BASI DI DATI 2

TECNICHE PER IL CONTROLLO DELLA CONCORRENZA

a.a 2024/2025

Prof.ssa G. Tortora

Tecniche per il controllo della concorrenza

- L'esecuzione di transazioni concorrenti senza alcun controllo può comportare svariati problemi al database.
- È necessario evitare che interferiscano fra di loro e garantire l'isolamento.
- Si usano delle tecniche di gestione delle transazioni, per garantire che il database sia sempre in uno stato consistente.
- Tali tecniche garantiscono la serializzabilità degli schedule, usando particolari protocolli.

Tecniche per il controllo della concorrenza (2)

- Tecniche di locking:

  i data item sono bloccati per prevenire che transazioni multiple accedano allo stesso item concorrentemente;

- Timestamp:

  un timestamp è un identificatore unico per ogni transazione generato dal sistema.

  Un protocollo può usare l'ordinamento dei timestamp per assicurare la serializzabilità.

Un fattore importante: la granularità

- La granularità di un data item è la porzione del database rappresentata dal data item.

- Un data item può essere della dimensione che varia da un attributo ad un singolo blocco di un disco o anche un intero file o un intero database.

Tecniche di Locking

Tecniche di Locking per il controllo della concorrenza

- Tali tecniche si basano sul concetto di *'blocco'* (*lock*) di un item.

- Un **lock** è una variabile associata ad un data item nel db, e descrive lo stato di quell'elemento rispetto alle possibili operazioni applicabili ad esso.

- I lock sono quindi un mezzo per sincronizzare l'accesso da parte di transazioni concorrenti agli elementi del db.

Tipi di Lock

- Diversi tipi lock possono essere usati per il controllo della concorrenza.
- In particolare, esamineremo:
  - Lock binari
  - Lock shared/esclusivi

- I *lock binari* sono più semplici ma molto restrittivi.
  Non sono molto usati nella pratica.

- I *lock shared/esclusivi*, molto usati nei DBMS commerciali, forniscono maggiori capacità di controllo e concorrenza.

Lock binari

- Un lock binario può assumere due valori (*o stati)*:
  - Locked (*o valore **1***)
  - Unlocked (*o valore **0***)

- A ciascun elemento X del db viene associato un distinto lock :
  - Se **Lock(X)=1**, le operazioni del db non possono accedere all'elemento X.
  - Se **Lock(X)=0**, si può accedere all'elemento X quando richiesto.

Lock_Item e Unlock_Item

- Le transazioni che usano lock binari devono contenere operazioni di lock_item e unlock_item.
- Una transazione chiede di accedere a un elemento X con l'istruzione **lock_item(X)**:
  - se Lock(X)=1 la transazione è forzata ad attendere, altrimenti pone Lock(X) a 1, ottenendo l'accesso all'elemento.
- Alla fine dell'uso di X, la transazione invia un'istruzione di **unlock_item(X)**, che pone Lock(X) a 0, permettendo l'accesso all'item ad altre transazioni.

Lock_Item e Unlock_Item (2)

- Un Lock binario rafforza la **mutua esclusione** di un data item.
- Le operazioni di lock_item e unlock_item devono essere implementate come unità indivisibili (*sezioni critiche*), nel senso che non è consentito alcun interleaving dall'avvio fino o al termine dell'operazione di lock/unlock o all'inserimento della transazione in una coda di attesa.
- Il DBMS dispone di un sottosistema di lock manager per seguire e controllare gli accessi ai lock.

Lock_Item

```
Lock_Item (X):
    B: if Lock(X) = 0
       then Lock(X) ← 1;
       else
            begin
                 wait (until Lock(X)=0 e il lock manager
                   seleziona la transazione);
                 goto B;
            end
```

Il comando di wait è considerato fuori dalla operazione di lock_item:

altre transazioni che vogliono accedere a X si trovano nella stessa coda.

