Due transazioni si dicono *serializzabili* se il risultato delle transazioni è uguale a quello eseguito sequenzialmente, dunque non avendo sovrapposizioni temporali.
Per esempio avendo:
r2(x) r1(x) r2(y) w2(y) r1(y) w1(x)
Vediamo che le operazioni sono eseguite in maniera "ordinata", dunque a delle letture seguono delle scritture. In particolare, una lettura di T1 su x e due letture di T2 su x e y, una scrittura di T2 su y e una lettura/scrittura di t1 su x. *Non ci sono transazioni sovrapposte.*
Se si scambiassero:

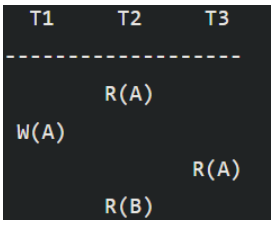
- r1(x) con r2(y)
- r1(x) con w2(y)

avremmo:
r2(y) r2(y) w2(y) r1(x) r1(y) w1(x)
Anche qui, avremmo una lettura e scrittura di T2, poi due letture di T1 su x e y e una operazione in scrittura in modo isolato, quindi senza che l'altra legga quel valore "pendente".
Dunque, tutto bene.

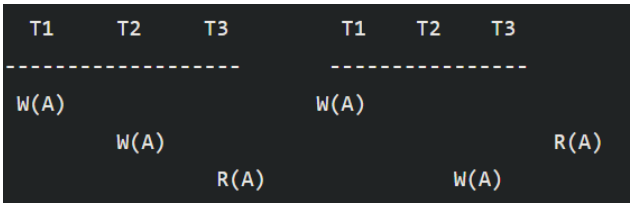
Per introdurre il concetto di *view-derivabilità* tra due schedule devono essere soddisfatte due condizioni:

- lettura iniziale, cioè se una transazione T1 legge il dato A dal database nello schedule S1, allora anche nello schedule S2 anche T1 deve leggere A dal database.

Ad esempio T2 che legge da:

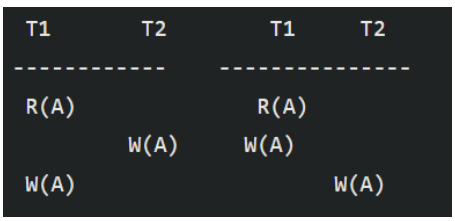


- lettura aggiornata, Se T_i stesse leggendo A che viene aggiornato da T_j in S1, allora in S2 anche T_i dovrebbe leggere A che è aggiornato da T_j.



Qui sopra abbiamo l'esempio in cui T3 legge un valore aggiornato da T2 (in S1) e anche T3 legge A aggiornato da T1 (dentro S2). Questo non è view equivalente.

- scrittura finale, se una transazione T1 ha aggiornato A per ultimo in S1 allora in S2 anche T1 dovrà eseguire le scritture finali.



In questo esempio vediamo che non sono view-equivalenti perché l'operazione di scrittura finale in S1 è fatta da T1, mentre in S2 è fatta da T2.

Buon riassunto di *view-equivalenza*:

Two schedules T1 and T2 are said to be view equivalent, if they satisfy all the following conditions:

1. **Initial Read:** Initial read of each data item in transactions must match in both schedules. For example, if transaction T1 reads a data item X before transaction T2 in schedule S1 then in schedule S2, T1 should read X before T2.

Read vs Initial Read: You may be confused by the term initial read. Here initial read means the first read operation on a data item, for example, a data item X can be read multiple times in a schedule but the first read operation on X is called the initial read. This will be more clear once we will get to the example in the next section of this same article.

2. **Final Write:** Final write operations on each data item must match in both the schedules. For example, a data item X is last written by Transaction T1 in schedule S1 then in S2, the last write operation on X should be performed by the transaction T1.

3. **Update Read:** If in schedule S1, the transaction T1 is reading a data item updated by T2 then in schedule S2, T1 should read the value after the write operation of T2 on same data item. For example, In schedule S1, T1 performs a read operation on X after the write operation on X by T2 then in S2, T1 should read the X after T2 performs write on X.

