

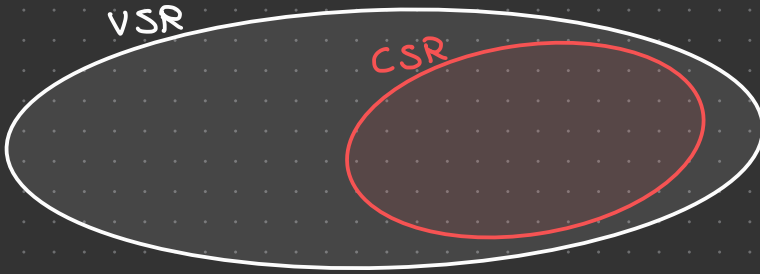
VSR \rightarrow Calcolo di 2 Insiemi: - Legge-De
- Scritture finali

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

CSR \rightarrow 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 Transac.

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:

- Necessito 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)

■ **$R_LOCK_k(x)$** \rightarrow Transazione T_k che Richiede il LOCK su una Risorsa CONDIVISA x in **Lettura**

■ **$W_LOCK_k(x)$** \rightarrow Transazione T_k che Richiede il LOCK su una Risorsa CONDIVISA x in **Scrittura**

■ **$Unlock_k(x)$** \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 . \leadsto 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. \leadsto 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 $\rightarrow S(x) \in \{\text{LIBERO}, R_LOCK, W_LOCK\}$

TRANSAZIONI R-LOCK $\rightarrow C(x) \in \{T_1, \dots, T_n\}$ con R-lock su x

Stato Richiesta	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

SERIALIZZABILITÀ SECONDO 2PL:

Una Transazione dopo Aver Rilasciato un Lock NON può Acquisirne Altri

2 Phase Locking

LOCK
ACQUISITI



① fase di ACQUISIZIONE

② fase di RILASCIO

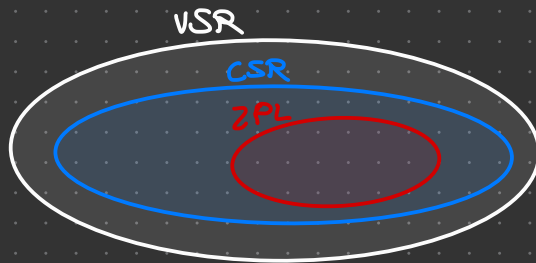
PERDITA DI AGGIORNAMENTO:

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:

① Se $S \in 2PL \Rightarrow S \in CSR$

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

È una Coppia di Conflitti Ciclici che Porterebbe ad un DEADLOCK

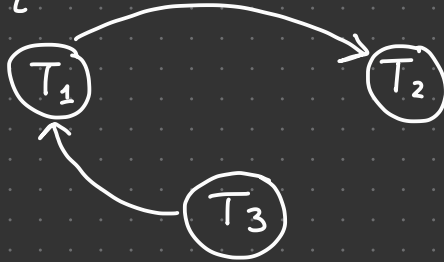
$W_1(x) R_2(x) \dots W_2(y) R_1(x)$

Ma NON è 2PL visto che dopo il RILASCIO ha fatto un'altra acquisiz. di un LOCK.

② $\exists S \in CSR \wedge S \notin 2PL$

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

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



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_1 è in Attesa di y e T_2 di x

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

RISORSA DEFINITA CON GRANULARITÀ DI TUPLA, IL NUMERO MEDIO DI TUPLE 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 $T_i < T_j$ Altrimenti Viene fatto **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 indice Quando viene Messa in Attesa T_i , e si Verifica da Quanto una Transazione è in Attesa e **si può decidere SOSPENDERE**

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

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

LIVELLO DI ISOLAMENTO:

/// SERIALIZABLE → Elimina TUTTE Anomalie

/// REPEATABLE READ → Rimane Inseguimento 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

/// SERIALIZABLE → 2PL Anche per letture

/// REPEATABLE READ → 2PL per le Letture ma Considera una Risorsa a Lvl di TUPLA e NON Tabella

/// READ COMMITTED → Richiede R-LOCK Condivisi e NON 2PL Stretto

/// READ UNCOMMITTED → NON Applica i LOCK in lettura