Esercizi di Database Svolti

Stefano Borzì

Rispondere alle seguenti domande. 2 punti se la risposta è giusta, -2 se la risposta è sbagliata.
a) Sono date due relazioni R(A,B) e S(B,C) dove B è chiave primaria in S. La cardinalità di una relazione è indicata col simbolo #. Le cardinalità delle due relazioni sono: #(R) = 5000 e #(S) = 2000. Quindi: #(R JoinNaturale S) <= ?
#(R JoinNaturale S) <= 10000

b) B: Sono date due relazioni R(A,B,C) e S(A,D,E) dove A è chiave primaria in S e R.A è una chiave esterna su S.A. Quindi: #(R JoinNaturale S) = #(R). VERO o FALSO?
VERO

Progettazione

Descrivere le regole di trasformazione della strategia Bottom-Up.

T1: si individua nella specifica una classe di oggetti con proprietà comuni e si introduce un'**entità** corrispondente.
T2: si individua nella specifica un legame logico fra entità e si introduce una associazione fra esse.
T3: si individua una generalizzazione fra entità.
T4: a partire da una serie di attributi si individua un'entità che li aggrega
T5: a partire da una serie di attributi si individua una relazione che li aggrega

Descrivere gli indici di prestazione per la valutazione degli schemi E-R. Cosa bisogna conoscere per poter effettuare l'analisi di questi indici?
{
- **Correttezza**: verificare che non ci siano errori **sintattici** o **semantici**.
- **Completezza:** tutti i dati di interesse sono rappresentati e tutte le operazioni possono essere eseguite a partire dai concetti dello schema
- **Leggibilità**: rappresentare i requisiti in maniera naturale e facilmente comprensibile
- **Minimalità**: ogni specifica deve essere rappresentata una sola volta (senza ridondanze).

}

Descrivere le fasi della progettazione logica
{

Input: schema E-R iniziale (da ristrutturare), Carico Applicativo.

Ristrutturazione dello schema E-R:
- analisi delle ridondanze
- eliminazione delle generalizzazioni
- partizionamento/accorpamento
- scelta degli identificatori primari

Output: schema E-R ristrutturato

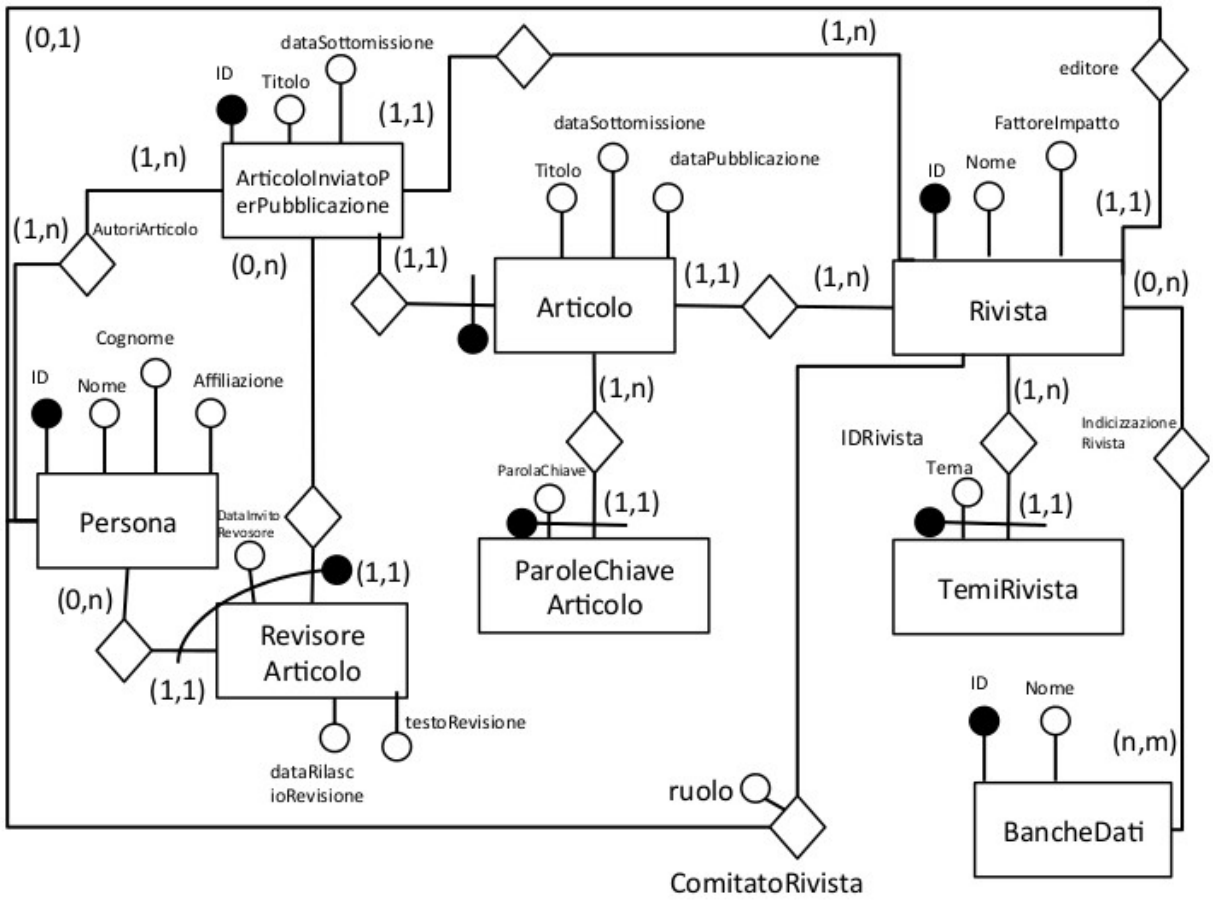
Traduzione verso il Modello Logico:
fa riferimento ad un modello logico (ad es. relazionale) e può includere ulteriore ottimizzazione che si basa sul modello logico stesso (es. normalizzazione).