Per il discorso di *conflict-serializzabile*, diciamo semplicemente che due operazioni sono in conflitto se:

- operano sullo stesso dato
- appartengono a due transizioni differenti
- almeno una delle operazioni è una scrittura

Detto in tre parole:

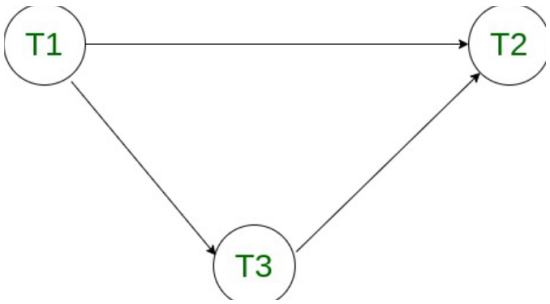
1) una transazione non potrà andare in conflitto con sé stessa a seguito di lettura/scrittura
2) non sapendo in un contesto reale in che ordine si eseguono le operazioni, banalmente, ogni volta che si ha una lettura/scrittura da parte di due transizioni diverse, si ha un conflitto.

Ad esempio:

Le operazioni che vanno in conflitto (si indicano dalla precedente alla successiva sono):

1) R3(X) e W2(X) [T3 -> T2]
2) W1(Y) e R3(Y) [T1 -> T3]
3) W1(Y) e W2(Y) [T1 -> T2]
4) R3(Y) e W2(Y) [T3 -> T2]

Va costruito il grafo dei conflitti, che segue le frecce indicate e ci si accorge che *dato che non ha cicli*, lo schedule è conflict-serializzabile.



Un esempio invece come, ragionando come prima:

S = r1(x), r2(y), r3(x), w3(x), w1(x), w1(y), r2(x), w2(x)

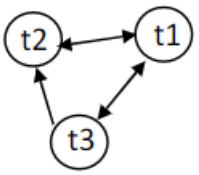
Si disegna il grafo dei conflitti avendo che:

- R1 dipende da W3 (w3(x) – r1(x))
- R3 dipende da W1 (w1(x) – r3(x))
- R2 dipende da W1 (w1(y) – r2(y))
- R2 dipende da W3 (w3(x) – r2(x))
- W2 dipende da R1 (w2(x) – r1(x))
- W2 dipende da R3 (w2(x) – r3(x))

E quindi dato che si ha un ciclo e le operazioni non sono fatte in ordine, S non è né conflict-serializzabile né view-serializzabile.

Attenzione: per capire veramente se un conflict è view-serializzabile, basta semplicemente usare il grafo dei conflitti e ordinarsi le operazioni come per il grafo; se si vede che le scritture finali cioè tutte le scritture) sono nello stesso ordine e le dipendenze sono le stesse, allora è view-serializzabile.

Se è CSR, allora comunque è VSR.



$W_1(A)\ R_2(A)\ R_2(B)\ W_2(D)\ R_3(C)\ R_1(C)\ W_3(B)\ R_4(A)\ W_3(C).$

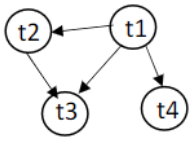
Nello schedule qui sopra:

1) scrive prima T1, T1 stessa e T2 leggono
2) scrive T2, T3 e T1 leggono
3) scrive T3, legge T4
4) scrive T3

Le operazioni sono fatte in maniera ordinata, le scritture non si sovrappongono.

È view serializzabile; i conflict ce ne stanno e di diversi.

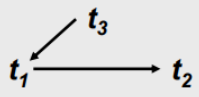
Infatti, il grafo dei conflitti riporta;



Attenzione alla relazione tra 2PL (Locking a due fasi) e CSR.

Consideriamo:

$S: r_1(x)\ w_1(x)\ r_2(x)\ w_2(x)\ r_3(y)\ w_1(y)$



Si nota che S sia CSR.
Affinché sia 2PL dovrebbe essere strutturato in questo modo (*con il locking che va fatto solamente nel caso di una relazione leggi-da* e *se le due transizioni operano sulla stessa risorsa*):

- w_lock1(x) r1(x) w1(x) unlock1(x) r2(x) w2(x) r3(y) w1(y)

Dato che esiste anche w1(y) che va in conflitto con r3(y), allora per precedenza da parte del grafo dei conflitti (il cui ordine sarebbe T3, T1, T2)
Si dovrebbe mettere prima w_lock1(y) prima di unlock1(x)

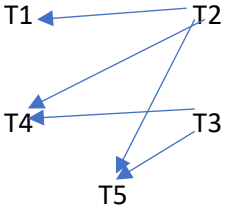
- w_lock1(x) r1(x) w1(x) w_lock1(y) unlock1(x) r2(x) w2(x) r3(y) w1(y)

Sempre per il grafo dei conflitti se si ha la lettura di y da parte di r3, essa deve precedere T2 e non è compatibile con w1(y) che segue.

Esercizio 2
Considera il seguente schedule:
S = r2(x) r1(x) w3(t) w1(x) r3(y) r4(t) r2(y) w2(z) w5(y) w4(z)
Esempi di conflitto:

S è conflict-serializable? Se sì, mostrare uno schedule che è conflict-equivalente.