La transazione è messa in una coda di attesa per l'item X finchè X viene sbloccato e la transazione ne ottiene l'accesso

Unlock_Item

Unlock_Item (X):

Lock(X) ← 0;

if qualche transazione è in attesa

then sveglia una delle transazioni in attesa;

Lock binari: implementazione

- Per implementare un lock binario è necessaria solo una variabile binaria **LOCK** associata ad ogni data item X del database.
- Ogni lock può essere visto come un record con tre campi:

  <nome data item, LOCK, transazione>
- con associata una coda delle transazioni che stanno provando ad accedere all'elemento.
- Gli elementi che non sono nella *lock table* sono considerati non bloccati (*unlocked*).
- Organizzazione della tabella: hash file.

Regole per lock binari

- Usando uno schema di lock binario, ogni transazione deve obbedire alle seguenti regole:
  1. Una transazione T deve impartire l'operazione di Lock_Item(X) prima di eseguire una Read_Item(X) o Write_Item(X).
  2. Una transazione T deve impartire l'operazione di Unlock_Item(X) dopo aver completato tutte le operazioni di Read_Item(X) e Write_Item(X).
  3. Una transazione T non impartirà un Lock_Item(X) se già vale il lock sull'elemento X.
  4. Una transazione T non impartirà un Unlock_Item(X) a meno che non valga già un lock sull'elemento X.

Regole per lock binari (2)

- Al più una transazione può mantenere il lock su un elemento X; vale a dire che due transazioni non possono accedere allo stesso elemento concorrentemente.

Lock shared/esclusivi

- Il lock binario è troppo restrittivo, poiché l'accesso ad un data item è consentito ad una sola transazione per volta.
- Possiamo consentire l'accesso in sola lettura a più transazioni contemporaneamente.
- Se una transazione deve scrivere un data item X deve avere un accesso esclusivo su X.
- Per questo motivo si utilizza un multiple mode lock, cioè un lock che può avere più stati.

Multiple Mode Lock

- Le operazioni di lock diventano tre:
  - Read_Lock(X)
  - Write_Lock(X)
  - Unlock(X)

- Un lock ha tre possibili stati:
  - Read_Locked (share locked)
  - Write_Locked (exclusive locked)
  - Unlocked

Operazioni di Lock

- Ciascuna delle tre operazioni, Read_Lock(X), Write_Lock(X), Unlock_Item(X), deve essere considerata indivisibile:
  - nessun interleaving deve essere consentito dall'inizio dell'operazione fino o al completamento o all'inserimento della transazione in una coda di attesa per quell'elemento.

Implementazione di lock shared/esclusivi

- Possibile implementazione:
  - Ogni lock è rappresentato da un record con quattro campi:

<Nome data item, Lock, Numero di read, Transazione/i bloccante/i>

- Lock assume un valore che permette di distinguere tra Read_Locked, Write_Locked e Unlocked.
- Per risparmiare spazio, il sistema mantiene nella *lock table* i record per gli elementi locked.

Read_Lock(X)

Read_Lock(X):

```
B: if LOCK(X) = “unlocked”
   then begin
                LOCK(X) ← “read_locked”;
                numero_di_read (X) ← 1;
        end;
   else
      if LOCK(X) = “read_locked”
      then numero_di_read (X) = numero_di_read (X) + 1;
      else
         begin
            wait (until LOCK(X) = “unlocked” and il gestore di
            lock sceglie la transazione);
            goto B;
         end;
```

Write_Lock(X)

```
Write_Lock(X):
   B:  if LOCK(X) = “unlocked”
       then LOCK(X) ← “write_locked”;
       else
          begin
                wait  (until LOCK(X) = “unlocked” e il
                gestore di lock sceglie la transazione);
                goto B;
          end;
```

Unlock(X)

```
Unlock(X):
    if LOCK(X) = “write_locked”
    then
      begin
            LOCK(X) ← “unlocked”;
           sveglia una delle transazioni in attesa se ne esistono;
      end;
    else   if LOCK(X) = “read_locked”
           then
             begin
               numero_di_read (X) = numero_di_read (X) – 1;
               if numero_di_read (X) =0
               then
                 begin
                   LOCK(X) = “unlocked”;
                   sveglia una delle transazioni in attesa se ne esistono;
                 end
              end;
```

Regole per lock shared/esclusivi

- Usando uno schema di shared/exclusive, ogni transazione deve obbedire alle seguenti regole:
  1. Una transazione T deve impartire l'operazione di Read_Lock(X) o Write_Lock(X) prima di eseguire una Read_Item(X).
  2. Una transazione T deve impartire l'operazione di Write_Lock(X) prima di eseguire una Write_Item(X).
  3. Una transazione T deve impartire l'operazione di Unlock(X) dopo aver completato tutte le operazioni di Read_Item(x) o Write_Item(X).
  4. Una transazione T non impartirà un Read_Lock(X) se già è in possesso di un lock condiviso in lettura o lock esclusivo in scrittura sull'elemento X.