("Discussione su come vengono tradotte le relazioni e cenni cosa analizza la fase di normalizzazione.")

}

Dallo schema relazionale produrre uno schema ER (definito in modo completo con attributi, chiavi, e cardinalità relazioni) che lo descrive.

Rivista(ID, Nome, FattoreImpatto, idEditore)
TemiRivista(IDRivista,Tema)
BancheDati(ID, Nome)
IndicizzazioneRivista(ID_Rivista, ID_BancaDati)
Articolo(ID, Titolo, rivista, dataSottomissione, dataPubblicazione)
ParoleChiaveArticolo(IDArticolo, ParolaChiave)
Persona(ID, Nome, Cognome, Affiliazione)
AutoriArticolo(articolo, autore)
ArticoloInviatoPerPubblicazione(ID, Titolo, rivista, dataSottomissione)
RevisoreArticolo(revisore, idarticolo, dataInvitoRevisore, dataRilascioRevisione, testorevisione)
ComitatoRivista(idrivista,idpersona,ruolo)



Normalizzazione

Considerate la relazione R(A,B,C,D) ed il seguente insieme di dipendenze funzionali

C -> D
C -> A
B -> C

Identificare le possibili chiavi per R;
B è una chiave
Decomporre R in BCNF;
R1(B,C), R2(C,D,A)

Avete ottenuto una decomposizione senza perdite e che mantiene le dipendenze? Discuterne; (?)
La decomposizione è senza perdite poiché facendo la giunzione (join) tra le due tabelle R1 ed R2 si può ottenere lo schema iniziale (R) quindi si può affermare che questa decomposizione gode della proprietà della "**preservazione dei dati**" (loss-less join)

La decomposizione in R1 ed R2 mantiene le dipendenze poiché tutte le dipendenze funzionali si "mantengono", ovvero la tabella R1 contiene B,C quindi la dipendenza funzionale $B \rightarrow C$ si "preserva" come anche le dipendenze $C \rightarrow D$ e $C \rightarrow A$ si mantengono poiché R2 contiene tutte e 3 i campi (A,C e D)

Gestione Transazioni

Descrivere le proprietà ACIDE delle transazioni

{

- **Atomicità**: una transazione non può lasciare il database in uno stato intermedio, dopo un guasto/errore prima del commit tutte le operazioni svolte devono essere annullate
- **Consistenza**: la transazione rispetta i vincoli di integrità
- **Isolamento**: una transazione non risente l'effetto di altre transazioni
- **Durata**: effettuato il commit di una transazione i suoi effetti nel database non vanno perduti, anche in presenza di guasti

}

La gestione dei lock con particolare attenzione al 2PL e il 2PL stretto.

