CI218: Controle de Concorrência

Profa. Carmem Hara

Departamento de Informática/UFPR

13 de dezembro de 2022

Roteiro

- Controle de concorrência por bloqueio
- Two-phase Locking (2PL)
- Detecção e prevenção de dealocks
- Controle de concorrência por timestamp

Cap. 14 do livro Sistema de Banco de Dados (3ed) - Silberschatz, Korth, Sudarshan

Tipos de Bloqueio

- **Compartilhado**: se uma transação T_i obtem um bloqueio compartilhado sobre o item Q, entao T_i pode ler, mas não pode escrever Q.
  operação: `lock-S (Q)`
- **Exclusivo:** se uma transação T_i obtem um bloqueio exclusivo sobre o item Q, então T_i pode ler e escrever Q.
  operação: `lock-X (Q)`

Toda transação deve solicitar o bloqueio apropriado do item Q ANTES de realizar qualquer leitura ou gravação.

- toda transação deve desbloquear um item de dado bloqueado por ela antes do final da transação.
  operação: `unlock (Q)`

Compatibilidade de Bloqueios

```
    T1                  T2
-----------         ------------
 lock-X(B)
 read(B)
 B:=B+50
 write(B)
 unlock(B)
                     lock-S(A)
                     read(A)
                     unlock(A)
                     lock-S(B)
                     read(B)
                     unlock(B)
                     print(A+B)
 lock-X(A)
 read(A)
 A:=A-50
 write(A)
 unlock(A)
```

	lock-S	lock-X
lock-S	sim	não
lock-X	não	não

Somente os bloqueios não evitam problemas

Solução: operações unlock ao final da transação

Problema que pode acontecer: Deadlock (impasse)

```
    T1                  T2
----------          -----------
 lock-X(B)
 read(B)
 B:=B-50
 write(B)
                     lock-S(A)
                     read(A)
                     lock-S(B)
 lock-X(A)
```

Solução: desfazer uma das transações

Problema que pode acontecer: Starvation (inanição)

```
   T1          T2          T3          ....        Tn
-------    ---------   ---------              -----------
lock-S(a)
read(a)
           lock-X(a)
                       lock-S(a)
unlock(a)
                       read(a)
                                                lock-S(a)
                       unlock(a)
```

A transação T2 pode esperar indefinidamente se o gerenciador de bloqueios conceder bloqueios compartilhados a outras transações.

Solução: o bloqueio em Q é concedido a Ti se não houver:

- nenhuma transação com bloqueio em Q que seja conflitante
- nenhuma outra transação esperando pelo bloqueio em Q e que tenha feito sua solicitação antes de Ti

Protocolos de Bloqueio

É um conjunto de regras que devem ser seguidas (e impostas pelo SGBD) para garantir que o escalonamento seja equivalente a algum escalonamento serial. Determinam quando uma transação pode ou não bloquear ou desbloquear um item de dado.

Consequência: restringe o número de escalonamentos possiveis. O conjunto de escalonamentos permitidos é um subconjunto de todos os escalonamentos serializáveis.

Protocolo de Bloqueio de Duas Fases (2-phase locking - 2PL)

Exige que a trasação faça as solicitações de bloqueio e desbloqueio em duas fases:

- **fase de crescimento:** pode obter bloqueios, mas não pode liberar nenhum
- **fase de encolhimento:** pode liberar bloqueios, mas não pode obtem nenhum

2PL: Exemplo

```
    T1                  T2
-----------         ------------
 lock-X(B)
 read(B)
 B:=B-50
 write(B)
 lock-X(A)                          <---- ponto de bloqueio de T1
                     lock-S(A)
 read(A)
 A:=A-50
 write(A)
 unlock(B)
 unlock(A)
                     read(A)
                     lock-S(B)      <----- ponto de bloqueio de T2
                     read(B)
                     unlock(A)
                     unlock(B)
                     print(A+B)
```

2PL: Serialização

- **Ponto de bloqueio** de uma transação: o momento que a transação obtém seu último bloqueio
- 2PL: o escalonamento é serializável e equivalente a um escalonamento serial na ordem dos pontos de bloqueio das transações
- 2PL não garante a ausência de deadlock
- 2PL pode causar rollback (cancelamento) em cascata