Si listano i conflitti:



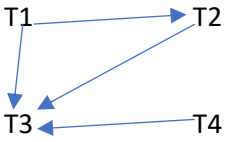
Quindi S è conflict-serializzabile.
Volendo riordinare le operazioni, si considera che quello da cui partono più frecce è T2.
Successivamente scegliamo T1 perché raggiunto direttamente da T2, perché raggiunto direttamente da T1.
T4 è raggiunta da T2 direttamente ma viene raggiunta da T3 che non viene raggiunta da T2.
Si mette dunque come secondo T3. Mettiamo poi T1 seguendo il ragionamento.
Infine, T4 poiché raggiunto da T3 e T5.
Ordine di esecuzione finale:

1) T2 r2(x), r2(y), w2(z)
2) T3 w3(t), r3(y)
3) T1 r1(x), w1(x)
4) T5 w5(y)
5) T4 r4(t), w4(z)

In questo modo i conflitti non sono stati invertiti (come se fossero eseguite da un unico client). Quindi, dato che CSR implica VSR, questa è corretta.

Esercizio 3
S = r1(x) w2(x) r3x) w1(u) w3(v) r3(y) r2(y) w3(u) w4(t) w3(t)

Dire se è conflict-serializzabile e trovare uno schedule seriale conflict-equivalente

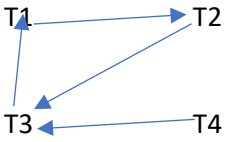


Esso è conflict-serializzabile:

Un ordine possibile è:
T1 – T4 – T2 – T3

Se per esempio si decidesse di spostare w1(u) nella posizione:
S = r1(x) w2(x) r3(x) w3(v) r3(y) r2(y) w3(u) w1(u) w4(t) w3(t)

si genererebbe un ciclo:



Qualsiasi cosa succede, w1(u) genera conflitti. Se volessi eseguire operazioni di scrittura, dovrei quindi evitare un ciclo in qualche modo.
L'operazione non può essere effettuata in quel punto; pima di effettuarla, occorre "rompere" il ciclo, cioè fare il commit o abort di t2 o t3 per togliere il nodo delle transazioni attive dal grafo.

Esercizio 4
Indicare se i seguenti schedule sono VSR:

1. r1(x), r2(y), w1(y), r2(x) w2,(x)
2. r1(x), r2(y), w1(x), w1(y), r2(x) w2,(x)
3. r1(x), r1(y), r2(y), w2(z), w1(z), w3(z), w3(x)
4. r1(y), r1(y), w2(z), w1(z), w3(z), w3(x), w1(x)

Sappiamo individuare se sono VSR se invertendo le transazioni, non si hanno relazioni del tipo *"legge da". Semplicemente: farsi il grafo dei conflitti e vedere se ci sta un ciclo. Se non è CSR non è VSR.*

1) Non è VSR: infatti invertendo ad esempio w1(y) con r2(y) oppure w2(x) con r1(x) si hanno relazioni di dipendenza.
2) Non è VSR, infatti si hanno varie dipendenze conflittuali, come w1(y) per r2(y) oppure r2(x) per w1(x). Dato che si hanno più relazioni di lettura/scrittura, cambiando l'ordine dei fattori, il risultato cambia.
3) In questo caso è VSR: infatti, non ci sono letture/scritture che concorrono, avendo una sola scrittura su X mentre tutte le altre agiscono su Z oppure su Y. Anche per w3, unico che esegue su x e z, quindi due valori diversi, non si ha ordine diverso di scrittura.
4) Non è VSR: infatti ho due scritture con T1 per x e z e due scritture su T3 per x e z; potenzialmente queste transazioni possono essere invertite e non avere lo stesso ordine di esecuzione finale.

Esercizio 5

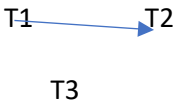
Classificare i seguenti schedule (come: NonSR, VSR, CSR). Nel caso uno schedule sia VSR oppure CSR, indicare tutti gli schedule seriali e esso equivalenti.

1. r1(x), w1(x), r2(z), r1(y), w1(y), r2(x), w2(x), w2(z)

2. r1(x), w1(x), w3(x), r2(y), r3(y), w3(y), w1(y), r2(x)

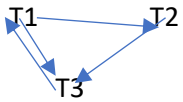
1) r1(x), w1(x), r2(z), r1(y), w1(y), r2(x), w2(x), w2(z)

Indicando al solito i conflitti con i colori, si vede che disegnando si avrebbe:



Vedendo anche che ci sono due scritture di w2 in x e z e una sola scrittura di w1 in x, possiamo dire che è certamente conflict-equivalente (dal grafo), mentre per la VSR, si avrebbero le stesse scritture finali, in quanto le dipendenze presenti hanno una lettura iniziale su z e y che non interferiscono l'una con l'altra e le stesse scritture finali su x e z, anche qui che non interferiscono.