{

la 2PL garantisce “a priori” la CSR e si basa principalmente su due regole:
- proteggere tutte le letture e scritture con lock
- ogni transazione dopo aver rilasciato un lock non può acquisirne altri

2PL stretto (unica differenza): I lock possono essere rilasciati solo dopo il commit o abort

}

Descrivere il funzionamento del logfile nei DBMS relazionali

{

È un file che riporta tutte le operazioni in ordine effettuate sul database (è un file sequenziale gestito dal controllore dell'affidabilità).
Il log serve a “ricostruire” le operazioni effettuate nel database nel caso di guasti.

}

Descrivere l’algoritmo di ripresa a caldo di un DBMS.

Dopo un guasto del server per ripristinare e stabilizzare il database si applica il cosiddetto “warm restart” (algoritmo di ripresa a caldo). Esso consiste nel trovare l'ultimo checkpoint percorrendo i log a ritroso, costruire due insiemi “UNDO” e “REDO” che conterranno rispettivamente le transazioni da disfare e da rifare, ripercorrere il log all'indietro fino alla più vecchia operazione di UNDO e REDO disfacendo tutte le azioni delle transazioni UNDO e, successivamente, percorrendo in avanti il log facendo tutte le azioni delle transazioni REDO.

Indicare se i seguenti schedule possono produrre anomalie; i simboli ci e ai indicano l’esito (commit o abort) della transazione.

1. r1(x), w1(x), r2(x), w2(y), a1, c2
2. r1(x), w1(x), r2(y), w2(y), a1, c2
3. r1(x), r2(x), r2(y), w2(y), r1(z), a1, c2
4. r1(x), r2(x), w2(x), w1(x), c1, c2
5. r1(x), r2(x), w2(x), r1(y),c1, c2
6. r1(x), w1(x), r2(x), w2(x),c1, c2

Anomalie:

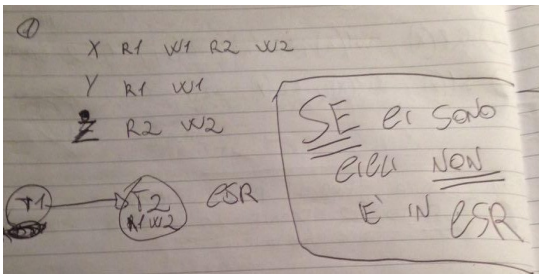
1. Lettura sporca
2. Nessuna anomalia
3. Nessuna anomalia
4. Perdita di aggiornamenti
5. Aggiornamento fantasma
6. Nessuna anomalia

Considerare i seguenti schedule

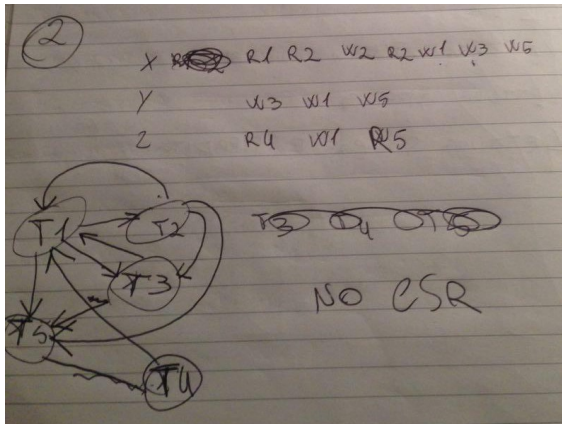
1. r1(x),w1(x),r2(z),r1(y),w1(y),r2(x),w2(x),w2(z)
2. r1(x),r2(x),w2(x),r2(x),r4(z),w1(x),w3(y),w3(x),w1(y),w5(x),w1(z),w5(y),r5(z)
3. r1(x),r3(y),w1(y),w4(x),w1(t),w5(x),r2(z),r3(z),w2(z),w5(z),r4(t),r5(t)
4. r1(x),r1(t),r3(z),r4(z),w2(z),r4(x),r3(x),w4(x),w4(y),w3(y),w1(y),w2(t)
5. r2(x),r4(x),w4(x),r1(y),r4(z),w4(z),w3(y),w3(z),w1(t),w2(z),w2(t)