Regole per lock shared/esclusivi (2)

4. Una transazione T non impartirà un Write_Lock(X) se già vale il lock in lettura o scrittura sull'elemento X.
5. Una transazione T non impartirà un Unlock(X) a meno che non valga già un lock sull'elemento X.

- I vincoli 4 e 5 possono essere tralasciati per permettere conversioni di lock …

Conversione di Lock

- Una transazione può invocare un Read_Lock(X) e poi successivamente incrementare il lock, invocando un Write_Lock(X).
  - Tale conversione è possibile solo se T è l'unica transazione che ha un Read_Lock su X.
  - Altrimenti, deve aspettare.
- È possibile anche decrementare un lock, se una transazione T invoca una Write_Lock(X) e successivamente una Read_Lock(X).

Conversione di Lock (2)

- Per permettere tali conversioni, è necessario che sia mantenuto un identificatore della transazione nella struttura del record per ciascun lock.
  - È ovviamente necessario modificare le operazioni di Read_Lock, Write_Lock e Unlock per supportare l'informazione aggiuntiva.

Lock e serializzabilità

- Lock binari e multiple-mode non garantiscono la serializzabilità degli schedule.

T_1	T_2
read_lock(*Y*);	read_lock(*X*);
read_item(*Y*);	read_item(*X*);
unlock(*Y*);	unlock(*X*);
write_lock(*X*);	write_lock(*Y*);
read_item(*X*);	read_item(*Y*);
X:=*X*+*Y*;	*Y*:=*X*+*Y*;
write_item(*X*);	write_item(*Y*);
unlock(*X*);	unlock(*Y*);

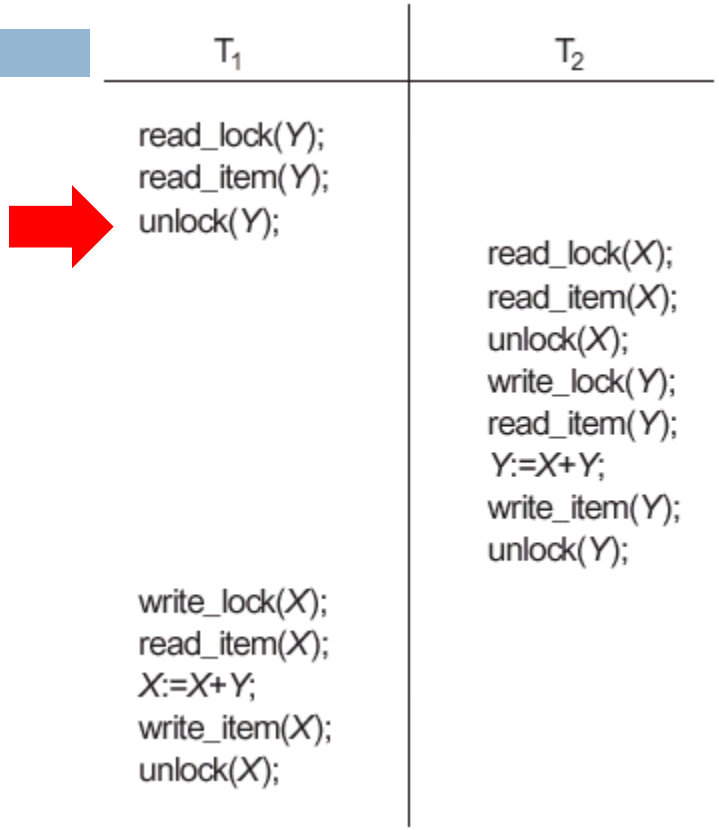
Esempio di schedule seriali

Dati i valori iniziali
X=20 e Y=30:

Se T_1 è seguito da T_2:
X=50, Y=80

Se T_2 è seguito da T_1:
X=70, Y=50

Lock e serializzabilità (2)



Dati i valori iniziali
X=20 e Y=30:

Risultato dello schedule S:
X=50, Y=50

Y è stato sbloccato troppo presto!

