CAP. 2 GESTIONE DELLE TRANSAZIONI

LE TRANSAZIONI

Una transazione identifica una unità elementare di lavoro svolta da un'applicazione, cui si vogliono associare particolari caratteristiche di correttezza, robustezza e isolamento.
Un sistema transazionale è un sistema che mette a disposizione un meccanismo per la definizione e l'esecuzione di transazioni con le caratteristiche suddette.

SPECIFICA DELLE TRANSAZIONI: COMMIT E ROLLBACK

Ogni transazione è specificata racchiudendo la sequenza di operazioni che la compongono all'interno di una coppia di istruzioni che ne specificano l'inizio e la conclusione.
L'istruzione COMMIT WORK specifica il fatto che si richiede una conclusione positiva della transazione e che quindi tutti gli aggiornamenti debbono essere salvati nella base di dati.
La ROLLBACK WORK si utilizza nel caso in cui dopo aver effettuato alcune operazioni le condizioni verificate portano il programma stesso a stabilire che gli aggiornamenti debbano essere annullati.
In assenza di istruzioni START TRANSACTION il sistema assume di solito che ciascuna istruzione da sola costituisca una transazione (MODALITA' AUTOCOMMIT).
Una transazione BEN FORMATA prevede un inizio BEGIN o START TRANSACTION e una fine con un'ideale END TRANSACTION nel cui corso viene eseguito solo uno dei due comandi COMMIT o ROLLBACK e in cui non avvengono operazioni di accesso e/o modifica alle base dei dati , successive all'esecuzione del comando di COMMIT o ROLLBACK.

PROPRIETA' "ACIDE" DELLE TRANSAZIONI

Le transazioni devono godere di proprietà particolari dette proprietà acide: atomicità, consistenza, isolamento e persistenza.

ATOMICITA'. L'atomicità rappresenta il fatto che una transazione è un'unità indivisibile di esecuzione.
Se durante l'esecuzione delle operazioni si verifica un errore, allora il sistema deve essere in grado di ricostruire la situazione esistente all'inizio della transazione, disfacendo il lavoro svolto dalle istruzioni eseguite fino a quel momento (UNDO).
Viceversa, dopo l'esecuzione del COMMIT, il sistema deve assicurare che la transazione lasci la base di dati nel suo stato finale ; ciò può comportare di rifare il lavoro svolto (REDO).
In questo modo l'operazione di COMMIT fissa il momento in cui la transazione va a buon fine.
Quando viene eseguito il comando ROLLBACK WORK la situazione è simile a un suicidio della transazione, ma il sistema può anche decidere di UCCIDERE una transazione che non riesce ad arrivare a corretto compimento.
Per indicare un fallimento di transazione (suicidio ed omicidio) si usa il termine ABORT.

CONSISTENZA. La consistenza richiede che l'esecuzione della transazione non violi i vincoli di integrità definiti sulla base di dati.
Quando il sistema rileva che una transazione sta violando uno dei vincoli, il sistema interviene per annullare la transazione o per correggere la violazione del vincolo.

La verifica di vincoli di integrità di tipo IMMEDIATO può essere fatta nel corso della transazione, mentre quella di tipo DIFFERITO viene effettuata alla conclusione della transazione dopo che l'utente ha richiesto il COMMIT.

ISOLAMENTO. L'isolamento richiede che l'esecuzione di una transazione sia indipendente dalla contemporanea esecuzione di altre transazioni.
In particolare si richiede che il risultato dell'esecuzione concorrente di un insieme di transazioni sia analogo al risultato che le stesse otterrebbero se eseguite da sole.
L'isolamento ha anche l'obiettivo di rendere l'esito di ciascuna transazione indipendente da tutte le altre.

PERSISTENZA. La persistenza invece richiede che l'effetto di una transazione che ha eseguito il COMMIT correttamente non venga più perso.

FIGURA 2.1 Componenti di un DBMS coinvolti nella gestione di interrogazioni e transazioni

CONTROLLO AFFIDABILITA'. Il controllo dell'affidabilità garantisce: l'atomicità e la persistenza. In pratica garantisce che le transazioni non vengano lasciate incomplete e che gli effetti di ciascuna transazione siano mantenuti in modo permanente.
Il controllore svolge il proprio compito attraverso il LOG, un archivio persistente su cui registra le varie azioni svolte dal DBMS.