Dire quali di questi sono CSR

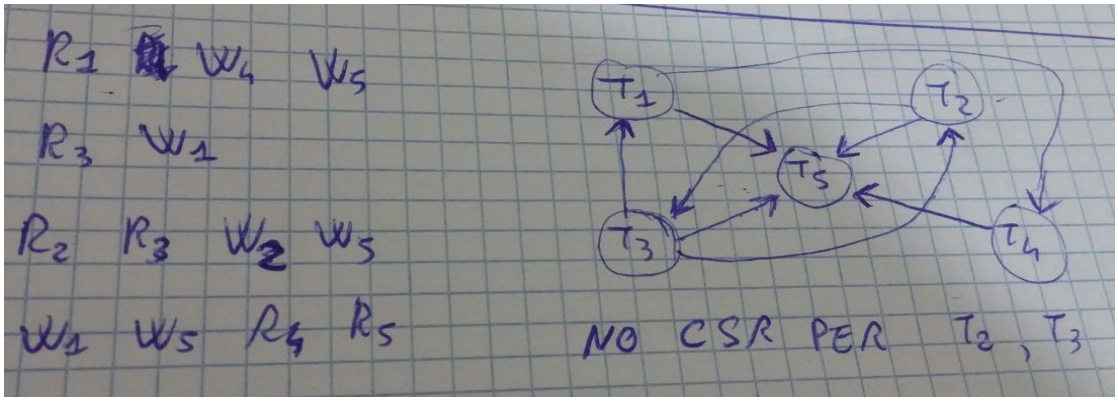
1)



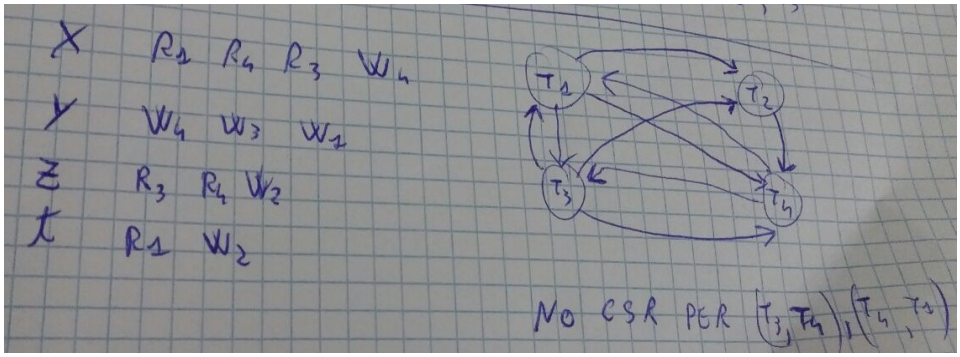
2)



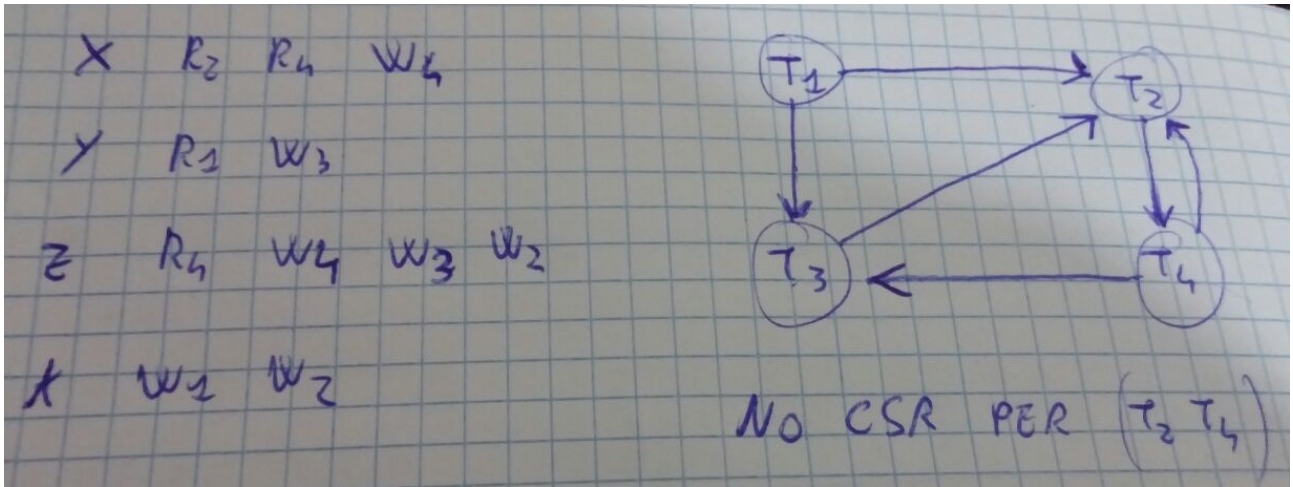
3)



