



Basi di Dati  
Modulo Tecnologie

# Esecuzione concorrente di transazioni (II parte)

Dr. Sara Migliorini

# Concetti di base

## Schedule

- Sequenza di operazioni di lettura e scrittura eseguite su risorse della base di dati da diverse transazioni concorrenti
  - S:  $r_1(x)$   $r_2(z)$   $w_1(x)$   $w_2(z)$
- Esso rappresenta una possibile esecuzione concorrente di diverse transazioni.
- Quali schedule accettare per evitare le anomalie?
- Si stabilisce di accettare **solo** gli schedule "equivalenti" ad uno schedule seriale.

# Concetti di base

## Schedule seriale

- E' uno schedule dove le operazioni di ogni transazione compaiono in sequenza senza essere inframmezzate da operazioni di altre transazioni.
- s1: r1(x) r2(x) w2(x) w1(x) NON è SERIALE
- s2: r1(x) w1(x) r2(x) w2(x) è SERIALE

# Concetti di base

## Schedule serializzabile

- È uno schedule “equivalente” ad uno schedule seriale
- A questo punto occorre definire il concetto di EQUIVALENZA tra schedule.
  - L'idea è che si vogliono considerare equivalenti due schedule che producono gli stessi effetti sulla base di dati.
  - Quindi accettare schedule serializzabili significa accettare schedule che hanno gli stessi effetti di uno schedule seriale.

# Equivalenza tra schedule

## Ipotesi di commit-proiezione

- Si suppone che le transazioni abbiano **esito noto**. Quindi si tolgono dagli schedule tutte le operazioni delle transazioni che non vanno a buon fine.
- Richiedono questa ipotesi le seguenti tecniche di gestione della concorrenza:
  - Gestione della concorrenza basata sulla **view-equivalenza**.
  - Gestione della concorrenza basata sulla **conflict-equivalenza**.

# View-equivalenza

Se avviene prima una SCRITTURA e poi una LETTURA sulla  
STESSA RISORSA ma su TRANSAZIONI DIVERSE

## Definizioni preliminari

- Relazione "LEGGE\_DA"

- Dato uno schedule  $S$  si dice che un'operazione di lettura  $r_i(x)$ , che compare in  $S$ , LEGGE\_DA un'operazione di scrittura  $w_j(x)$ , che compare in  $S$ , se  $w_j(x)$  precede  $r_i(x)$  in  $S$  e non vi è alcuna operazione  $w_k(x)$  tra le due.

- Esempio

- $S1: r1(x) \ r2(x) \ w2(x) \ w1(x)$        $LEGGE\_DA(S1) = \emptyset$
- $S2: r1(x) \ w1(x) \ r2(x) \ w2(x)$        $LEGGE\_DA(S2) = \{(r2(x), w1(x))\}$



# View-equivalenza

## Definizioni preliminari

- "SCRITTURE FINALI"
  - Dato uno schedule  $S$  si dice che un'operazione di scrittura  $w_i(x)$ , che compare in  $S$ , è una SCRITTURA FINALE se è l'ultima operazione di scrittura della risorsa  $x$  in  $S$ .
- Esempio
  - $S1: r1(x) r2(x) w2(x) w3(y) w1(x)$ 
    - $SCRITTURE\_FINALI(S1) = \{w3(y), w1(x)\}$
  - $S2: r1(x) w1(x) r2(x) w2(x) w3(y)$ 
    - $SCRITTURE\_FINALI(S2) = \{w2(x), w3(y)\}$

# View-equivalenza

## View-equivalenza

- Due schedule  $S1$  e  $S2$  sono view-equivalenti ( $S1 \approx_v S2$ ) se possiedono le stesse relazioni LEGGE\_DA e le stesse SCRITTURE FINALI.

## View-serializzabilità

- Uno schedule  $S$  è view-serializzabile (VSR) se esiste uno schedule seriale  $S'$  tale che  $S' \approx_v S$ .

**Attenzione:** gli schedule seriali  $S'$  si generano considerando tutte le possibili permutazioni delle transazioni che compaiono in  $S$ . Inoltre, NON SI DEVE cambiare l'ordine delle operazioni di una transazione, ciò significherebbe modificare la transazione!