ARCHITETTURA DEL CONTROLLORE

Il controllore è responsabile di realizzare i comandi transazionali BEGIN TRANSACTION, COMMIT, ROLLBACK e di realizzare le operazioni di ripristino detto RIPRESA A CALDO e RIPRESA A FREDDO.
Inoltre esso riceve richieste di accessi a pagine in lettura e scrittura che passa al buffer manager e genera altre richieste di lettura e scrittura di pagine necessarie a garantire la robustezza e la resistenza ai guasti.
Il controllore predispone i dati necessari per eseguire i meccanismi di ripristino dei guasti realizzando azioni di CHECKPOINT e di DUMP.

FIGURA 2.2

MEMORIA STABILE. Per poter operare il controllore deve disporre di memoria stabile, cioè resistente ai guasti. La memoria stabile è un'astrazione perché nessuna memoria può essere associata a una probabilità nulla di fallimento, tuttavia meccanismi di replicazione possono renderla prossima allo zero. Un guasto della memoria stabile viene considerato catastrofico in quanto non previsto.
Essa viene realizzata in modi diversi.
Es.: una unità a nastro, una coppia di dispositivi come un'unità a nastro e un disco su cui vengono scritte le info.
La più classica realizzazione utilizza due unità a disco disposte a specchio:
essi contengono la stessa info e devono essere scritte con una "scrittura attenta" su entrambi i dischi per definirsi riuscito.

ORGANIZZAZIONE DEL LOG

Il log è un file sequenziale di cui è responsabile il controllore, scritto in memoria stabile e contiene info ridontante e permette di ricostruire il contenuto della base di dati a seguito di guasti. Sul log

vengono registrate le azioni svolte dalle varie transazioni nell'archivio temporale di esecuzione delle azioni stesse.
I record del log sono di due tipo: di transazione e di sistema.
Quelli di transazione registrano le attività svolte da ciascuna transazione nell'ordine in cui esse vengono effettuate.
I record di sistema indicano l'effettuazione di operazioni specifiche del controllore dell'affidabilità chiamate DUMP e CHECKPOINT.

SCRITTURA DEI RECORD NEL LOG. I record di log sono:

- record di BEGIN , COMMIT e ABORT che contengono oltre al tipo di record anche l'identificativo T della transazione
- record di UPDATE che contengono l'identificativo T , l'identificativo O dell'oggetto su cui avviene l'update e poi 2 valori BS e AS che descrivono rispettivamente il valore dell'oggetto O prima della modifica (BEFORE STATE) e dopo la modifica (AFTER STATE).
- Record di INSERT e di DELETE analoghi a quelli di update da cui si differenziano per l'assenza nei primi del BE e nei secondi dell'AS.

Questi record consentono di disfare e rifare le rispettive azioni sulla base di dati.
UNDO: per disfare un'azione sull'oggetto O è sufficiente ricopiare nell'oggetto O il valore BS; l'insert viene disfatto cancellando O.
REDO: per rifare un'azione sull'oggetto O basta ricopiare in O il valore AS; il dolete viene rifatto cancellando O.

CHECKPOINT E DUMP.

Un CHECKPOINT è un'operazione che viene svolta periodicamente dal gestore dell'affidabilità con l'obiettivo di registrare quali transazioni sono attive.
Questa operazione è costituita dai seguenti passi:

1. si sospende l'accettazione di operazioni di scrittura, COMMIT o ABORT, da parte di ogni transazione
2. si trasferiscono in memorie di massa tutte le pagine sporche del buffer relative a transazioni che hanno già effettuato il OCMMIT.
3. si scrive in modo sincrono (FORCE) nel log un record di CHECKPOINT che contiene gli identificatori delle transazioni attive
4. si riprende l'accettazione delle operazioni sopra sospese.

Un DUMP è una copia completa della base di dati , che viene normalmente effettuata in mutua esclusione con tutte le altre transazioni quando il sistema non è operativo.
La copia viene memorizzata su memoria stabile (nastro) ed è detta BACKUP.
Dopo l'operazione di DUMP viene scritto nel log un record di DUMP che segnala la presenza di una copia fatta in un determinato istante, dopodiché il sistema può tornare al suo funzionamento normale.

