Guida Completa alle Transazioni nei Database Relazionali

Concetti fondamentali sulle Transazioni

Una **transazione** è una sequenza di operazioni che vengono eseguite come un'unità indivisibile. Nei database, le transazioni devono rispettare le proprietà ACID:

- **Atomicità**: la transazione viene eseguita completamente o non viene eseguita affatto
- **Consistenza**: la transazione porta il database da uno stato consistente a un altro
- **Isolamento**: l'esecuzione concorrente di transazioni deve produrre lo stesso risultato dell'esecuzione sequenziale
- **Durabilità**: gli effetti di una transazione completata devono persistere anche in caso di guasti

Serializzabilità e Gestione della Concorrenza

Concetti Chiave

- **Schedule**: sequenza di operazioni di I/O di transazioni concorrenti
- **Operazioni di base**: `r(x)` (lettura), `w(x)` (scrittura)
- **Schedule seriale**: le transazioni sono eseguite una dopo l'altra senza sovrapposizioni
- **Schedule serializzabile**: produce lo stesso risultato di uno schedule seriale

Tipi di Serializzabilità

1. View-Serializzabilità (VSR)

Due schedule S1 e S2 sono **view-equivalenti** se:

1. **Lettura iniziale**: se una transazione legge un valore originale in S1, lo legge anche in S2
2. **Lettura aggiornata**: se una transazione legge un valore scritto da un'altra transazione in S1, lo stesso accade in S2
3. **Scrittura finale**: se una transazione scrive l'ultimo valore di un dato in S1, lo stesso accade in S2

Uno schedule è **view-serializzabile** se è view-equivalente a uno schedule seriale.

2. Conflict-Serializzabilità (CSR)

Due operazioni sono in **conflitto** se:

1. Appartengono a transazioni diverse
2. Operano sullo stesso dato
3. Almeno una è un'operazione di scrittura

Due schedule S1 e S2 sono **conflict-equivalenti** se:

- Includono le stesse operazioni
- Ogni coppia di operazioni in conflitto appare nello stesso ordine in entrambi

Uno schedule è **conflict-serializzabile** se è conflict-equivalente a uno schedule seriale.

Relazioni tra VSR e CSR

- Ogni schedule CSR è anche VSR
- Non tutti gli schedule VSR sono CSR
- CSR può essere verificato in modo efficiente usando il grafo dei conflitti
- VSR è computazionalmente più difficile da verificare

Grafo dei Conflitti

Per verificare se uno schedule è CSR:

1. Crea un nodo per ogni transazione
2. Crea un arco orientato da Ti a Tj se un'operazione di Ti è in conflitto con un'operazione di Tj e Ti precede Tj
3. Se il grafo contiene cicli, lo schedule NON è CSR
4. Se il grafo è aciclico, lo schedule è CSR

Meccanismi di Controllo della Concorrenza

Locking a Due Fasi (2PL)

Il **locking a due fasi** è un protocollo che garantisce CSR:

1. **Fase crescente**: la transazione acquisisce tutti i lock necessari
2. **Fase decrescente**: la transazione rilascia i lock

Tipi di lock:

- **Lock condiviso** (`r_lock`): per operazioni di lettura
- **Lock esclusivo** (`w_lock`): per operazioni di scrittura

Caratteristiche:

- 2PL garantisce CSR
- CSR non implica 2PL

- Evita il deadlock impedendo l'acquisizione di nuovi lock dopo averne rilasciato uno

Recovery Management (Gestione dei Guasti)

Tipi di Guasti

- **Guasti di sistema (soft)**: errori di programma, crash, cadute di tensione
- **Guasti di dispositivo (hard)**: guasti su dispositivi di memoria secondaria

Tecniche di Recovery

1. **DUMP**: copia periodica dell'intero database su memoria stabile
2. **Log**: registro delle operazioni con informazioni per annullare o ripetere le operazioni
3. **Checkpoint**: punto del log dove tutte le transazioni attive sono salvate

Struttura del Log

- **B(Ti)**: inizio della transazione Ti
- **C(Ti)**: commit della transazione Ti
- **A(Ti)**: abort della transazione Ti
- **U(Ti,O,B,A)**: update della transazione Ti sull'oggetto O dal valore B al valore A
- **I(Ti,O,A)**: insert della transazione Ti dell'oggetto O con valore A
- **D(Ti,O,B)**: delete della transazione Ti dell'oggetto O con valore B
- **CK(T1,T2,...)**: checkpoint con transazioni attive T1, T2, ...