Esempio di schedule nonserializzabile

- Occorre un protocollo (*cioè una serie di regole*) per stabilire il posizionamento delle operazioni di lock/unlock in ogni transazione.

Il protocollo Two-Phase Locking

Protocollo Two-Phase Locking

- Definizione: una transazione T segue il protocollo Two-Phase Locking (*2PL*) se tutte le operazioni locking (Read_Lock, Write_Lock) precedono la prima operazione di Unlock nella transazione.

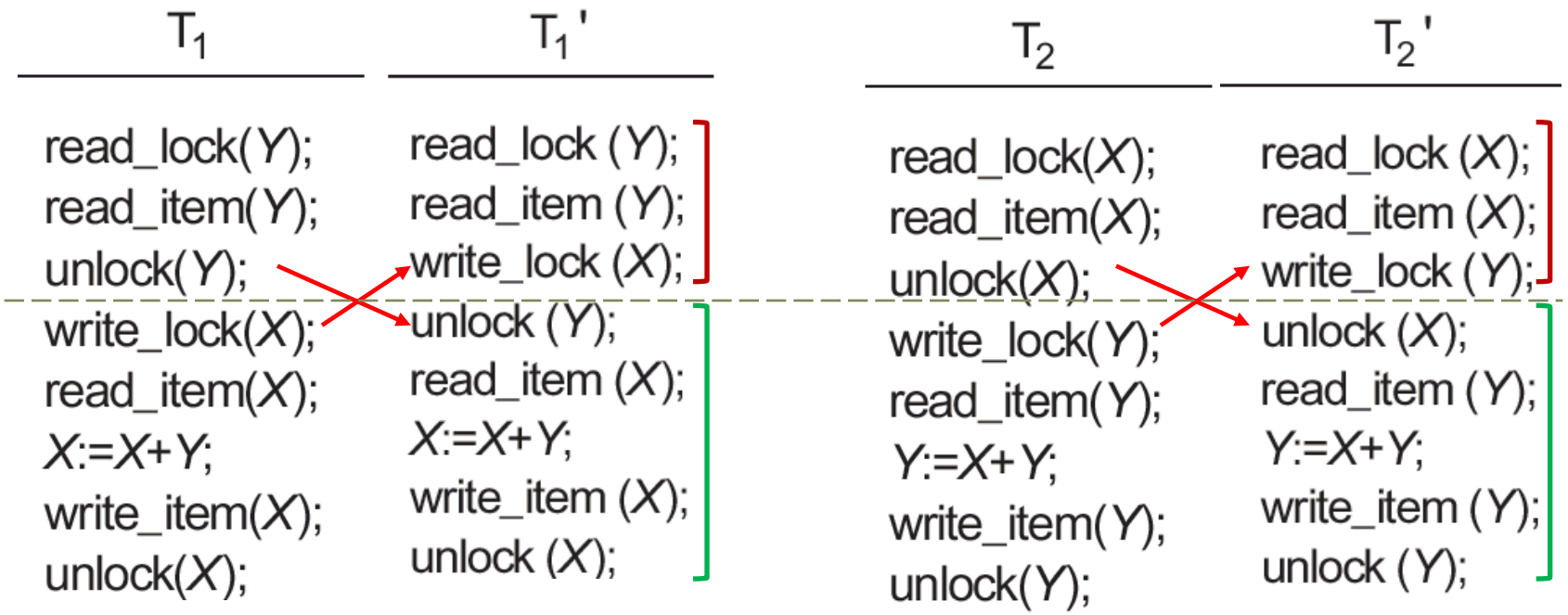
- Una transazione del genere può essere divisa in due fasi:
  1. expanding phase
  2. shrinking phase

Expanding e Shrinking phase

- Nella expanding phase, possono essere acquisiti nuovi lock su elementi ma nessuno può esserne rilasciato.
- Nella shrinking phase i lock esistenti possono essere rilasciati ma non possono essere acquisiti nuovi lock.
- Se la conversione di lock è permessa, l'upgrading deve essere fatta durante la fase di espansione ed il downgrading durante la contrazione.

