

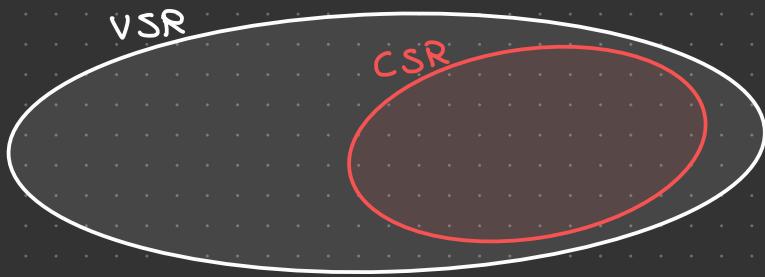
VSR → Calcolo di 2 Insiemi: - Legge-Da
- Scrivere finali

2 Problemi: - Ipotesi Commit Proiezione
- Complessità Computazionale del test

CSR → Individuazione & Analisi Conflitti

Si Riduce la Complessità ma c'è Sempre il problema dell'
Ipotesi

RELAZIONE TRA LE 2:



IPOTESI DI COMMIT PROIEZIONE

Ciò che Rende **INAPPLICABILE** Nel Mondo Reale

Controllo della Concorrenza che NON Richiedono di Conoscere l'ESITO delle Transaz.

① Tecnica di PostgreSQL \Rightarrow Viste proprie delle Basi di Dati,
Chiamate anche **TIMESTAMP con SCRITTURE BUFFERIZZATE**

Oppure del LOCKING a 2 FASI, ovvero il **2PL**

2PL:

- Necessita un Meccanismo per la **Gestione dei LOCK**
- Politica di Concessione dei LOCK Sulle **RISORSE**
- Regole Basate Sui LOCK per **Garantire la SERIALIZZABILITÀ**

Primitive di LOCK \Rightarrow Riuscire a Bloccare le **RISORSE** Su cui Vuole AGIRE (per fare OPERAZIONI di Lettura o Scrittura)

■ **R-Lock_x(x)** \rightarrow Transazione T_x che Richiede il **LOCK** su una Risorsa CONDIVISA x in **Lettura**

■ **W-Lock_x(x)** \rightarrow Transazione T_x che Richiede il **LOCK** su una Risorsa CONDIVISA x in **Scrittura**

■ **Unlock_x(x)** \rightarrow Libera la Risorsa CONDIVISA x

REGOLE PER L'UTILIZZO:

① Ogni lettura fatta da T_k su x deve essere preceduta da una primitiva $R_LOCK_k(x)$ e seguita da una primitiva $UNLOCK_k(x)$.

Sono ammesse più R_LOCK contemporanee sulla stessa risorsa x . $\rightsquigarrow R_LOCK$ è condiviso

② Ogni scrittura da T_k su x deve ESSERE preceduta da una $W_LOCK_k(x)$ e seguita da una $UNLOCK_k(x)$ e però NON sono ammesse più W_LOCK sulla stessa risorsa. $\rightsquigarrow Lock$ esclusivo

Quando una transazione T_k segue le 2 regole è ben formata. Rispetto al locking

COME VENGONO CONCESSI?

Bisogna mantenere per ciascuna risorsa x le seguenti info:

* STATO della RISORSA $\rightarrow S(x) \in \{LIBERO, R_LOCK, W_LOCK\}$

* TRANSAZIONI $R_LOCK \rightarrow C(x) \in \{T_1, \dots, T_n\}$ con R_LOCK su x

Richiesta Statuto	LIBERO	R-Lock	W-Lock
R-lock	OK! $s(x) = R_lock$ $c(x) = \{T_K\}$	OK! $s(x) = R_lock$ $c(x) = c(x) \cup \{T_K\}$	OK! Metto in Attesa fino a che tolgo lock
W-Lock	OK! $s(x) = W_lock$	POSSIEDO GIÀ LA RISORSA?	OK! Attesa
Unlock	ERRORE!	OK! $c(x) = c(x) \setminus T_K$ e se $c(x) = \emptyset$ $s(x) = LIBERO$ e si VERIFICA la coda di ATTESA	OK! $s(x) = LIBERO$ vado a Vedere la coda

SI IF $|c(x)| = 1 \wedge T_K \in c(x)$ allora $s(x) = W_lock$

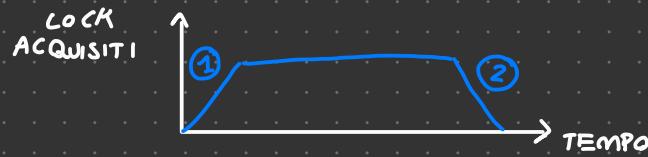
"ho fatto un UPGRADE del lock"

NO Metto in Attesa il W-Lock e non faccio NESSUNA operazione

SERIAZZABILITÀ SECONDO 2PL.