# Esercizi sul test VSR

---

- Gli schedule corrispondenti alle anomalie di perdita di aggiornamento, di letture inconsistenti ed di aggiornamento fantasma, non sono view-serializzabili.
- Esempi di applicazione del test VSR.

# View-equivalenza

## Osservazioni

- L'algoritmo per il test di **view-equivalenza** tra due schedule è di **complessità lineare**.
- L'algoritmo per il test di **view-serializzabilità** di uno schedule è di **complessità esponenziale** (è necessario generare tutti i possibili schedule seriali).
- In conclusione:
  - la VSR richiede algoritmi di complessità troppo elevata,
  - richiede l'ipotesi di commit-proiezione
  - $\Rightarrow$  non è applicabile nei sistemi reali.

# Conflict-equivalenza

## Definizioni preliminari

- "CONFLITTO"
  - Dato uno schedule  $S$  si dice che una coppia di operazioni  $(a_i, a_j)$ , dove  $a_i$  e  $a_j$  compaiono nello schedule  $S$ , rappresentano un conflitto se:
    - $i \neq j$  (transazioni diverse)
    - entrambe operano sulla stessa risorsa
    - almeno una di esse è una operazione di scrittura
    - $a_i$  compare in  $S$  prima di  $a_j$ .

# Conflict-equivalenza

## Conflict-equivalenza

- Due schedule  $S1$  e  $S2$  sono conflict-equivalenti ( $S1 \approx_c S2$ ) se possiedono le stesse operazioni e gli stessi conflitti (= le operazioni in conflitto sono nello stesso ordine nei due schedule).

## Conflict-serializzabilità

- Uno schedule  $S$  è conflict-serializzabile (CSR) se esiste uno schedule seriale  $S'$  tale che  $S' \approx_c S$ .

# Conflict-equivalenza

## Osservazioni

- L'algoritmo per il test di **conflict-serializzabilità** di uno schedule è di **complessità lineare**.
- Algoritmo (dato uno schedule  $S$ ):
  - Si costruisce il grafo dei conflitti  $G(N,A)$  dove
    - $N=\{t_1, \dots, t_n\}$  con  $t_1, \dots, t_n$  transazioni di  $S$ ;
    - $(t_i, t_j) \in A$  se esiste almeno un conflitto  $(a_i, a_j)$  in  $S$ .
  - Se il grafo così costruito è ACICLICO allora  $S$  è conflict-serializzabile.

# Conflict-equivalenza

## Verifica algoritmo sul grafo

- Se  $S$  è CSR  $\Rightarrow$  il suo grafo è ACICLICO
  - Se uno schedule  $S$  è CSR  $\Rightarrow$  è  $\approx_C$  ad uno schedule seriale  $S_{ser}$ .
  - Supponiamo che le transazioni in  $S_{ser}$  siano ordinate secondo il TID:  $t_1, t_2, \dots, t_n$ . Poiché  $S$  è  $\approx_C$  a  $S_{ser}$ ,  $S_{ser}$  ha tutti i conflitti di  $S$  esattamente nello stesso ordine.
  - Ora nel grafo di  $S_{ser}$  poiché è seriale ci possono essere solo archi  $(i,j)$  con  $i < j$  e quindi il grafo non può avere cicli, perché un ciclo richiede almeno un arco  $(i,j)$  con  $i > j$ .
  - Poiché  $S \approx_C S_{ser}$  ha come già detto gli stessi conflitti di quest'ultimo e quindi lo stesso grafo che risulterà anch'esso ACICLICO.

# Conflict-equivalenza

## Verifica algoritmo sul grafo

- Se il grafo di  $S$  è ACICLICO  $\Rightarrow S$  è CSR
  - Se il grafo di  $S$  è **aciclico**, allora esiste fra i nodi un **"ordinamento topologico"**, vale a dire una numerazione dei nodi **tale che il grafo contiene solo archi  $(i,j)$  con  $i < j$ .**
  - Per dimostrare che  $S$  è CSR occorre generare a partire da questa osservazione uno schedule seriale che abbia gli stessi conflitti di  $S$ .
  - Lo schedule seriale le cui **transazioni sono ordinate secondo l'ordinamento topologico è CONFLICT-equivalente a  $S$** , in quanto ha lo stesso grafo di  $S$  e quindi gli stessi conflitti di  $S$ .
  - Quindi l'ACICLICITÀ del grafo di  $S$  assicura la possibilità di ottenere uno schedule seriale  $\approx_c$  a  $S$  e quindi assicura che  $S$  sia CSR.