Ripresa a Caldo (Warm Restart)

Procedimento:

1. Trovare l'ultimo checkpoint nel log
2. Costruire gli insiemi UNDO e REDO:
   - **UNDO**: transazioni senza commit (da annullare)
   - **REDO**: transazioni con commit (da ripetere)
3. Ripercorrere il log all'indietro eseguendo le operazioni UNDO
4. Ripercorrere il log in avanti eseguendo le operazioni REDO

Esempi di Esercizi Risolti

Esempio 1: Analisi di Conflict-Serializzabilità

Considerando lo schedule: `S = r2(x) r1(x) w3(t) w1(x) r3(y) r4(t) r2(y) w2(z) w5(y) w4(z)`

Analisi dei conflitti:

1. Identificare le coppie in conflitto e tracciare il grafo
2. Verificare se il grafo ha cicli
3. Se non ci sono cicli, lo schedule è CSR

In questo caso, il grafo dei conflitti non presenta cicli, quindi S è CSR.

Lo schedule seriale equivalente sarebbe: `r2(x) r2(y) w2(z) r1(x) w1(x) w3(t) r3(y) r4(t) w4(z) w5(y)`

Esempio 2: Recovery Management

Dato il log:

```
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, A4), 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(T7), guasto
```

Procedimento:

1. Trovare l'ultimo checkpoint: `CK(T1, T4, T5, T6)`
2. Costruire gli insiemi:
   - UNDO = {T1, T5, T6, T7, T8} (transazioni senza commit)
   - REDO = {} (nessuna transazione ha fatto commit dopo l'ultimo checkpoint)
3. Eseguire le operazioni UNDO in ordine inverso:
   - U(O3, B7)
   - U(O6, B6)
   - U(O5, B5)
   - U(O4, B4)
   - U(O3, B3)
   - Delete di O1

Esempio 3: View-Serializzabilità

Per lo schedule: `S = r1(x) r1(y) r2(y) w2(z) w1(z) w3(x)`

Per determinare se è VSR:

1. Esaminare le relazioni leggi-da
2. Verificare le scritture finali
3. Confrontare con possibili schedule seriali

In questo caso, S è VSR perché le relazioni leggi-da e le scritture finali possono essere mantenute in uno schedule seriale.

Esercizio 2: Conflict-Serializzabilità

Lo schedule in esame è:

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

Analisi del Grafo dei Conflitti

Per determinare se questo schedule è conflict-serializzabile, dobbiamo:

1. Identificare tutte le coppie di operazioni in conflitto
2. Costruire il grafo dei conflitti
3. Verificare se il grafo contiene cicli

Le coppie in conflitto sono:

- r1(x) e w2(x): conflitto read-write su x → T1 → T2
- w2(x) e r3(x): conflitto write-read su x → T2 → T3
- w1(u) e w3(u): conflitto write-write su u → T1 → T3
- r3(y) e r2(y): non c'è conflitto (entrambe sono letture)
- operazioni su t: non creano cicli nel grafo

Il grafo dei conflitti risulta: T1 → T2 → T3 ← T4

Poiché non ci sono cicli nel grafo, lo schedule è conflict-serializzabile.

Schedule Equivalente

Lo schedule seriale conflict-equivalente sarebbe:

```
r1(x) w1(u) w2(x) r2(y) w4(t) r3(x) w3(v) r3(y) w3(u) w3(t)
```

Questo schedule rispetta l'ordine T1 → T2 → T4 → T3 imposto dal grafo dei conflitti.

Osservazione

Se si spostasse w1(u) dopo w3(u), si creerebbe un ciclo nel grafo (T1 → T2 → T3 → T1), rendendo lo schedule non serializzabile.

