ciascuna delle quali è coperta da almeno due autobus di uguale tipologia (NB: una stazione è coperta da una tratta se ne è stazione di partenza o di destinazione).
2. Le tipologie di autobus delle quali non esiste alcun autobus che copra una tratta che parte dalla città di Roma e che copre una distanza inferiore a 200 km.



INGNERIA
TICA

GAIÀ BERTOLINO

$\sigma \pi \cup \cap - \bowtie$

Tratta (codice, stazionePartenza, stazioneDestinazione, distanza)
 Autobus (codice, trattaCoperta, tipologiaAutobus, data, durata)
 Stazione (codice, nome, città, nazione)

Tratta[stazionePartenza] FK Stazione[codice];
 Tratta[stazioneDestinazione] FK Stazione[codice];
 Autobus[trattaCoperta] FK Tratta[codice].

autoAlm2 = almeno 2 autobus di uguale tipologia

tratteSi = le tratte che sono coperte da almeno due autobus di uguale tipologia

tratteAlm2 = almeno 2 tratteSi diverse che abbiano la stessa stazione di partenza (che in tratteSi+ è \$3 perché c'è 1 di tratteSi e 4 di tratta)

stazioniSi = stazioni che sono il punto di partenza di almeno due tratteSi

autoAlm2 = autobus $\bowtie_{\$1=\$1 \wedge \$3=\$3}$ autobus

tratteSi = $\pi_{\$2} (\text{autobus } \bowtie_{\$1=\$1 \wedge \$3=\$3 \wedge \$2=\$2} \text{ Autobus} \bowtie \text{ Autobus})$
autobus)

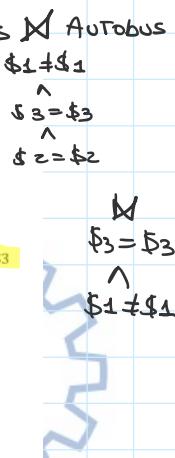
tratteSi+ = (tratteSi $\bowtie_{\$1=\$1}$ tratta)

tratteAlm2 = tratteSi+ $\bowtie_{\$1=\$1 \wedge \$3=\$3}$ tratteSi+

tratteAlm2 = (tratteSi $\bowtie_{\$1=\$1}$ tratta) $\bowtie_{\$1=\$1 \wedge \$3=\$3}$ (tratteSi $\bowtie_{\$1=\$1}$ tratta)

stazioniSi = $\pi_{\$3} (\text{tratteAlm2})$

Q = $\pi_{\$3} (\text{stazioniSi } \bowtie_{\$1=\$1} \text{ stazione})$



1) Stazioni da cui

- partono almeno due tratte
- ogni tratta è coperta da almeno due autobus della stessa tipologia

~~Almeno 2 Tratte = $(\text{Tratta } \bowtie \text{ Tratta})$~~

$\begin{matrix} \$2=\$2 \\ \wedge \\ \$3=\$3 \\ \wedge \\ \$1 \neq \$3 \end{matrix}$

$\hookrightarrow \text{codiciTratte}$
 $\text{Coperte2Autobus} = \Pi_{\$2} (\text{Autobus } \bowtie \text{ Autobus})$

$\begin{matrix} \$2=\$2 \\ \wedge \\ \$3=\$3 \\ \wedge \\ \$1 \neq \$1 \end{matrix}$

Query = $\Pi_{\$2} \left\{ \Pi_{\$3} \left[(\text{Coperte2Autobus} \bowtie \text{ Tratta}) \bowtie (\text{Coperte2Autobus} \bowtie \text{ Tratta}) \right] \bowtie \text{ Stazione} \right\}$

$\begin{matrix} \$3=\$3 \\ \wedge \\ \$1=\$1 \\ \wedge \\ \$4=\$4 \\ \wedge \\ \$1 \neq \$1 \end{matrix}$

