INF / UFG

Disciplina

Banco de Dados

Conteúdo

Controle de Concorrência:

2PL e *Deadlock*

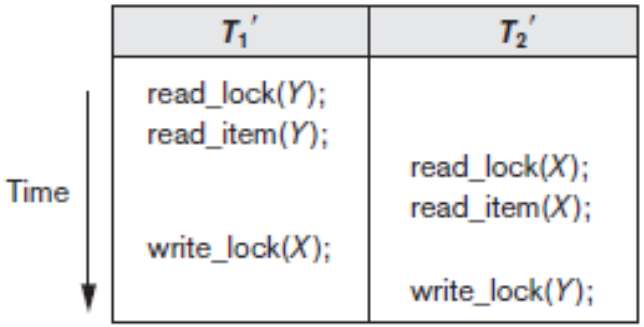
Deadlock (impasse)

Cada transação **T**, de um conjunto de duas ou mais transações, está esperando por algum item que está bloqueado por alguma outra transação desse conjunto:

>> todas as transações desse conjunto estão aguardando entre si.

Exemplo:

>>T1' está aguardando (na fila de espera) por X (X está bloqueado por T2');
>>T2' está aguardando (na fila de espera) por Y (Y está bloqueado por T1').



Deadlock: Protocolos de Prevenção

São protocolos que previnem a ocorrência de *deadlocks*.
São pessimistas.

Esses protocolos não são geralmente usados na prática, pois:

>> fazem suposições não realistas (*deadlocks* ocorrem com frequência);
>> agregam *overhead* elevado ao controle de concorrência (controles adicionais para evitar *deadlock*).

2PL Conservador
Requer que todos os bloqueios da transação ocorram antecipadamente, antes do início da transação (em geral, não é um pressuposto prático):

>> se qualquer bloqueio não puder ser obtido, nenhum bloqueio é realizado;
>> nesse caso, ocorre uma espera e nova tentativa é realizada;
>> limita concorrência.

Outra alternativa (também limita concorrência)
Ordenar todos os itens do banco de dados:

>> uma transação que for bloquear vários itens, deverá fazer na ordem dada;
>> o programador (ou o sistema) estará atento à escolha da ordem de itens.

Deadlock: Protocolos de Prevenção

Outros protocolos usam o conceito de ***timestamp***.

Timestamp.
TS(T) é um identificador que indica a ordem em que T foi iniciada:
Se TS(T1) < TS(T2), então T1 iniciou antes de T2.

Técnica esperar-ou-morrer (*wait-die*)
Técnica ferir-ou-esperar (*wound-wait*)

Vantagem:
Evitam espera indefinida (*starvation*) de uma Tx:
>> quanto mais antiga for Tx, maior a sua prioridade.

Desvantagem:
Muitos abortos podem ser provocados, sem nunca ocorrer um *deadlock*.

Deadlock: Protocolos de Prevenção

Suponha que:

>> Transação **Tx** tenta bloquear **X**,

>> **X** está bloqueado por outra transação **Ty**.

técnica esperar-ou-morrer (*wait-die*) – baseada em *timestamp*:

se ***TS(Tx) < TS(Ty)*** então
    ***Tx* espera**
senão
    início
        abort(*Tx*)
        start(*Tx*) com o mesmo *TS*
    fim

Deadlock: Protocolos de Prevenção

Suponha que:

>> Transação **Tx** tenta bloquear **X**,

>> **X** está bloqueado por outra transação **Ty**.

técnica ferir-ou-esperar (*wound-wait*) – baseada em *timestamp*:

se ***TS(Tx) < TS(Ty)*** então
início
abort(*Ty*)
start(*Ty*) com o mesmo *TS*
fim
senão
***Tx* espera**

Deadlock: Protocolos de Prevenção

Técnicas NÃO baseadas em timestamp.

Técnica sem-espera (no-*waiting*)

Se uma transação T não puder obter um bloqueio:

>> T é imediatamente abortada, e

>> T é reiniciada após certo tempo, sem checar se algum *deadlock* ocorreu.

Deadlock: Protocolos de Prevenção

Técnicas NÃO baseadas em timestamp.

Técnica espera-cautelosa (*cautious-waiting*)

```
se ( Tx deseja D ) e ( D está bloqueado por Ty ) então
    se ( Ty não está em alguma Fila-WAIT ) então
            Tx espera
    senão
            início
                    abort (Tx)
                    start (Tx)
            fim
```

Vantagem:
Se Ty já está em espera, Tx é abortada para evitar um possível ciclo de espera.

Desvantagem:
A mesma das técnicas baseadas em *timestamp*:
>> muitos abortos podem ser provocados, sem nunca ocorrer um *deadlock*.