ESECUZIONE DELLE TRANSAZIONI E SCRITTURA DEL LOG.

Il controllore dell'affidabilità deve garantire che siano eseguite 2 regole per consentire di ripristinare la correttezza della base di dati a fronte di guasti.
- LA REGOLA WAL (WRITE AHEAD LOG) impone che la parte BEFORE STATE (BS) dei record di un log venga scritta nel log prima di effettuar la corrispondente operazione sulla base

di dati. Questa consente di disfare la scrittura già effettuata in memoria di massa da parte di una transazione che non ha ancora effettuato un OCMMIT.
- LA REGOLA DI COMMIT – PRECEDENZA impone che la parte AFTER STATE (AS) dei record di un log venga scritta nel log prima di effettuare il COMMIT. Questa consente di rifare la scrittura già decisa da una transazione che ha effettuato il COMMIT ma le cui pagine modificate non sono ancora state trascritte dal buffer manager in memoria di massa.

GESTIONE DEI GUASTI

Dal punto di vista di un DBMS i guasti si suddividono in 2 classi:

- GUASTI DI SISTEMA sono indotti da "bachi software" ad esempio del sistema operativo o da interruzioni del funzionamento dei dispositivi. Si traducono in una perdita del contenuto della memoria centrale, mantenendo invece valido il contenuto della memoria di massa.
- GUASTI DI DISPOSITIVO sono relativi ai dispositivi di gestione della memoria di massa. Si traducono in una perdita del contenuto della base di dati, ma non del log. La perdita del contenuto del log è classificata come evento catastrofico e non vi è alcun rimedio.

Il modello ideale in cui ci poniamo è detto di FAIL-STOP: quando il sistema individua un guasto di qualsiasi tipo, esso forza immediatamente un arresto completo delle transazioni ed il successivo ripristino del corretto funzionamento del sistema operativo (BOOT).
Quindi viene attivata una procedura di ripresa, detta A CALDO (WARM RESTART) nel caso di guasto di sistema e A FREDDO (COLD RESTART) nel caso di guasto di dispositivo.

FIGURA 2.5

FUNZIONAMENTO NORMALE

All'atto del guasto esisteranno transazioni potenzialmente attive e delle quali non si conosce se abbiano ultimato le loro azioni sulla base di dati.
Alcune di esse hanno effettuato il COMMIT e per loro è necessario RIFARE le azioni al fine di garantire la persistenza; altre non hanno effettuato il COMMIT e per loro è necessario DISFARE le azioni in quanto non è noto quali azioni siano state realmente effettuate e vi è l'impegno di lasciare la base di dati nello stato precedente all'esecuzione della transazione.

RIPRESA A CALDO. E' articolata in quattro fasi successive:

1. Si accede all'ultimo blocco del log, che era corrente all'istante del guasto, e si ripercorre all'indietro il log fino al record di CHECKPOINT
2. Si decidono le transazioni da rifare o disfare. Si costruiscono due insiemi detti di UNDO e di REDO contenenti identificativi di transazioni.
3. Si percorre all'indietro il log disfacendo le transazioni nel set di UNDO risalendo fino alla prima azione della transazione più vecchia bei due insiemi di UNDO e REDO; questa azione potrebbe precedere il record di CHECKPOINT nel log.
4. Infine si applicano le azioni di REDO nell'ordine in cui sono registrate nel log. Così viene replicato il comportamento delle transazioni originali.

RIPRESA A FREDDO. E' articolata in tre fasi successive:

1. Si accede al DUMP e si ricopia selettivamente la parte deteriorate della base di dati. Si accede anche al più recente record di DUMP nel log.

2. Si ripercorre in avanti il log applicando alla parte deteriorata sia le azioni sulla base di dati sia quelle di COMMIT o ABORT riportandosi nella situazione precedente al guasto.
3. Infine si svolge una ripresa a caldo.

CONTROLLO DI CONCORRENZA

Un'unità di misura che viene solitamente utilizzata per caratterizzare il carico applicativo di un BDMS è il numero di transazioni al secondo (TPS) che devono essere gestite dal DBMS per soddisfare le applicazioni.
E' indispensabile che le transazioni di un DBMS vengono eseguite concorrentemente; infatti solo la concorrenza delle transazioni consente un uso efficiente del DBMS massimizzando le TPS e minimizzando i tempi di risposta.