SELECT S. nome

FROM 2tratte AS T1, Stazione AS S, 2tratte AS T2

WHERE T1.codice \neq T2.codice

AND T1.città = T2.città

FROM licenze AS L1, viuzione AS V1, circuite AS C1
 WHERE T1.codice ≠ T2.codice
 and T1.siazionePartenza = T2.siazionePartenza
 and T1.siazioneDesinazione ≠ T2.siazioneDesinazione;
 and T1.siazionePartenza = S.codice

```

CREATE VIEW 2Tratte AS
(SELECT *
FROM Coperta2Autobus AS C , Tratta AS T
WHERE C.TrattaCoperta = T.codice
);
  
```

```

CREATE VIEW Coperta2Autobus (Tratta) AS
(SELECT A1.TrattaCoperta
FROM Autobus AS A1 , Autobus AS A2
WHERE A1.codice ≠ A2.codice
and A1.TrattaCoperta = A2.TrattaCoperta
and A1.tipologiaAutobus = A2.tipologiaAutobus);
  
```

2) Tipologie autobus

- sono fanno parte di tratte che partono da zona e dist. < 200 km

$$\Delta \text{Riunivolare} = \Pi_{\$1} \left\{ \left[\begin{array}{l} G_{\$2=\text{Roma}} \\ \wedge \\ \$4 < 200 \end{array} \right] \wedge \text{Tratta} \wedge \text{Autobus} \right\} \wedge \text{Siazione} \quad \wedge \quad \begin{array}{l} \$2 = \$1 \\ \$2 = \$2 \end{array}$$

$$\text{Query} = \Pi_{\$3} (\text{Autobus}) - \Delta \text{Riunivolare}$$

GAIA BERTOLINO

SQL

```

SELECT Autobus.Tipologia
FROM Autobus
  
```

EXCEPT

```

SELECT
FROM Autobus, Tratta
WHERE Autobus.TrattaCoperta = Tratta.codice
and Tratta.distanza < 200
and Tratta.siazionePartenza = "Roma"
  
```

Corso di Laurea in Ingegneria Informatica (DM 270)
Esame di Basi di Dati
6 marzo 2018

Esercizio 4: Interrogazioni su un database relazionale (50 minuti)

Sia dato il seguente schema di database:

- CONCESSIONARIA(Codice, nome, Indirizzo, Città, Cap, Nazione)
- AUTOMOBILE(CodAutomobile, Prezzo, Nome, Marca, Concessionaria)
- GUIDATORE(CodiceFiscale, Nome, Cognome, Città, DataNascita, Cittadinanza)
- ACQUISTO(CodAcquisto, Automobile, Guidatore, Anno)

con i seguenti vincoli di chiave esterna:

- AUTOMOBILE [Concessionaria] \ominus_k CONCESSIONARIA[Codice];
- ACQUISTO [Automobile] \ominus_k AUTOMOBILE[CodAutomobile];
- ACQUISTO[Guidatore] \ominus_k GUIDATORE[CodiceFiscale];

Lo studente formuli in SQL e, dove possibile, in Algebra Relazionale le seguenti interrogazioni:

- 1) Le concessionarie della città di Torino o Napoli che hanno venduto almeno due automobili, ciascuna delle quali è stata acquistata da almeno due guidatori distinti nati tra il 1984 e il 1988.
- 2) Il nome delle concessionarie situate in Germania che non hanno mai venduto un'automobile ad un guidatore nato nel 1998.

- CONCESSIONARIA(Codice, nome, Indirizzo, Città, Cap, Nazione)
 - AUTOMOBILE(CodAutomobile, Prezzo, Nome, Marca, Concessionaria)
 - GUIDATORE(CodiceFiscale, Nome, Cognome, Città, DataNascita, Cittadinanza)
 - ACQUISTO(CodAcquisto, Automobile, Guidatore, Anno)
-
- AUTOMOBILE [Concessionaria] \ominus_k CONCESSIONARIA[Codice];
 - ACQUISTO [Automobile] \ominus_k AUTOMOBILE[CodAutomobile];
 - ACQUISTO[Guidatore] \ominus_k GUIDATORE[CodiceFiscale];