2) r1(x), w1(x), w3(x), r2(y), r3(y), w3(y), w1(y), r2(x)

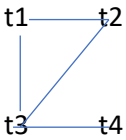


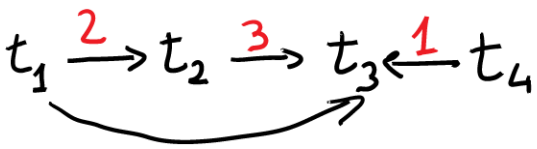
Come si vede, non è CSR essendoci un ciclo. Si vede che non è neanche VSR in quanto l'ordine delle scritture finali non viene rispettato dalle letture iniziali, portando quindi a differenti viste sul dato. Dunque, S non è né VSR che CSR.

Data lo schedule S = r1(x)w2(x)r3(x)w1(u)w3(v)r3(y)r2(y)w3(u)w4(t)w3(t) sapendo che S è conflict-serializzabile:

1. Mostrare/Spiegare se S è (o non è) view-serializzabile;
2. se S view-serializzabile, mostrare uno schedule seriale T che è view-equivalente a S, mostrando perché S and T sono view-equivalenti.

1) Il grafo dei conflitti mostra che:





Dunque, essendo conflict-serializzabile è anche view-serializzabile.

2) Partendo dal grafo dei conflitti, è possibile serializzare le transazioni.
Usiamo quindi t1 che ha due transazioni uscenti (1)
andando poi verso t2 che è raggiunto direttamente da t1 (2)
per poi usare t4, che raggiunge t3 (3)
e raggiungere appunto t3, successivamente raggiunto da t1 e t2 (4)
Quindi, è possibile serializzare le transizioni nel seguente ordine: t1, t2, t4, t3.
Lo schedule seriale quindi è: T = r1(x) w1(u) w2(x) r2(z) w4(t) r3(x) w3(v) r3(z) w3(u) w3(t).

Se invece vogliamo parlare della view-equivalenza, abbiamo:

- r3(x) che legge da w2(x)
- r1(x) r2(z) r3(x) che non leggono da scritture
- le scritture finali che sono w2(x) w4(t) w3(v) w3(u)

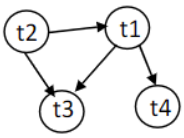
Quindi, T ha una leggi-da e le stesse scritture finali, che non hanno relazioni leggi-da.
Dunque, con la view-equivalenza si ha che S e T sono proprio view-equivalenti.

Domanda 3 (2 Punti)

Dato il seguente schedule, con grafo di conflitti in figura:

$R_2(A)\ R_2(B)\ W_1(A)\ W_2(D)\ R_3(C)\ R_1(C)\ W_3(B)\ R_4(A)\ W_3(C)$.

Quale delle seguenti affermazioni è vera?



Il grafo dei conflitti evidenza che siamo in presenza di uno schedule conflict-serializzabile, conseguentemente è anche view-serializzabile.

Se noi esaminiamo tuttavia i conflitti per bene:

- esiste una leggi-da iniziale, da R2 a W1
- esiste una leggi-da successiva, tra R2 e W3
- ci sono due scritture in possibile conflitto, W1 e W3
- esiste una leggi-da finale, tra W1 ed R4

Serializzando le transazioni, si potrebbe ipotizzare un ordine:

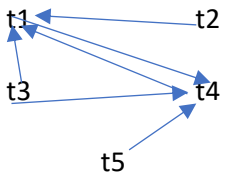
1) t2
2) t1
3) t4
4) t3

Avremo quindi: R2(A) R2(B) W2(D) W1(A) R1(C) R4(A) R3(C) W3(B) W3(C)
Formalmente quindi si vede che rimangono le stesse relazioni leggi-da e similmente sono uguali anche le scritture finali. Questo dimostra formalmente la view-equivalenza.

Indicare e motivare se lo schedule è conflict-serializzabile

$$r_4(y)w_1(z)r_2(y)w_3(x)w_1(y)r_1(x)r_3(z)w_5(z)w_5(y)w_4(z)r_4(x)$$

Letteralmente si disegna il grafo dei conflitti (si spera di aver incluso tutto, comunque esiste un ciclo):



Il grafo dei conflitti ha un ciclo dalla transazione 4 alla transazione 1, dovuta a r4(y) w1(z)… w1(y)w4(z).

Quindi, lo schedule non è conflict-serializzabile.