4)



5)



Come svolgere un esercizio di schedule 2PL lock

Il lock a due fasi si compone di due tipi di lock
- **Lock condiviso** per la lettura (più transazioni possono leggere nello stesso momento)
- **Lock esclusivo** per la scrittura

Una transazione può "potenziare" il suo lock di lettura per portarlo in scrittura (quindi accesso esclusivo) solo se nessun'altra transazione lo ha già

L1, L2, W2
W2 non può essere acquisito perché la prima transazione non ha rimosso il suo lock quindi un tentativo di scrittura potrebbe causare incosistenza

Negli esercizi, la richiesta di lock non è palese ma si deduce da 2 cose
Quando appare per la prima volta la transazione → Richiesta di lock (condiviso/esclusivo)
Quando una transazione non compare più → tutti i lock sono stati rilasciati
(i passaggi da condiviso a esclusivo non sono esplicitamente menzionate)
Per cui, tutto l'esercizio si svolge ordinando temporalmente le richieste di lock
E vedendo se una risorsa (precedentemente bloccata) è ancora sotto il controllo (anche condiviso) di un'altra

Esempio:

R1(x), R2(x), W3(x) --> tutto ok, r1 e r2 sono compatibili mentre r3 non ha concorrenti

Invece nel caso di:

R1(x), W1(x), R2(x), W1(x) --> errore

Perché (secondo ciò che ho scritto prima) fino alla fine, la transazione "1" non rilascia il suo lock quindi R2x non potrà mai avere un controllo anche parziale

Altro esempio:

R1(x), W1(x), W2(x)

Il tutto va a buon fine, poichè la transazione "1" legge, scrive e poi rimuove il lock (non ricomparendo da li in poi)

Quindi la transazione "2" è libera di accedere ad una risorsa che non è soggetta ad alcun tipo di lock

~ Salvatore Manfredi

Esercizio scheduler

Considerare i seguenti schedule
1. r1(x),w1(x),r2(z),r1(y),w1(y),r2(x),w2(x),w2(z)
2. r1(x),r2(x),w2(x),r2(x),r4(z),w1(x),w3(y),w3(x),w1(y),w5(x),w1(z),w5(y),r5(z)
3. r1(x),r3(y),w1(y),w4(x),w1(t),w5(x),r2(z),r3(z),w2(z),w5(z),r4(t),r5(t)
4. r1(x),r1(t),r3(z),r4(z),w2(z),r4(x),r3(x),w4(x),w4(y),w3(y),w1(y),w2(t)
5. r2(x),r4(x),w4(x),r1(y),r4(z),w4(z),w3(y),w3(z),w1(t),w2(z),w2(t)

Si presentassero a uno scheduler che usa il locking a due fasi, quali transazioni verrebbero messe in attesa?
Si noti che, una volta posta in attesa una transazione, le sue successive azioni non vanno più considerate

1. Non è presente nessun attesa di lock
2. w1(x), w2(x), w3(x), w5(x) → in attesa
3. w1(y), w4(x), w5(x) → in attesa
4. w2(z), w4(x) → in attesa
5. w4(x), w3(y) → in attesa

Applicare l'algoritmo di ripresa a caldo all'esempio di seguito e calcolare l'insieme di REDO

esercizio 1

```
B(T1)
B(T2)
U(T2, O1, B1, A1)
I(T1, O2, A2)
B(T3)
C(T1)
B(T4)
U(T3,O2,B3,A3)
U(T4,O3,B4,A4)
CK(T2,T3,T4)
C(T4)
B(T5)
U(T3,O3,B5,A5)
U(T5,O4,B6,A6)
D(T3,O5,B7)
A(T3)
C(T5)
I(T2,O6,A8)
```