guidatoriSi = guidatori nati tra il 1984 e il 1988

acquistiSi = gli acquisti effettuati da guidatoriSi

guidAlm2 = almeno 2 guidatoriSi distinti che hanno acquistato la stessa automobile

autoSi = automobili acquistate da almeno 2

guidatoriSi (guidAlm2)

vendAlm2 = almeno 2 autoSi distinte vendute dalla stessa concessionaria

Q = concessionaria di Torino o Napoli che ha venduto almeno 2 autoSi (vendAlm2)

guidatoriSi = $\sigma_{\$5 \leq 1988 \wedge \$5 \geq 1984}$ (guidatori)

acquistiSi = acquisto $\bowtie_{\$3=\$1}$ guidatoriSi

guidAlm2 = acquistiSi $\bowtie_{\$2=\$2 \wedge \$3!=\$3}$ acquistiSi

guidAlm2 = (acquisto $\bowtie_{\$1!=\$1 \wedge \$2=\$2 \wedge \$3!=\$3}$ acquisto) $\bowtie_{\$3=\$1}$

guidatoriSi) $\bowtie_{\$7=\$1}$ guidatoriSi

guidAlm2 = (acquisto $\bowtie_{\$3=\$1}$ guidatoriSi) $\bowtie_{\$1!=\$1 \wedge \$2=\$2 \wedge \$3!=\$3}$

(acquisto $\bowtie_{\$3=\$1}$ guidatoriSi)

autoSi = $\pi_{\$2}$ (guidAlm2)

GE
NERIA
TICA
LINO

Corso di Laurea in Ingegneria Informatica (DM 270)
Esame di Basi di Dati
17 giugno 2019

Esercizio 4: Interrogazioni su un database relazionale (50 minuti)

Sia dato il seguente schema di database:

- CASAMADRE(CodCasaMadre, Nome, AnnoNascita)
 - PARRUCCHIERE(CodParrucchiere, Nome, Indirizzo, Città, CasaMadre)
 - CLIENTE(CodiceFiscale, Nome, Cognome, Città, Età, Genere)
 - SERVIZIO(Parrucchiere, Cliente, Data, Tipologia)
- con i seguenti vincoli di chiave esterna:
- PARRUCCHIERE[CasaMadre] \sqsubseteq_R CASAMADRE[CodCasaMadre];
 - SERVIZIO [Parrucchiere] \sqsubseteq_R PARRUCCHIERE[CodParrucchiere];
 - SERVIZIO[Cliente] \sqsubseteq_R CLIENTE[CodiceFiscale];

Lo studente formuli in SQL e, dove possibile, in Algebra Relazionale le seguenti interrogazioni:

- 3) Le case madri nate tra il 1980 e il 2015 a cui afferiscono almeno due parrucchieri della città di Roma o Firenze, ciascuno dei quali ha almeno due clienti distinti di età compresa tra 18 e 25 anni.
- 4) Il nome dei parrucchieri situati nella città di Roma che nell'anno 2018 non hanno mai effettuato un servizio di tipologia "colore" a un cliente di genere maschile.



CASAMADRE

Cod.CasaMadre	Nome	AnnoNascita
\$1	\$2	\$3

APPUNTI DI INGEGNERIA

INFORMATICA
GAIA BERTOLINO

PARRUCCHIERE

Cod.Parrucchiere	Nome	Indirizzo	Città	CasaMadre
\$1	\$2	\$3	\$4	\$5

CLIENTE

CodiceFiscale	Nome	Cognome	Città	Età	Genere
\$1	\$2	\$3	\$4	\$5	\$6

SERVIZIO

Parrucchiere	Cliente	Data	Tipologia
\$1	\$2	\$3	\$4

- 1) • Casamadre
- Tra 1980 e 2015
- Due parrucchieri di Roma V firenze
- Ogni parrucchiere ha due clienti con età fra 18 e 25