Bancos de Dados I

Controle de Concorrência com Locks

Transações em SQL

Prof. Altigran Soares da Silva

Controle de Concorrência com Locks

- O uso de locks (bloqueios) e uma das principais técnicas para controlar a execução concorrente de transações
- Um **lock** é uma variável associada a um item de dados em um BD
- Geralmente, existem um lock para cada item de dados.
- Descreve o estado do item com respeito às possíveis operações que podem ser realizadas.
- Se utilizado corretamente, ajuda a sincronizar o acesso das transações concorrentes.

V2018.1

Tipos de Locks

- Lock Binário:
  - Apenas dois estados: lock e unlock
  - Muito simples e muito restritivo
  - Não é usado na prática
- Lock Multi-modo (shared/exclusive)
  - Mais geral
  - Usado na prática em SGBDs
  - Três estados: read-lock, write-lock, unlock

Locks Binários

- Garantem exclusão mútua em um item de dados
  - Em um dado instante, somente uma transação pode manter o lock de um item X
- Se o lock(X)=0, X pode ser acessado por uma transação
- Se o lock(x)=1, X não pode ser acessado por outra transação

Operação lock_item(X)

- Transação T executa lock_item (X):
  - Se lock(X) = 1
    - A transação T é forçada a esperar em uma fila Q(X)
  - Se lock(X) = 0
    - lock(x) ← 1
    - A transação T pode acessar X

Operação unlock_item(X)

- Transação T executa unlock_item (X):
  - lock(X) ← 0
  - "Acorda" a próxima transação esperando na fila Q(X), se houver.

lock_item(X) e unlock_item(X) no SO

- lock_item e unlock_item devem ser implementadas como seções críticas no Sistema Operacional
- Isso significa que não pode haver premptação durante sua execução
- Cada item X deverá ter uma fila Q(X) para organizar a espera das transações

Tabela de Locks

- Para cada item dados:
  - <item_id, valor_lock,transação,fila>
- Gerente de Locks:
  - Subsistema do SGBD que mantêm a tabela de locks e garante o controle de acesso

Regras para Locks Binários

- T deve executar
  - lock_item(X) antes de read(X) ou write(X)
  - unlock_item(X) depois de read(X) and write(X)
- T não deve executar
  - lock_item(X) se já tem o lock de X
  - unlock_item(X) a menos que já tenha o lock de X

Locks Multi-modo

- lock(X) ∈ {read-lock,write-lock, unlock}
- read-lock ou lock compartilhado
  - Várias transações podem ler o item
  - Nenhuma transação pode escrever o item
- write-locked ou lock exclusivo
  - Somente uma transação pode acessar o item
- Tabela de Locks
  - <item_id, valor_lock, leitores,transacs,fila>.

Regras para Locks Multi-modo

- T deve executar
  - read_lock(X) ou write_lock(X) antes de read(X)
  - write_lock(X) de write_item(X)
  - unlock(X) depois de read(X) e write(X)
- T não deve executar
  - read_lock(X) se ela já tem o bloqueio do item
  - write_lock(X) se ela já tem o bloqueio do item
  - unlock(X) a menos que ela já tenha o bloquio do item

Conversão de Locks Multi-modo

- Upgrade:
  - Se T á única transação que possui o bloueio de leitura para X, T pode executar write_lock(X) operation
- Downgrade
  - Uma transação T pode executar incialmente um write_lock(X) e mais tarde executar um read_lock(X)

Excessões à regra geral

Locks e Serializabilidade: Exemplo 1

- O simples uso de locks não garante serializabilidade
- Exemplo 1:
  - atualização perdida

T1: (joe)	T2: (fred)	X	Y
write_lock(X)			
read_item(X);		4	
X:= X - N;		2	
unlock(X)			
	write_lock(X)		
	read_item(X);	4	
	X:= X + M;	7	
	unlock(X)		
write_lock(X)			
write_item(X);		2	
unlock(X)			
write_lock(Y)			
read_item(Y);			8
	write_lock(X)		
	write_item(X);	7	
	unlock(X)		
Y:= Y + N;			10
write_item(Y);			10
unlock(Y)			

Locks e Serializabilidade: Exemplo 2

- Inicialmente
  - X=20 e Y=30
- Escalanomentos Seriais
  - T1 seguido de T2: X=50 e Y=80
  - T2 seguido de T1: X=70 e Y=50

T1	T2
read_lock(Y);	read_lock(X);
read_item(Y);	read_item(X);
unlock(Y);	unlock(X);
write_lock(X);	write_lock(Y);
read_item(X);	read_item(Y);
X:=X+Y;	Y:=X+Y;
write_item(X);	write_item(Y);
unlock(X);	unlock(Y);