0. UNDO = {T2, T3, T4}REDO = {}
1. C(T4) → UNDO = {T2, T3} REDO = {T4}
2. B(T5) → UNDO = {T2, T3, T5} REDO = {T4}
3. C(T5) → UNDO = {T2, T3} REDO = {T4, T5}

(Undo)

4. I(T2, O6, A8) → D(O6)
5. D(T3, O5, B7) → O5 = B7
6. U(T3, O3, B5, A5) → O3 = B5
7. U(T3, O2, B3, A3) → O2 = B3
8. U(T2, O1, B1, A1) → O1 = B1

(Redo)

9. U(T3, O3, B5, A5) → O3 = A5
10. U(T5, O4, B6, A6) → O4 = A6

esercizio 2

```
DUMP, B(T1),
B(T2),
B(T3),
I(T1, O1, A1),
D(T2, O2, B2),
B(T4),
U(T4, O3, B3, A3),
U(T1, O4, B4,
A(4),
C(T2),
CK(T1, T3, T4),
B(T5),
B(T6),
U(T5, O5,B5, A5),
A(T3),
CK(T1, T4, T5, T6),
B(T7),
A(T4),
U(T7, O6, B6, A6),
U(T6, O3, B7, A7),
B(T8),
A(C7),
guasto
```

UNDO = {T1, T4, T5, T6} REDO = {}
UNDO = {T1, T4, T5, T6, T7} REDO = {}
UNDO = {T1, T4, T5, T6, T7} REDO = {}
UNDO = {T1, T4, T5, T6, T7, T8} REDO = {}
UNDO = {T1, T4, T5, T6, T7, T8, /*C7*/} REDO = {}

(Undo)

U(T6, O3, B7, A7) => O3 = B7
U(T7, 06, B6, A6) => O6 = B6
U(T5, O5, B5, A5) => O5 = B5
U(T1, O4, B4) => O4 = B4
U(T4, O3, B3, A3) => O3 = B3
D(T2,O2,B2) => I(T2,O2,B2)
I(T1, O1, A1) => D(O1)

(Redo)
Non ci sono REDO da svolgere

Definire i 4 livelli di isolamento previsti dai DBMS.

- **READ UNCOMMITTED:** livello in cui sono visibili gli aggiornamenti effettuati da altri utenti se non consolidati (può causare problemi di consistenza dei dati)
- **READ COMMITTED:** livello in cui sono visibili gli aggiornamenti solo dopo il consolidamento
- **REPEATABLE READ (Default):** in questo caso l'aggiornamento è visibile solo se i dati sono consolidati e la transazione che legge è stata terminata
- **SERIALIZIBLE:** funziona come il repeatable read ma, in più, la lettura di un dato blocca gli aggiornamenti fino al termine della transazione

XML

Dato il seguente DTD:

```
<!ELEMENT collezione (descrzione,ricette*)>
<!ELEMENT descrizione ANY>
<!ELEMENT ricetta (titolo,ingredienti*,preparazione,commenti?,fattori_nutritivi)>
<!ELEMENT titolo (#PCDATA)>
<!ELEMENT ingredienti (ingrediente*,preparazione)?>
<!ATTLIST ingrediente nome CDATA #REQUIRED
                      quantità CDATA #IMPLIED
                       unità CDATA #IMPLIED>
<!ELEMENT preparazione (passi*)>
<!ELEMENT passo (#PCDATA)>
<!ELEMENT commenti (#PCDATA)>
<!ELEMENT fattori_nutritivi EMPTY>
<!ATTLIST fattori nutritivi proteine CDATA #REQUIRED
                       carboidrati CDATA #REQUIRED
                      grassi CDATA #REQUIRED
                      calorie CDATA #REQUIRED
                      alcol CDATA #IMPLIED>
```

(a) Dare un file xml che descriva una ricetta con 3 passi.

```
<collezione>
   <descrizione>..descrizione..</descrizione>
   <ricetta>
      <titolo>titolo della ricetta</titolo>
      <ingredienti>
         <ingrediente nome="nome" quantita="100" unita="grammi" />
         <preparazione>
            <passo>1</passo>
            <passo>2</passo>
            <passo>3</passo>
         </preparazione>
      </ingredienti>
      <preparazione>
         <passo>..preparazione ricetta..</passo>
         <passo>ciao</passo>
      </preparazione>
      <commenti>no comment</commenti>
      <fattori-nutritivi proteine="c" carboidrati="c" grassi="c" calorie="c" alcol="c" />
   </ricetta>
</collezione>
```

(b) Scrivere una query in xquery per trovare le ricette che hanno 450 calorie e che richiedono 5 passi

```
for $ricetta in doc("ricette.xml")//ricetta
let $c = $ricetta/fattori-nutritivi/@calorie;
let $p = $ricetta/preparazione/passo;

return   { where ( ($c == 450) and (count($p) == 5) ) $ricetta; }
```

Descrivere lo XML Schema e le differenze rispetto al DTD.