ARCHITETTURA

Il controllore della concorrenza riceve le richieste di accesso ai dati e decide se autorizzarle o meno,eventualmente riordinandole.
Poiché esso stabilisce anche l'ordine degli accessi viene anche chiamato *scheduler.*
FIGURA 2.6

ANOMALIE DELLE TRANSAZIONI CONCORRENTI

L'esecuzione concorrente di varie transazioni può causare problemi di correttezza o *anomalie.*
Cinque casi tipici sono i seguenti.
PERDITA DI AGGIORNAMENTO
LETTURA SPORCA
LETTURE INCONSISTENTI
AGGIORNAMENTO FANTASMA
INSERIMENTO FANTASMA

TEORIA DEL CONTROLLO DI CONCORRENZA

Una transazione è definita come una sequenza di azioni di lettura o scrittura caratterizzate dallo stesso indice. Per quanto concerne la teoria del controllo di concorrenza ogni transazione è un oggetto sintattico, di cui si conoscono soltanto le azioni di ingresso/uscita.
Ad es. una transazione t1 è rappresentata dalla sequenza *t1 : r1(x) r1(y) w1(x) w1(y)*
Uno schedule rappresenta la sequenza di operazioni di ingresso/uscita presentate da transazioni concorrenti. Uno schedule S1 è una sequenza del tipo *S1 : r1(x) r2(z) w1(x) w2(z)*
Le operazioni compaiono nello schedule seguendo l'ordine temporale con cui sono eseguite nella base di dati.
Il controllo di concorrenza ha la funzione di accettare alcuni schedule e rifiutarne altri, per ciò il modulo che gestisce il controllo di concorrenza è detto anche *scheduler.*
Lo schedule si dice una *commit-proiezione* della esecuzione delle operazioni di ingresso/uscita, contenente le sole azioni di transazioni che producono un commit.
Questa assunzione è funzionale ed è inaccettabile nella pratica in quanto lo schedulerdeve decidere se accettare o meno le azioni di una transazione senza conoscere il suo esito finale.
Definiamo *seriale* uno schedule in cui le azioni di tutte le transazioni compaiono in sequenza, senza essere inframmezzate da istruzioni di altre transazioni.
L'esecuzione della commit-proiezione di uno schedule è corretta quando produce lo stesso risultato prodotto da un qualunque schedule seriale delle stesse transazioni. In questo caso diremo che è *serializzabile*.

VIEW-EQUIVALENZA

Due schedule vengono detti *view-equivalenti* se possiedono la stessa relazione "legge-da" e le stesse scritture finali. Una operazione di scrittura *wi(x)* viene detta una *scrittura finale* se è l'ultima scrittura dell'oggetto x che appare nello schedule.
Uno schedule è detto *view-serializzabile* se è view-equivalente a un generico schedule seriale.

CONFLICT-EQUIVALENZA

Una nozione di equivalenza più facilmente utilizzabile si basa sulla definizione di conflitto.
Due azioni *ai* e *aj*, con i diverso da j, si dice che ai è in *conflitto* con aj se entrambe operano sullo stesso oggetto e almeno una di esse è una scrittura. Possono esistere conflitti lettura-scrittura (*rw* o *wr*) e conflitti scrittura-scrittura (*ww*).
Si dice che lo schedule Si è *conflict-equivalente* allo schedule Sj se i due presentano le stesse operazioni e ogni coppia di operazioni in conflitto è nello stesso ordine in entrambi gli schedule.
Uno schedule è quindi *conflict-serializzabile* se esiste uno schedule seriale a esso conflict-equivalente.
L'insieme di tutti gli schedule conflict-serializzabili viene denominato CSR, ed è possibile dimostrare che la classe degli schedule CSR è strettamente inclusa in quella degli schedule VSR.
E' possibile determinare se uno schedule è conflict-serializzabile tramite il *grafo dei conflitti*.
Si può dimostrare che lo schedule è CSR se e solo se il grafo è aciclico.