2PL: Exemplo de Cancelamento em Cascata

```
  T5          T6          T7
---------- ----------  -----------
lock-X(a)
read(a)
lock-S(b)
read(b)
write(a)
unlock(a)
            lock-X(a)
            read(a)
            write(a)
            unlock(a)
                        lock-S(a)
                        read(a)
abort
```

Solução: 2PL restrito
os bloqueios exclusivos são mantidos até que a transacao seja efetivada committed (efetivada)

Outra solução: 2PL rigoroso
todos os bloqueios são mantidos até que a transação seja efetivada

Deadlock

Duas estratégias:

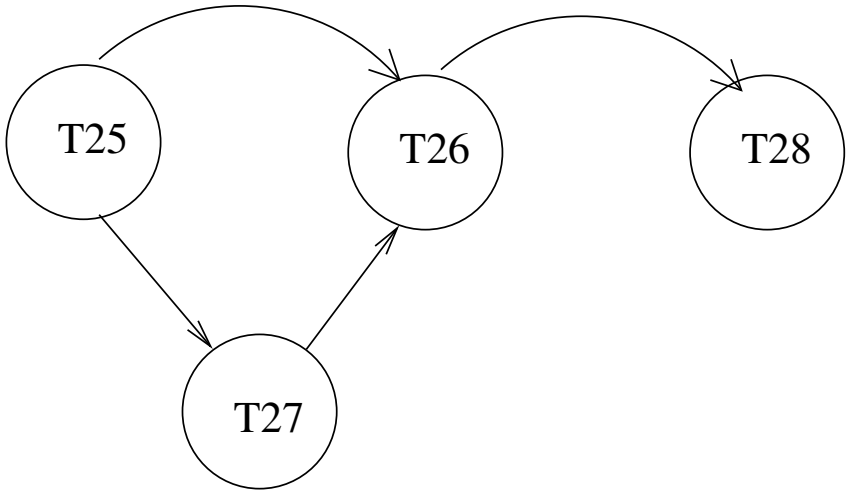
- detecção e recuperação de deadlock
- prevenção de deadlock
  melhor quando a propabilidade de deadlock é alta

Detecção de Deadlock

Grafo de esperar-por:

- cria-se um vértice para cada transação ativa
- cria-se uma aresta $T_i \rightarrow T_j$ se T_i está esperando que T_j libere um item de dado que T_i precisa
- a aresta é removida quando o item é liberado e T_i não precisa mais esperar
- há deadlock quando houver um ciclo no grafo

Detecção de Deadlock - Exemplo



- o sistema nao está em deadlock
- se inserir a aresta $T_{28} \rightarrow T_{27}$ o grafo passa a ter um ciclo envolvendo as transações T_{26}, T_{27} e T_{28}
- o número de vezes que a função de detecção de deadlock é chamado depende de:
  - frequencia de ocorrência de deadlocks
  - quantas transações são afetadas por um deadlock

Recuperação de Deadlock

Ações:

1. Selecionar uma vítima (transação) para ser desfeita:
   Escolher a transação que representa menor "custo", baseado em:
   - tempo de processamento já passado e ainda necessário para terminar a transação
   - quantidade de dados já utilizados e que ainda serão usados
   - quantas transações serão envolvidas no rollback
2. Rollback:
   é necessário determinar até que ponto a transação tem que ser revertida

É preciso cuidar com:

- Inanição:
  garantir que uma transação seja escolhida como vítima somente um número finito de vezes.

Prevenção de Deadlock

Abordagens:

- ordenação das solicitações de bloqueios
  Ex: protocolo de árvore
  Desvantagem: dificuldade de prever quais os itens a serem bloqueados
- obtenção de todos os bloqueios juntos
  Desvantagem: manutenção do bloqueio mesmo que o item de dado seja utilizado muito pouco
- timeout
  - aplicado quando as transações são curtas
  - dificuldade de determinar quanto tempo uma transação deve esperar
- preempção e rollback de transações
  - esperar-morrer
  - ferir-esperar

Prevenção de Deadlock: Esperar-morrer

- cada transação tem um timestamp

```
se Ti solicita um item mantido por Tj
   se TS(Ti) < TS(Tj)
      Ti espera
   senão
      Ti é desfeita (mas continua com o mesmo TS)
```