Deadlock: Protocolos de Detecção

O sistema checa se o estado em *deadlock* realmente existe.

Quando usar protocolos de detecção...
Transações curtas, transações bloqueiam poucos itens, carga de transações é leve.

Quando usar protocolos de prevenção...
Transações longas, transações usam muitos itens, carga de transações é pesada.

O conteúdo acima é controverso, pois na prevenção ocorre mais abortos provocados pelo SGBD (o sistema presume que haverá *deadlock*), e se as transações forem longas e pesadas, a implementação do protocolo seria cara.

Deadlock: Protocolos de Detecção

Uma estratégia para a detecção de *deadlock* ...

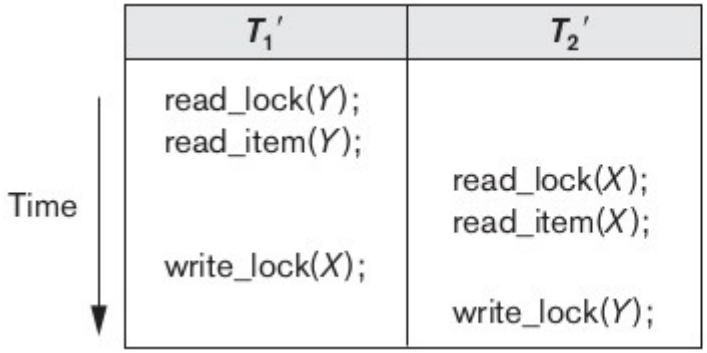
Construir um grafo ***wait-for*** :

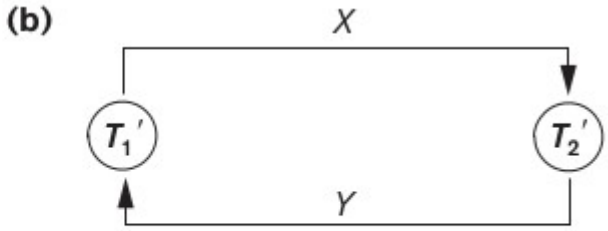
cada nó refere-se a uma transação em execução.

arco (Ti → Tj) :Ti precisa bloquear X, que está bloqueado por Tj

deadlock ocorre quando há um ciclo

Quando testar se há *deadlock*?





Deadlock: Protocolos de Detecção

Grafo *wait-for*

Quando testar se há *deadlock* no grafo ***wait-for*** ? Com que frequência ...

R1 – Checar se há *deadlock* a cada nova aresta, pode causar *overhead* excessivo.

R2 – Outros critérios:
>> número de transações em execução,
>> tempo de espera para obter bloqueio.

E se for detectado o estado de *deadlock* ? R – uma ou mais transações serão abortadas …

Qual o critério para a seleção de vítimas ?

Diretrizes: (i) evitar transações antigas (em execução por um longo tempo; (ii) evitar transações que tenham realizado muitas atualizações; (iii) selecionar transações que tenha feito poucas atualizações (possivelmente, mais jovens).

Deadlock: Outros Métodos de Tratamento

Timeout

Método simples e barato.

Se uma transação espera (para obter bloqueio) por um período maior do que um parâmetro do sistema, o sistema aborta a transação.

O aborto da transação ocorre independentemente se há ou não *deadlock*.

Starvation (fome, inanição)

Uma transação não pode prosseguir por um período indefinido de tempo, enquanto outras transações continuam normalmente.

Cenário 1
Ocorre quando o controle de concorrência é “injusto”, dando prioridade a algumas transações em detrimento de outras.

Uma solução simples é manter uma fila de prioridades, baseada na ordem em as transações solicitaram o bloqueio de dados. Ainda, a prioridade de uma transação na fila poderá aumentar baseando-se no tempo de espera.

Cenário 2
O algoritmo seleciona a mesma transação como vítima repetidamente, fazendo-a abortar e nunca terminar a execução.

Uma solução é elevar a prioridade de transações que tenham tido múltiplos abortos. As estratégias ***wait-die*** (esperar ou morrer) e ***wound-wait*** (ferir ou esperar) constituem soluções, pois uma transação abortada é reiniciada com o *timestamp* anterior.

Controle de Concorrência Baseado em 2PL

Resumo

O Uso de bloqueio, combinado com o protocolo de bloqueio de duas fases (2PL), garantem serialização.

Os escalonamentos serializáveis produzidos pelo protocolo 2PL possuem seus escalonamentos seriais equivalentes, baseando-se na ordem em que as transações bloqueiam os itens de dados que necessitam.

Se uma transação necessita de um item de dado já bloqueado, ela é forçada a esperar até que o item seja liberado.

Algumas transações podem ser abortadas e reiniciadas, devido ao problema de *deadlock*.