LOCKING A DUE FASI

Il principale meccanismo di controllo di concorrenza usato in quasi tutti i DBMS commerciali si basa sul *locking,* tecnica in cui tutte le operazioni di lettura e scrittura devono essere protette tramite l'esecuzione di opportune primitive, *r_lock*, *w_lock* e *unlock*; lo schedule riceve una sequenza di queste primitive da parte delle transazioni e ne determina la correttezza con una semplice *ispezione* di una struttura dati. Nelle operazioni di lettura e di scrittura si devono rispettare i seguenti vincoli:

1. Ogni operazione di lettura deve essere preceduta da un *r_lock* e seguita da un *unlock*; il lock si dice in questo caso *condiviso*, perché su un dato possono essere contemporaneamente attivi più lock
2. Ogni operazione di scrittura deve essere preceduta da un *w_lock* e seguita da un *unlock*;in tal caso il lock si dice *esclusivo*, perché non può coesistere con altri lock sullo stesso dato.

Se una transazione segue queste regole si dice *ben formata rispetto al locking*.
Quando una richiesta di lock è concessa dal gestore della concorrenza si dice che la corrispondente risorsa viene *acquisita* dalla transazione richiedente; all'atto dell'unlock la risorsa viene *rilasciata*.
Quando una richiesta di lock non viene concess, la transazione richiedente viene messa in *stato di attesa* finchè la risorsa non viene sbloccata e torni ad essere disponibile.
Ogni richiesta di lock che perviene al lock manager è caratterizzata solo dall'identificativo della transazione che fa la richiesta e dalla risorsa per la quale la richiesta viene effettuata.
La politica che viene seguita dal lock manager per concedere i lock è rappresentata nella tabella dei conflitti in cui le righe identificano le richieste e le colonne lo stato della risorsa richiesta.
Locking a due fasi (2PL): Una transazione, dopo aver rilasciato un lock, non può acquisirne altri.
Come conseguenza di questo principio si possono distinguere nell'esecuzione due diverse fasi: una prima in cui si acquisiscono i lock per le risorse cui si deve accedere (fase crescente), e una seconda fase in cui i lock acquisiti vengono rilasciati (fase calante).
La classe 2PL è contenuta in CSR.
Locking a due fasi stretto (strict 2PL): I lock di una transazione possono essere rilasciati solo dopo aver correttamente effettuato le operazioni di commit/abort.

In pratica con questo vincolo i lock vengono rilasciati solo al termine della transazione, dopo che ciascun dato è stato portato nel suo stato finale.
Questa versione del 2PL viene usata dai sistemi commerciali.
I lock di predicato devono essere definiti con riferimento a condizioni (o predicati) di selezione, impedendo non solo l'accesso ai dati coinvolti ma anche la scrittura dio nuovi dati che soddisfano il predicato.

CONTROLLO DI CONCORRENZA BASATO SUI TIMESTAMP

Questo metodo è meno efficace del locking a due fasi e utilizza i timestamp, cioè identificatori associati ad ogni evento temporale che definiscono un ordinamento totale sugli eventi. Nei sistemi centralizzati, il timestamp viene generato leggendo il valore dell'orologio di sistema al momento in cui è avvenuto l'evento.
Il controllo di concorrenza mediante timestamp avviene nel seguente modo:

- A ogni transazione si associa un timestamp che rappresenta il momento di inizio della transazione;
- Si accetta uno schedale solo se esso riflette l'ordinamento seriale delle transazioni in base al valore del timestamp di ciascuna transazione.

Questo metodo impone che le transazioni risultino serializzate in base all'ordine in cui esse acquisiscono il loro timestamp.
A ogni oggetto x vengono associati due indicatori, WTM(x) e RTM(x), che sono i timestamp della transazione che ha eseguito l'ultima scrittura e di quella con ts più grande che ha letto x.
Allo scheduler arrivano richieste di accesso agli oggetti del tipo read(x,ts) o write(x,ts).Esso non fa altro che concedere o meno l'operazione seguendo questa politica:

- Read(x,ts): se ts < WTM(x) la transazione viene uccisa altrimenti la richiesta viene accettata; in tal caso RTM(x) viene aggiornato e posto pari al massimo tra RTM(x) e ts.
- Write(x,ts): se ts < WTM(x) o ts < RTM(x) la transazione viene uccisa, altrimenti la richiesta viene accettata; in tal caso WTM(x) viene aggiornato e posto pari a ts.