Exemplo:

```
    T1                  T2
-----------         ------------
 lock-X(B)
 read(B)
 B:=B-50
 write(B)
                     lock-S(A)
                     read(A)
                     lock-S(B)
 lock-X(A)
```

Prevenção de Deadlock: Ferir-esperar

- cada transação tem um timestamp

```
se Ti solicita um item mantido por Tj
   se TS(Ti) < TS(Tj)
      Tj é desfeita (mas continua com o mesmo TS)
   senão
      Ti espera
```

- não há problema de inanição porque quando uma transação é desfeita ela continua com o mesmo timestamp
- Problema: podem ocorrer rollbacks desnecessários

Comparação entre Abordagens de Preempção e Rollback

	tempo de espera	**número de reversões**
esperar-morrer	quanto mais antiga a transação maior a possibilidade de esperar	uma transação pode ser desfeita várias vezes se o item nao for liberado rapidamente
ferir-esperar	transação mais antiga nunca espera	para cada conflito só ocorre uma reversão

Promoção e Rebaixamento de Bloqueio

```
T8: read(a1)
    read(a2)
    ...
    read(an)
    write(a1)

T9: read(a1)
    read(a2)
    print a1+a2
```

```
   T8          T9
---------- ---------
lock-S(a1)
            lock-S(a1)
lock-S(a2)
            lock-S(a2)
lock-S(a3)
....
            unlock(a1)
            unlock(a2)
lock-S(an)
upgrade(a1)
```

Promoção e Rebaixamento de Bloqueio

- lock-S $\Longrightarrow$ lock-X e' um upgrade:
  só pode ser feito na fase de crescimento
- lock-X $\Longrightarrow$ lock-S e' um downgrade:
  só pode ser feito na fase de encolhimento

Geração das instrucoes de lock/unlock

- read(Q): gera lock-S(Q)
  seguido de write(Q) $\longrightarrow$ upgrade(Q)
- write(Q): gera lock-X(Q)
- commit: gera unlock de todos os itens de dados

Controle de Concorrência com Timestamp

- um timestamp por transação: $TS(T_i)$
  - $TS(T_i)$ é o tempo do início da transação T_i:
    relógio do sistema (clock) ou contador lógico
  - os timestamps determinam a ordem de serializacao:
    se $TS(T_i) < TS(T_j)$ então o sistema produz um escalonamento equivalente ao escalonamento serial em que T_i aparece antes de T_j
- 2 timestamps por item Q utilizado em alguma transação:
  - W-timestamp(Q):
    maior timestamp da transação que executou `write(Q)` com sucesso
  - R-timestamp(Q):
    maior timestamp da transação que executou `read(Q)` com sucesso

Protocolo de Controle de Concorrência por Timestamp

- se T_i executa um `read(Q)`:
  - se $TS(T_i)$ <W-timestamp(Q) então desfazer T_i
  - se $TS(T_i)$ >=W-timestamp(Q) então
    - executa read(Q)
    - R-timestamp(Q) $\leftarrow$ maior valor entre $TS(T_i)$ e R-timestamp(Q)
- se T_i executa um `write(Q)`:
  - se $TS(T_i)$ <R-timestamp(Q) então desfazer T_i
  - se $TS(T_i)$ <W-timestamp(Q) então desfazer T_i
  - se $TS(T_i)$ $\geq$R-timestamp(Q) e $TS(T_i)$ $\geq$W-timestamp(Q) então
    - executa write(Q)
    - W-timestamp(Q) $\leftarrow$ $TS(T_i)$
- quando T_i é desfeita, recebe um **novo timestamp** e é reiniciada.

Exemplo: Protocolo de Controle de Concorrência por Timestamp

```
     T1                T2
-------------- ------------
read(B)
                read(B)
                B:= B-50
                write(B)
read(A)
                read(A)
print(A+B)
                A:= A+50
                write(A)
                print(A+B)
```

Regra de Escrita de Thomas

- altera o protocolo de timestamp permitindo maior nível de concorrência
- usa o conceito de serialização por visão

```
   T16                T17
------------   ------------
 read(Q)
                  write(Q)
 write(Q)

se Ti emite write(Q):
  . se TS(Ti) < R-timestamp(Q) então
      desfaz Ti
  . se TS(Ti) < W-timestamp(Q) então
      ignora write(Q)
  . senão
      executa write(Q)
      W-timestamp(Q):= TS(Ti)
```