XML Schema è stato creato per definire in modo più preciso gli elementi e la composizione di un documento XML. (XSD = XML Schema Definition)
Gli XSD sono estendibili, in formato XML, più ricchi e completi dei DTD, capaci di supportare tipi di dati diversi da PCDATA e namespace multipli.

Descrivere il costrutto sequence

Il costrutto **sequence** definsice l'ordine di come devono apparire i sotto-elementi (o gli attributi) di un elemento complesso (complexType element), al contrario del costrutto **all** che definisce gli elementi di un elemento complesso ma non l'ordine.

Dato il file xml (colonna sinistra) e il foglio di stile XSLT (colonna destra) scrivere il file XML che si ottiene come output dopo l'applicazione della trasformazione.

```
<?xml version="1.0" encoding="UTF-8"?>
<films xmlns:xsi="http://www.w3.org/2001/XMLScherna-instance">
 <film stelle="5">
    <titolo>Un Lupo Mannaro Americano a Londra</titolo>
     <anno>1981</anno>
     <regista>
          <cognome>Landis</cognome>
         <nome>John< /nome>
     </regista>
 </film>
 <film stelle="4">
    <titolo>La cosa</titolo>
    <anno>1982</anno>
   <regista>
      <cognome>Carpenter</cognome>
     <nome>John</ nome>
   </regista>
 </film>
 <film stelle="3">
   <titolo> La mosca</titolo>
   <anno>1986</anno>
  <regista>
        <cognome> Cronenberg </cognome>
       <nome> David < /nome>
    </regista>
 </film>
</films>
```

```
<?xml version='1.O'?>
<xsl:stylesheet
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
   <xsl:apply-templates/>
</xsl:template>

<xsl:template match="films">
<film-registi>
  <xsl : apply-templates/>
</film-registi >
</xsl:template>

<xsl:template match="film">
 <registra film="{titolo}">
   <nome>
     <xsl:value-of select="regista/nome"/>
     <xsl:value-of select="regista/cognome"/>
   </nome>
 </regista>
</xsl : template>
</xsl : stylesheet>
```

```
<film-registi>
   <regista film="Un Lupo Mannaro Americano a Londra">
      <nome>
         John
         Landis
      </nome>
   </regista>
   <registra film="La cosa">
      <nome>
         John
         Carpenter
     </nome>
   </regista>
   <registra film="La mosca">
      <nome>
         David
         Cronenberg
      </nome>
   </regista>
</film-registi>
```

Dati i seguenti file XML:

Catalogo.xml	prezzi.xml	ordine.xml
<catalogo> <prodotto dipartimento="WMN"> <numero>557</numero> <nome language="it">Maglione</nome> <colori>blu nero </colori> </prodotto> < prodotto dipartimento="ACC"> < numero >563</ numero > < nome language="it">Cappello</ nome > </prodotto > <prodotto dipartimento="ACC"> < numero >443</ numero > <nome language="it">Borsa da viaggio</nome> </prodotto > < prodotto dipartimento="MEN"> <numero>784</numero> <nome language="it">Maglietta</nome> <colori>bianco grigio</colori> <descrizione>La nostra maglietta preferita!</descrizione> </ prodotto > </catalogo >	<prezzi> <listino datainizio="2006-11-15"> <prod num="557"> <prezzo valuta="EU">29.99</prezzo> <sconto tipo="CLR">10.00</sconto> </prod> <prod num="563"> <prezzo valuta="EU">69.99</prezzo> </prod> <prod num="443"> <prezzo valuta="EU">39.99</prezzo> <sconto tipo="CLR">3.99</sconto> </prod> </listino> </prezzi>	<ordine num="00299432" data="2006-09-15" cliente="0221A"> <item dipartimento="WMN" num="557" quantita="1" colore="blu"/> <item dipartimento ="ACC" num="563" quantita ="1"/> <item dipartimento ="ACC" num="443" quantita ="2"/> <item dipartimento ="MEN" num="784" quantita ="1" colore ="bianco"/> <item dipartimento ="MEN" num="784" quantita ="1" colore ="grigio"/> <item dipartimento ="WMN" num="557" quantita ="1" colore ="nero"/> </order>