In pratica ogni transazione non può leggere o scrivere un dato scritto da una transazione con timestamp superiore.
Il metodo TS comporta l'uccisione di tante transazioni; inoltre questa versione è corretta sotto l'ipotesi di uso di commit-proiezioni.
Per rimuovere queste ipotesi è necessario "bufferizzare" le scritture, cioè effettuarle in memoria e trascriverle in memoria di massa dopo il commit; ciò comporta che le letture da parte di altre transazioni dei dati memorizzati nel buffer e in attesa di commit vengano a loro volta messe in attesa del commit della transazione scrivente.
Una modifica del metodo consiste nell'utilizzo delle *multiversioni*, che consiste nel mantenere diverse copie degli oggetti della base di dati, per ogni transazione che modifica la base di dati.
Con questo metodo le regole di comportamento diventano:

- Read(x,ts) : una lettura è sempre accettata. Si legge un xk siffatto: se ts > WTMn(x), allora k = n, altrimenti si prende i in modo che sia WTMi(x) < ts < WTMi+1(x).
- Write(x,ts) : se ts < RTM(x) si rifiuta la richiesta, altrimenti si aggiunge una nuova versione del dato (n cresce di uno) con WTMn(x) = ts.

MECCANISMI PER LA GESTIONE DEI LOCK

Un lock manager è un processo in grado di essere invocato da tutti i processi che intendono accedere alla base di dati. I processi per accedere alle risorse dovranno eseguire delle procedure di r_lock, w_lock e unlock, in genere caratterizzate dai seguenti parametri:

r_lock(T, x, errcode, timeout)

w_lock(T, x, errcode, timeout)
unlock(T,x)

T è l'identificativo della transazione, x è l'elemento per il quale si richiede o si rilascia il lock; errcode rappresenta un valore restituito dal lock manager e vale 0 qualora la richiesta sia soddisfatta, mentre assume un valore diverso da zero qualora la richiesta non sia stata soddisfatta.
Timeout rappresenta l'intervallo massimo di tempo che la procedura chiamante è disposta ad aspettare per ottenere il lock sulla risorsa.
Quando un processo richiede una risorsa e la richiesta può essere soddisfatta, il lock manager restituisce immediatamente il controllo al processo.
Quando invece la richiesta non può essere immediatamente soddisfatta, il sistema inserisce in una coda associata alla risorsa il processo richiedente; il processo associato alla transazione viene sospeso.
Quando infine scatta un timeout e la richiesta è insoddisfatta, la transazione richiedente può eseguire un `rollback`, cui generalmente seguirà una ripartenza della stessa transazione, oppure decidere di proseguire, richiedendo nuovamente il lock.
In quanto alle tabelle di lock si accede frequentemente, il lock manager mantiene queste informazioni in memoria centrale in modo da minimizzare i tempi di accesso.

LOCK GERARCHICO

In molti sistemi reali è possibile specificare i lock a livelli diversi: si parla allora di *granularità dei lock*. Ad esempio è possibile bloccare intere tabelle, o insiemi di tuple, o campi di singole tuple.
Per introdurre livelli diversi di granularità del locking, si opera un'estensione del protocollo di lock tradizionale, detta *lock gerarchico*.
Questa tecnica permette alle transazioni di definire in modo molto efficiente il lock di cui hanno bisogno, operando al livello prescelto della gerarchia.
La tecnica fornisce un insieme più ricco di primitive di richiesta di lock, ciascuna con gli opportuni parametri:

- XL: lock esclusivo. Corrisponde al write-lock del protocollo normale;
- SL: lock condiviso. Corrisponde al read-lock del protocollo normale.

I tre lock successivi sono specifici di questa tecnica:

- ISL: intenzione di lock condiviso. Esprime l'intenzione di bloccare in modo condiviso uno dei nodi che discende dal nodo corrente;
- IXL: intenzione di lock esclusivo. Esprime l'intenzione di bloccare in modo esclusivo uno dei nodi che discende dal nodo corrente.
- SIXL: lock condiviso, intenzione di lock esclusivo. Blocca in modo condiviso ed esprime l'intenzione di bloccare in modo esclusivo uno dei nodi che discende dal nodo corrente.