# Esercizi sul test CSR

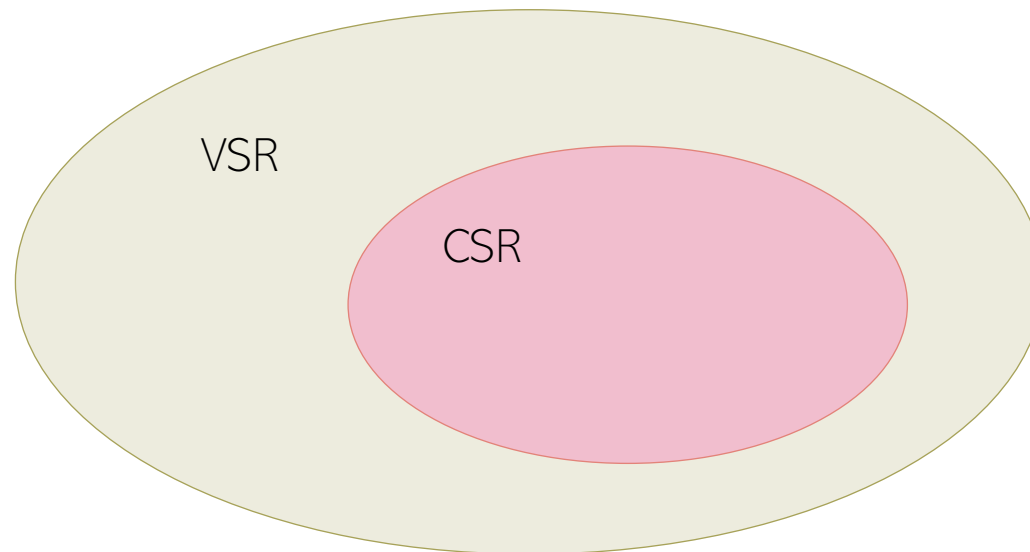
- Esempi di applicazione del test CSR



# CSR vs VSR

## Osservazioni

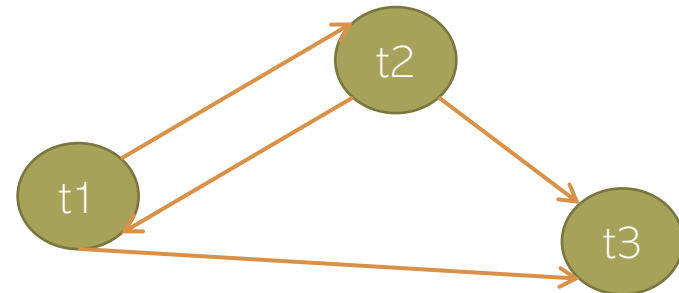
- La conflict-serializzabilità è condizione sufficiente ma non necessaria per la view-serializzabilità.



# CSR $\subset$ VSR

Controesempio per la non necessità:

- $r1(x) \ w2(x) \ w1(x) \ w3(x) \ \text{LEGGE\_DA}=\emptyset \ \text{SCR\_FIN}=\{w3(x)\}$
- view-serializzabile: in quanto view-equivalente allo schedule seriale
  - $r1(x)w1(x)w2(x)w3(x)\text{LEGGE\_DA}= \emptyset \ \text{SCR\_FIN}=\{w3(x)\}$
- non conflict-serializzabile: in quanto il grafo dei conflitti è ciclico.



# CSR $\Rightarrow$ VSR

Supponiamo  $S1 \approx_C S2$  e dimostriamo che  $S1 \approx_V S2$ :

- Osservazione: poiché  $S1 \approx_C S2$  i due schedule hanno:
  - stesse scritture finali: se così non fosse, ci sarebbero almeno due scritture sulla stessa risorsa in ordine diverso e poiché due scritture sono in conflitto i due schedule non sarebbero  $\approx_C$ .
  - stessa relazione "legge\_da": se così non fosse, ci sarebbero scritture in ordine diverso o coppie lettura-scrittura in ordine diverso e quindi, come sopra sarebbe violata la  $\approx_C$ .

# Esercizi

---

- Esercizi sui test VSR e CSR