a) Scrivere in xquery un'interrogazione per unire le informazioni dei file catalogo.xml e ordine.xml. Creare un elenco di tutti gli elementi in ordine, insieme con il loro numero, il nome, e la quantità (es. <item num="557" nome="Maglione" quantita="1"/>)

for $item **in doc(**"ordine.xml"**)**/item
	for $c **in doc(**"Catalogo.xml"**)**
	where (($c/prodotto/**@**dipartimento == $item/**@**dipartimento**) and (**$c/prodotto/numero/**text()** == $item/**@**num**))** $item/**@**nome := "{$c/prodotto/nome/**text()**}" /* si può fare? */

return $item;

/* *return alternativo da inserire dentro il where*
return <item dipartimento="{$item/@dipartimento}" num="{$item/@num}" quantità="{$item/@quantità}" colore="{$item/@colore}" nome="{$c/prodotto/nome/text()}">
*/

b) Scrivere in xquery un'interrogazione per unire le informazioni dei file catalogo.xml e ordine.xml e prezzi.xml. Creare un elenco di tutti gli elementi in ordine di tipo "ACC", insieme con il loro numero, il nome, e la prezzo(es. <item num="563" nome="Cappello" prezzo="69.99"/>

for $item **in doc(**"ordine.xml"**)**/item
	for $c **in doc(**"Catalogo.xml"**)**
	where ($c/prodotto/**@**dipartimento == "AAC" **)**
		where (($item/**@**dipartimento == "AAC"**) and (**$c/prodotto/numero/**text()** == $item/**@**num**))**
		$item/**@**nome = "{$c/prodotto/nome/**text()**}"
	for $p **in doc(**"prezzi.xml"**)**/listino/prod
	where ($item/**@**num == $p/num/**text())**
		$item/**@**prezzo := "{$p/prezzo/**text()**}"
		where ($p/sconto**)**
		$item/**@**sconto := "{$p/sconto/**text()**}"
return $item;

Si scriva un possibile DTD che modelli le relazioni riportate di seguito.
Implementare quindi una query per estrarre i pazienti che sono stati ricoverati nel 2014.

Ricovero(reparto, paziente, dataricovero, datadimissioni, tipologiaRicovero, medicoResponsabile, cf, nome, cognome, cap, email cellulare)
prestazioni(codice, nome, costo)
ErogazionePrestazione(paziente, ricovero, prestazione, data)

```
<!DOCTYPE ospedale [
<!ENTITY Ricovero  (reparto, paziente, dataricovero, datadimissioni, tipologiaRicovero, medicoResponsabile, cf, nome, cognome, cap, email cellulare)>
<!ENTITY reparto (#PCDATA)>
<!ENTITY paziente (#PCDATA)>
<!ENTITY datadimissioni (#PCDATA)>
<!ENTITY dataricovero (#PCDATA)>
<!ENTITY tipologiaRicovero (#PCDATA)>
<!ENTITY medicoResponsabile (#PCDATA)>
<!ENTITY cf (#PCDATA)>
<!ENTITY nome (#PCDATA)>
<!ENTITY cognome (#PCDATA)>
<!ENTITY cap (#PCDATA)>
<!ENTITY email (#PCDATA)>
<!ENTITY cellulare (#PCDATA)>

<!ENTITY prestazioni (codice, nome, costo)>
<!ENTITY codice (#PCDATA)>
<!ENTITY nome (#PCDATA)>
<!ENTITY costo (#PCDATA)>

<!ENTITY ErogazionePrestazione (paziente, ricovero, prestazione, data)>
<!ENTITY paziente (#PCDATA)>
<!ENTITY ricovero (#PCDATA)>
<!ENTITY prestazione (#PCDATA)>
<!ENTITY data (#PCDATA)>
]>
```

```
for $paziente in doc("ospedale.dtd")/Ricovero
where $paziente/dataricovero = 2014
return $paziente
```