Le regole che deono essere rispettate dal protocollo sono:

1. Si richiedono i lock partendo dalla radice e scendendo lungo l'albero.
2. Si rilasciano i lock partendo dal nodo bloccato di granularità più piccola e risalendo lungo l'albero.
3. Per poter richiedere un lock SL o ISL su un nodo, si deve già possedere un lock ISL o IXL sul suo nodo padre.
4. Per poter richiedere un lock IXL, XL o SIXL su un nodo, si deve già possedere un lock SIXL o IXL sul suo nodo padre.
5. Le regole di compatibilità utilizzate dal lock manager per decidere se accettare la richiesta di lock in base allo stato del nodo e al tipo di richiesta sono riportate nella tabella in FIGURA 2.14.

BLOCCO CRITICO

Il blocco critico costituisce un problema rilevante, tipico dei sistemi concorrenti in cui si introducono condizioni di attesa.
Supponiamo di avere una transazione t1 che deve eseguire la sequenza di operazioni r(x), w(y), e una seconda transazione t2 che deve eseguire la sequenza di operazioni r(y), w(x). Con il protocollo di lock a due fasi, si può presentare il seguente schedale:
r_lock1(x),r_lock2(y),read1(x),read2(y),w_lock1(y),w_lock2(x)
A questo punto nessuna delle due transazioni riesce a procedere e il sistema è bloccato.
In conclusione la probabilità che si verifichi un blocco critico nei sistemi transazionali sia bassa, ma non nulla.
Le tecniche comunemente usate per risolvere il problema del blocco critico sono tre:

1. timeout;
2. prevenzione (deadlock prevention)
3. rilevamento (deadlock detection)

USO DEL TIMEOUT

Le transazioni rimangono in attesa di una risorsa per un tempo prefissato.Se questo tempo passa e la risorsa non è stata concessa, allora alla richiesta di lock viene data risposta negativa.
Il problema vero e proprio riguarda la scelta del valore di timeout.
Da una parte un valore elevato di timeout tende a risolvere tardi i blocchi critici, mentre un timeout troppo basso corre il rischio di rilevare come blocchi critici anche situazioni in cui una transazione sta aspettando una risorsa senza che vi sia un vero deadlock.

PREVENZIONE DEI BLOCCHI CRITICI

Vi sono alcune tecniche, una di queste prevede di richiedere il lock di tutte le risorse necessarie alla transazione in una sola volta. Essa presenta però il problema che le transazioni spesso non conoscono a priori le risorse cui vogliono accedere.
Un'altra tecnica si basa sul fatto che le transazioni acquisiscano un timestamp e consiste nel consentire l'attesa di una transazione ti su una risorsa acquisita da tj solamente se vale una determinata relazione di precedenza tra i timestamp ti e tj (es. $i < j$)
Per quanto riguarda la politica di scelta della transazione da uccidere vi sono diverse alternative.
Interrompenti (preemptive) e non interrompenti. E' interrompente se può risolvere il conflitto uccidendo la transazione che possiede la risorsa; al contrario non è interrompente una transazione può essere uccisa solo all'atto di una nuova richiesta.
Una politica può essere quella di uccidere le transazioni che hanno fatto meno lavoro. Un problema di questa politica è che può capitare che una transazione faccia accesso a un oggetto a cui accedono molte altre transazioni. Può così capitare che la transazione trovi sempre un conflitto e venga ripetutamente uccisa. In questa situazione non vi sono blocchi critici ma *blocchi individuali* (*starvation*). Per risolverla bisogna garantire che una transazione non possa essere uccisa un numero illimitato di volte. Questa tecnica non è usata nei DBMS commerciali.

RILEVAMENTO DEI BLOCCHI CRITICI

Questa tecnica prevede di non porre vincoli al sistema, ma di controllare il contenuto delle tabelle di lock, per rilevare eventuali situazioni di blocco.
Il controllo può avvenire a intervalli prefissati o quando scade un timeout di attesa di una transazione.

Il rilevamento di un blocco critico richiede di analizzare le relazioni di attesa tra le varie transazioni e di determinare se esiste un ciclo. Questa ricerca risulta abbastanza efficiente e alcuni DBMS commerciali utilizzano questa tecnica per risolvere il problema dei blocchi critici.