Transazioni 2PL: *Esempio*

- Le transazioni T_1 e T_2 viste in precedenza non seguono il protocollo 2PL.
- Le riscriviamo come T_1' e T_2':



2PL e serializzabilità

- È dimostrabile che se ogni transazione in uno schedule segue il protocollo 2PL, allora lo schedule è serializzabile.

- 2PL può però limitare la concorrenza in uno schedule:
  - la garanzia della serializzabilità viene pagata al costo di non consentire alcune situazioni di concorrenza possibili, poiché alcuni elementi possono essere bloccati più del necessario, finché la transazione necessita di effettuare letture e scritture.

2PL Conservativo

- Il protocollo 2PL appena visto è detto 2PL di base.

- Una variazione del 2PL è nota come 2PL conservativo (*o statico*):
  - richiede che una transazione, prima di iniziare, blocchi tutti gli elementi a cui accede, predichiarando i propri read_set e write_set:
    - Il read_set è l'insieme di tutti i data item che saranno letti dalla transazione.
    - Il write_set è l'insieme di tutti i data item che saranno scritti dalla transazione.

2PL Conservativo (2)

- Se qualche data item dei due insiemi non può essere bloccato, la transazione resta in attesa finché tutti gli elementi necessari non divengono disponibili.
- Il 2PL conservativo è un protocollo deadlock-free.
- Non viene usato nella pratica perché è necessario predichiarare il read-set ed il write-set.

  Cosa difficile in molte situazioni.

2PL Stretto

- La variazione più diffusa del protocollo 2PL è il 2PL stretto, che garantisce schedule stretti, in cui le transazioni non possono né scrivere né leggere un elemento X finché l'ultima transazione che ha scritto X non termina (con commit o abort).

- Nel 2PL stretto, quindi, una transazione non rilascia nessun lock esclusivo finché non termina.

- Non è deadlock-free.

Generazione automatica di richieste di read e write lock

- In molti casi il **sottosistema per il controllo della concorrenza** genera automaticamente le richieste di lock:
  - Quando la transazione effettua una Read_Item(X), il sistema genera una operazione Read_Lock(X).
  - Quando la transazione effettua una Write_Item(X), il sistema genera una operazione Write_Lock(X).

Generazione automatica di richieste di read e write lock (2)

- Se T invoca un Read_Item(X), il sistema invoca un Read_Lock(X) per T.
  - Se lo stato di LOCK(X) = "write_locked" da una T':
    - il sistema pone T nella coda di attesa per X;
    - altrimenti, esegue il Read_Lock(X) ed quindi l'operazione di Read_Item(X) per T.
- Se T invoca un Write_Item(X), il sistema invoca un Write_Lock(X) per T.
  - Se lo stato di LOCK(X) = "write_locked" OR "read_locked" da una T':
    - il sistema pone T nella coda di attesa per X.
  - Se lo stato di LOCK(X) = "read_locked" dall'unica transazione T:
    - il sistema promuove il lock a "write_locked" ed esegue l'operazione di Write_Item(X) per T.
  - Se lo stato di LOCK(X) = "unlocked":
    - il sistema esegue la Write_Lock(X) e quindi l'operazione di Write_Item(X) per T.

Problemi connessi all'uso dei lock

- Il protocollo di lock a due fasi garantisce la serializzabilità, ma non consente tutti i possibili schedule serializzabili
  - cioè alcune schedule serializzabili vengono vietati dal protocollo.
- Causa Deadlock.
- Causa Starvation.