Una Transazione dopo Aver Rilasciato un LOCK NON può Acquisirne Altri

2 Phase Locking



① fase di ACQUISIZIONE

② fase di RILASCIO

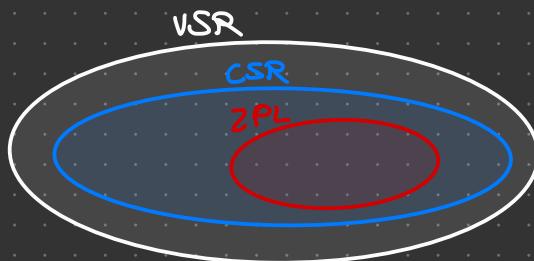
PERDITA DI AGGIORNAMENTO:

Si può VERIFICARE IL DEADLOCK

Per Rimuovere l'ipotesi di commit Proiezione si Aggiunge una Condizione Aggiuntiva:

"Una Transazione può RILASCIARE i Lock Solo Quando ha eseguito CORRETTAMENTE un Commit o un ROLLBACK a Seguito di un ABORT"

Che definisce così 2PL-Stretto



2PL e CSR:

④ Se $S \in 2PL \Rightarrow S \in CSR$

Per Assurdo $S \in PL \wedge \underbrace{S \notin CSR}$

E' una Coppia di Conflitti Critici che Porterebbe ad un DEADLOCK

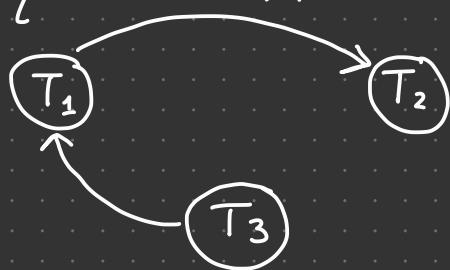
$w_1(x) R_2(x) \dots w_z(y) R_1(x)$

Ma non è 2PL visto che dopo il rilascio ha fatto un'altra acquisizione di un lock.

② $\exists S \in CSR^* \text{ } S \notin 2PL$

S: $R_1(x) W_1(x) R_2(x) W_2(x) R_3(y) W_1(y)$

$\text{CONFLITTI}(S) = \{(R_1(x), W_2(x)), (W_1(x), R_2(x)), (W_1(x), W_2(x)), (R_3(y), W_1(y))\}$



È Aciclico Quindi è CONFLICT SERIALIZZABILE MA NON 2PL visto che c'è un rilascio del lock che poi viene richiesto e non è Accettabile

Blocco Critico \Rightarrow Nell'Esecuzione di 2 Transazioni Concorrenti: T_1, T_2 Abbiamo che entrambe hanno Bloccato delle Risorse x, y Per cui T_2 è in Attesa di y e T_2 di x

Ma Quanto Spesso Si Verifica? Dipende dal #Risorse e dalle Lunghezza delle Transazioni

RISORSA DEFINITA CON GRANULARITÀ DI TUPLA, IL NUMERO
MEDIO DI TUPPE IN UNA TABELLA

$$P(\text{Blocco Critico} = 2) = \frac{1}{n^2}$$

COME LO RISOLVO?

- ① Imposto dei **TIMEOUT** prima dell' EVENTUALE ABORT
- ② Devo **Prevenire**:

- Una Transazione Blocca Tutte Le Risorse di cui ha Bisogno All'Inizio diminuendo però di Molto le Transazioni.
- Una Transazione T_i va ad Acquisire un TIMESTAMP ts_i All' Inizio dell' Esecuzione.

T_i può Attendere una RISORSA BLOCCATA da T_j Solo se $T_i < T_j$ Attualmente viene fatto $\text{ABORT}(T_i)$

- ③ **Rilevamento:** Analisi periodica della Tabella di Lock per Verificare Se c'è un Blocco Critico, si Risolve facendo un ABORT [Quella IMPLEMENTATA REALMENTE]

STARVATION: è un Problema **poco probabile**, viene Risolto con Tecniche Molto Simili a Quelle per il Blocco Critico.

Si ANALIZZA la Tab. delle Relazioni d'Attesa, in cui si indica Quando viene Messa in Attesa T_i , e si Verifica da Quanto una Transazione è in Attesa e si può decidere SOSPENDERE

↳ CONCESSIONE di R-LOCK su RISORSA per PERMETTERE W-LOCK

Penso Rinunciare in TUTTO/PARTE alla Gestione della Concorrenza, selezionare la TIPOLOGIA di Anomalie che possono ESSERE Accettate

LIVELLO DI ISOLAMENTO:

■ SERIALIZZABLE → Elimina TUTTE le Anomalie

■ REPEATABLE READ → Rimane Invecemente fantasma

■ READ COMMITTED → Rimane Lettura Inconsistente & *Fantasma

■ READ UNCOMMITTED → Risolve Solo Perdita Aggiornamento

Tutti i LVL Richiedono il 2PL Stretto per le Operazioni In Scrittura

■ SERIALIZZABLE → 2PL Anche per letture

■ REPEATABLE READ → 2PL per le letture ma Considera una Risorsa a livello di TUPLA e non Tabella

■ READ COMMITTED → Richiede R-LOCK Condivisi e non 2PL Stretto

■ READ UNCOMMITTED → non Applica i lock in lettura