Esercizio 3: Analisi di Schedule

Esempi di Schedule da Classificare

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

Analisi:

- Conflitti: w1(x) → r2(x), w1(y) → nessun conflitto, r2(x) → w2(x)
- Grafo: T1 → T2
- Non ci sono cicli, quindi è CSR
- È anche VSR (poiché ogni CSR è anche VSR)
- Lo schedule seriale equivalente è: eseguire completamente T1, poi T2

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

Analisi:

- Conflitti: w1(x) → w3(x), w3(x) → r2(x), r3(y) → w3(y), w3(y) → w1(y), w1(y) → nessun conflitto, w1(x) → r2(x)
- Grafo: T1 → T3 → T2 → T1 (ciclo)
- C'è un ciclo, quindi NON è CSR
- Non è neanche VSR perché l'ordine delle scritture finali non viene rispettato dalle letture

Esercizio 4: Verifica della View-Serializzabilità

Ricordiamo che uno schedule è view-serializzabile se:

1. Mantiene le stesse relazioni "legge-da" di uno schedule seriale
2. Mantiene le stesse scritture finali

Schedule da Classificare:

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

Analisi:

- Non è VSR: invertendo w1(y) con r2(y) o w2(x) con r1(x) si alterano le relazioni di dipendenza.
- Le operazioni sono interconnesse in modo tale che cambiando l'ordine cambierebbe il risultato.

2. `r1(x), r2(y), w1(x), w1(y), r2(x) w2(x)`

Analisi:

- Non è VSR: ci sono diverse relazioni lettura/scrittura che creano dipendenze.
- Cambiando l'ordine, il risultato finale cambierebbe.

3. `r1(x), r1(y), r2(y), w2(z), w1(z), w3(x)`

Analisi:

- È VSR: non ci sono letture/scritture concorrenti sullo stesso oggetto.
- L'unica scrittura su x è w3(x), su y non ci sono scritture, su z ci sono w2(z) e w1(z) ma sono su due transazioni diverse.

- Le transazioni possono essere eseguite in qualsiasi ordine senza alterare il risultato finale.

4. `r1(y), r1(y), w2(z), w1(z), w3(x), w1(x)`

   Analisi:
   - Non è VSR: ci sono due scritture con T1 per x e z e due scritture con T3 per x e z.
   - Invertendo l'ordine delle transazioni, l'ordine delle scritture finali cambierebbe.

Esercizio 5: Problema del Deadlock in 2PL

Considerando uno schedule con potenziale deadlock:

Analisi:

- w1(u) genera conflitti con altre transazioni attive
- Quando si tenta di eseguire w1(u), si crea un ciclo nel grafo delle attese
- Per risolvere il problema, occorre "rompere" il ciclo
- Opzioni: fare commit o abort di T2 o T3 per rimuovere il nodo dal grafo delle transazioni attive

Questo esempio illustra come l'implementazione del protocollo 2PL possa portare a situazioni di deadlock che richiedono interventi specifici per essere risolte.

Concetti fondamentali da ricordare

1. **Conflitto**: due operazioni di transazioni diverse sullo stesso dato, di cui almeno una è una scrittura
2. **CSR (Conflict-Serializable)**:
   - Uno schedule è CSR se il suo grafo dei conflitti è aciclico
   - Può essere verificato in tempo polinomiale
   - Ogni CSR è anche VSR
3. **VSR (View-Serializable)**:
   - Mantiene le stesse relazioni "legge-da" e scritture finali di uno schedule seriale
   - Più difficile da verificare (problema NP-completo)
   - Non tutti gli schedule VSR sono CSR
4. **2PL (Two-Phase Locking)**:
   - Garantisce CSR ma è più restrittivo
   - Diviso in fase crescente (acquisizione lock) e decrescente (rilascio)
   - 2PL $\Rightarrow$ CSR, ma CSR $\not\Rightarrow$ 2PL
5. **Verifica pratica della serializzabilità**:
   - Costruire il grafo dei conflitti
   - Verificare se contiene cicli
   - Se è aciclico, l'ordine topologico del grafo fornisce uno schedule seriale equivalente

Questi concetti sono fondamentali per garantire l'isolamento delle transazioni e la correttezza dei risultati in un ambiente di database concorrente.