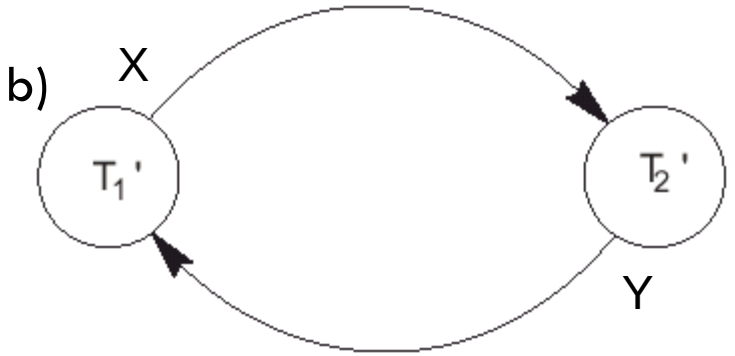
Cos'è un Deadlock?

- Si ha un deadlock quando due (o più) transazioni aspettano qualche item bloccato da altre transazioni T' in un insieme.

  Ogni transazione T' è in una coda di attesa e aspetta che un elemento sia rilasciato da un'altra transazione in attesa.

a)

T_1'	T_2'
read_lock(*Y*);	
read_item(*Y*);	
	read_lock(*X*);
	read_item(*X*);
write_lock(*X*);	
	write_lock(*Y*);



Esempio di deadlock:

a) Uno schedule di T_1' e T_2' in deadlock.

b) Il grafo wait-for (delle attese) corrispondente.

Protocolli di deadlock prevention

- Per prevenire il deadlock, occorre usare un protocollo apposito (*deadlock prevention protocol*).

- Il protocollo a prevenzione di deadlock usato nel 2PL conservativo richiede che ogni transazione blocchi tutti i data item di cui ha bisogno in anticipo;

  se qualcosa non può essere ottenuta, nessun elemento è bloccato per cui la transazione aspetta e riprova dopo.

- *Svantaggi:* limita la concorrenza.

Altri schemi di deadlock prevention

- Sono stati proposti molti altri schemi per la prevenzione di deadlock:
  - Basati su timestamp.
  - Senza timestamp:
    - Algoritmo non-waiting.
    - Algoritmo cautions waiting.
  - Basati su timeout.
    - T richiede un lock aspettando fino ad un certo **timeout**.
    - Se il lock non viene assegnato in tempo, T subisce un'abort, esegue un rollback e ricomincia (*indipendentemente dalla presenza di deadlock*)

- In attesa
- Abort
- Preempt

Prevenzione di deadlock basata su timestamp

- Il Timestamp TS(T) di una transazione T è un identificatore unico associato ad ogni transazione.
- Un timestamp si basa sull'ordine di partenza delle transazioni:
  - Se T_1 inizia prima di T_2, allora

$$\mathbf{TS(T_1) < TS(T_2)}$$

Schemi basati su timestamp

- *T_i prova a bloccare X che è bloccato da T_j*

- Schema wait-die:
  - se $TS(T_i) < TS(T_j)$, (T_i più vecchia) allora T_i aspetta;
  - altrimenti (T_i più giovane) T_i viene abortita (**abort**) e ripartirà con lo stesso timestamp.

Schemi basati su timestamp (2)

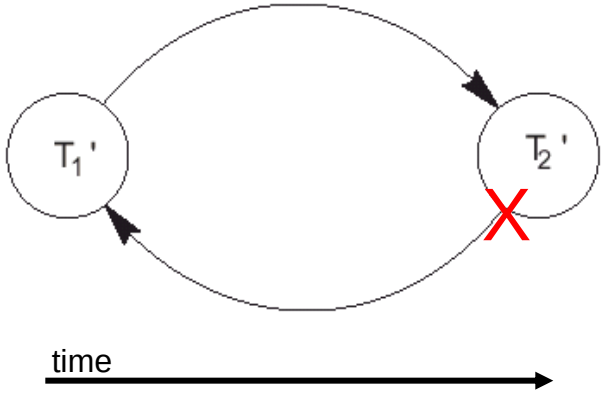
- Schema wound-wait:
  - se $TS(T_i) < TS(T_j)$, (T_i più vecchia) allora T_j fallisce (**preempt**) e viene riavviata successivamente con lo stesso timestamp;
  - altrimenti (T_i più giovane) T_i aspetta.

- Entrambi gli schemi fanno fallire la transazione **più giovane**.
- Sono deadlock-free, ma possono causare l'abort di transazioni senza necessità (*non provocherebbero realmente un deadlock*).

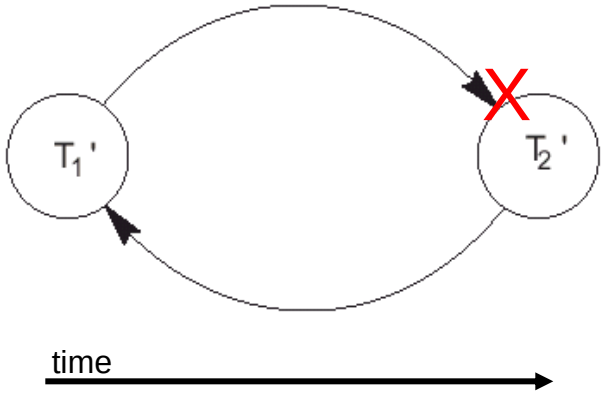
Schemi basati su timestamp (3)

- T_1' prova a bloccare X che è bloccato da T_2'
- T_2' prova a bloccare Y che è bloccato da T_1'

- wait-die



- wound-wait



Schemi senza timestamp

- **No-waiting (NG):** Se una transazione T non è in grado di ottenere un lock, viene immediatamente interrotta e successivamente fatta ripartire dopo un intervallo di tempo senza effettuare controlli che si possa verificare un deadlock.
  - Nessuna T aspetta, quindi non esiste la possibilità di deadlock.
  - Questo metodo non è pratico. Può causare la terminazione e il riavvio di transazioni inutilmente.

- **Cautious Waiting (CW):** Se $\mathbf{T_i}$ cerca un lock sull'item X ma non è in grado in quanto X è bloccato da $\mathbf{T_k}$**:**
  - Se $\mathbf{T_k}$ non è in attesa di qualche altro item bloccato, allora $\mathbf{T_i}$ aspetta, altrimenti $\mathbf{T_i}$ viene terminata.

Deadlock detection

- Approccio più pratico: *il sistema controlla l'esistenza di un deadlock.*
- Per riconoscere un stato di deadlock, si usa il grafo wait_for (delle attese):
  - Si crea un nodo per ogni transazione in esecuzione.
  - Si aggiunge un arco tra il nodo T_i e il nodo T_j se T_i aspetta di bloccare un elemento usato da T_j.
  - Si cancella un arco tra T_i e T_j appena l'elemento richiesto da T_i viene allocato a T_i.
- Se c'è un ciclo nel grafo, si è in uno stato di deadlock.
- Una volta riconosciuto un deadlock, si sceglie quale transazione abortire, usando un criterio di selezione della vittima.

Deadlock detection (2)

- L'algoritmo che seleziona la vittima in genere evita di scegliere transazioni che:
  - Sono in esecuzione da molto tempo.
  - Hanno effettuato molti aggiornamenti.

- La **deadlock detection** è una soluzione valida quando non ci sono molte interferenze tra le transazioni:
  - le transazioni sono brevi ed ogni transazione blocca pochi elementi.
  - in alternativa, per transazioni lunghe si usa il **deadlock prevention**.

Starvation

- La starvation è un altro problema che può sorgere con l'utilizzo dei lock.
- Una transazione è nello stato di starvation se non può procedere per un tempo indefinito mentre altre transazioni nel sistema continuano normalmente.
- La causa è nello schema di waiting non sicuro che dà la precedenza ad alcune transazioni invece di altre.

Starvation (2)

- Un possibile schema di waiting sicuro (*safe*) usa una coda first-come-first-serve:
  - Le transazioni bloccano gli elementi rispettando l'ordine con cui hanno richiesto il lock.

- Un altro schema è basato su **priorità**, che aumenta proporzionalmente al tempo atteso dalla transazione.
- *Starvation si può avere anche negli algoritmi per il trattamento del deadlock, se l'algoritmo seleziona ripetutamente la stessa transazione come vittima.*

  Soluzione: *l'algoritmo può usare priorità più alte per transazioni che sono state abortite più volte.*

- Gli schemi wait-die e wound-wait escludono la starvation.

Granularità degli item

Dimensione di data item

- Tutte le tecniche per il controllo della concorrenza assumono che il db sia formato da un insieme di data item.

- Un data item può essere:
  - Un record del db.
  - Un campo di un record del db.
  - Un blocco del disco.
  - Un intero file.
  - L'intero db.

Granularità di data item

- La dimensione di un data item è detta granularità del data item.

- La granularità può essere:
  - Fine: riferita a data item di piccole dimensioni.
    *(**es.** campo di un record)*
  - Grossa: riferita a data item di dimensioni maggiori.
    *(**es.** file, database)*

Granularità e prestazioni

- La granularità **influenza le prestazioni** nel controllo della concorrenza e del recovery.

- Maggiore è il livello di granularità, minore è la concorrenza permessa
  - Se la dimensione di un data item è un blocco del disco, una transazione che necessita di leggere un record X in un blocco B effettuerà un lock dell'intero blocco.
  - Altre transazioni, interessate a record diversi da X ma ugualmente in B, resteranno quindi inutilmente in attesa.

Granularità e prestazioni (2)

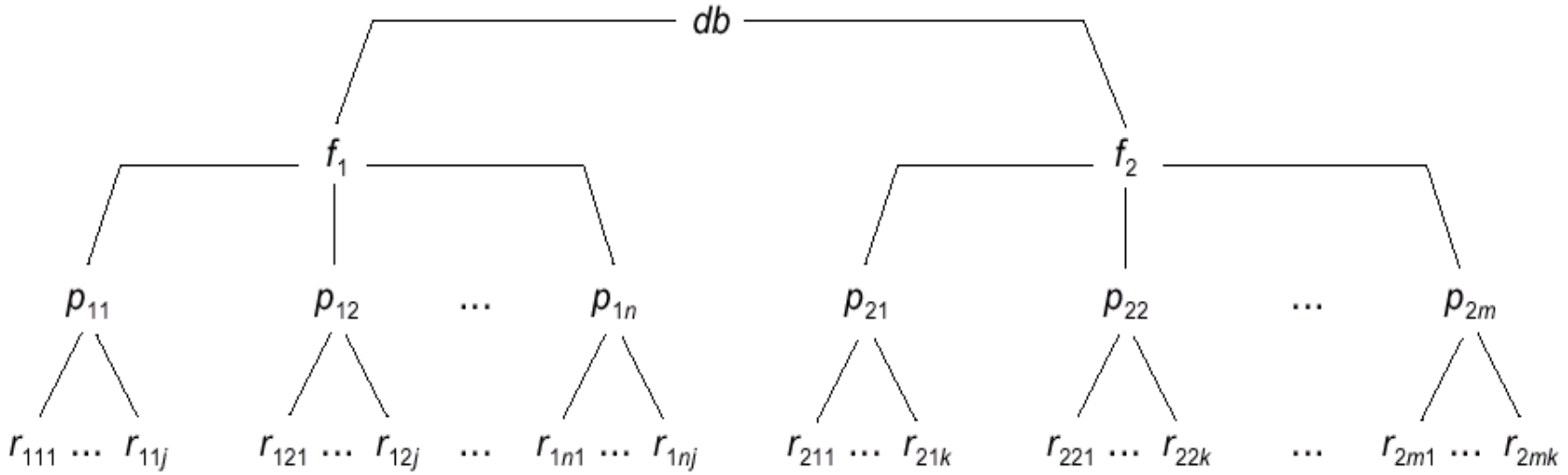
- Per contro, a un data item di dimensioni inferiori corrisponde un numero maggiore di item nel database:
  - Ci sarà quindi una quantità superiore di lock ed il lock manager introdurrà un overhead nel sistema a causa delle molte operazioni che dovrà gestire.
  - Sarà richiesto molto spazio per gestire la tabella dei lock.

Quale è la taglia migliore?

- Dipende dal tipo di transazioni coinvolte:
  - Se una transazione tipica accede a:
    - un piccolo numero di record, è vantaggioso avere una granularità di un record.
    - molti record nello stesso file, è vantaggioso avere una granularità a livello blocco o a livello file.

Granularità multipla

- La soluzione è nella possibilità di definire granularità multiple:
  - un lock può essere chiesto su item a qualsiasi livello di granularità.



Una gerarchia di granularità