Locks e Serializabilidade: Exemplo 2

- Inicialmente:
  - X=20 e Y=30
- Ao final:
  - X=50 e Y=50

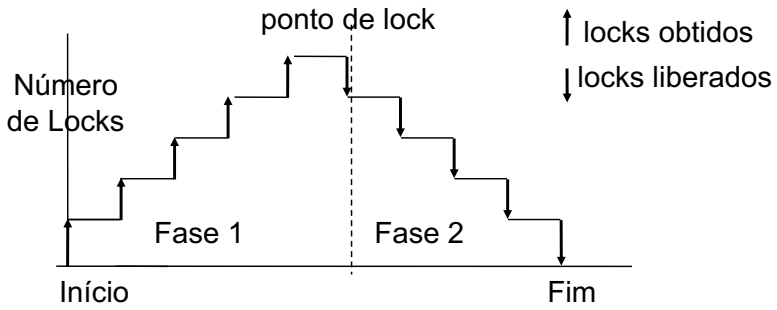
T1	T2
read_lock(Y);	
read_item(Y);	
unlock(Y);	
	read_lock(X);
	read_item(X);
	unlock(X);
	write_lock(Y);
	read_item(Y);
	Y:=X+Y;
	write_item(Y);
	unlock(Y);
write_lock(X);	
read_item(X);	
X:=X+Y;	
write_item(X);	
unlock(X);	

Two-Phase Lock (2PL)

- Two-Phase Lock ou Bloqueio em 2 fases
- Protocolo para controle de concorrência baseada em bloqueios (locks)
- Um transação segue o 2LP se todos os locks (read_lock, write_lock) precedem o primeiro unlock

Two-Phase Lock (2PL)

- Fase 1: Expansão, locks são obtidos mas nenhum é liberado
- Fase 2: Contraçao, locks são liberados mas nenhum novo lock é obtido



Two-Phase Lock (2PL)

- As duas transações abaixo seguem o 2PL

T1	T2
read_lock(Y);	read_lock(X);
read_item(Y);	read_item(X);
write_lock(X);	write_lock(Y);
unlock(Y);	unlock(X);
read_item(X);	read_item(Y);
X:=X+Y;	Y:=X+Y;
write_item(X);	write_item(Y);
unlock(X);	unlock(Y);

2PL e Serializabilidade

- Se todas as transações que participam de um escalonamento S seguem o 2PL, então S é garantidamente serializável
- Não é necessário aplicar o teste de serializabilidade sobre S
- Se o mecanismo de controle de concorrência garante o 2PL então a serializabilidade é efetivamente garantida.

2PL e Concorrência

- Uma transação T pode ter que “segurar” um item X somente porque precisa bloquear um outro item Y antes de liberar X.
- Uma outra transação pode ser obrigada a esperar para acessar X, mesmo que não precise mais usá-lo.
- T pode ter que bloquear Y muito antes do necessário pra poder liberar um item X
- Se Y é bloqueado antes do necessário, pode ser que outra transação tenha que esperar por Y, mesmo que este item não tenha sido usado ainda.

Variação 1: 2PL Conservativo

- A transação deve bloquear todos os itens antes que a transação inicie
  - Pre-declaração de “read-set” e “write-set”
- Se algum dos itens não pode ser bloqueado, nenhum será. A transação espera por todos os itens
- Vantagem: garante ausência de deadlocks
- Desvantagem: difícil de implementar na prática

Variação 2: 2PL Estrito

- Nenhum write-lock é liberado até que a transação execute um *commit* ou *abort*
- Nenhuma outra transação pode acessar um item até que T comprometa ou aborte.
- Garante escalonamentos estritos, sem necessade de rollback em cascata
- Variação mais popular do 2PL
- Não é livre de deadlocks

Variação 3: 2PL Rigoroso

- Nenhum write-lock ou read-lock é liberado até que a transação execute um *commit* ou *abort*
- Similar ao 2PL estrito, exceto que é mais restritivo
- Mais simples de implementar, já que todos os locks são mantidos até o commit

Variações 2PL e Concorrência

- As variações 2PL reduzem ainda mais o nível de concorrência do sistema
- Se usarmos conservativo e rigoroso, teremos como resultado escalonamentos seriais!
- O 2PL garante que os escalonamentos gerados são serializavéis.
- No entanto, podem haver escalonamentos serializáveis que não usam 2PL

Deadlock

- Ocorre quando cada transação em conjunto de duas ou mais transações está esperando por um item X, mas X está bloqueado por um outra transação T' no conjunto
- Assim, cada transação no conjunto está em uma filam esperando que um outra transação libere um item de dado.

Deadlock - Exemplo

T1	T2
read_lock(Y);	read_lock(X);
read_item(Y);	read_item(X);
write_lock(X);	write_lock(Y);
unlock(Y);	unlock(X);
read_item(X);	read_item(Y);
X:=X+Y;	Y:=X+Y;
write_item(X);	write_item(Y);
unlock(X);	unlock(Y);

Escalonamento que causa Deadlock

T1	T2
read_lock(Y);	
read_item(Y);	
	read_lock(X);
	read_item(X);
write_lock(X);	

Técnicas para controle de Deadlock

- Eliminação da possibilidade de deadlocks através do 2PL conservativo
- Prevenção de Deadlock
  - Um transação que solicita um novo lock é abortada se existe a possibilidade de deadlock
- Detecção de Deadlock
  - O SGBD periódicamente verifica a ocorrência de deadlocks. Se ocorrer, uma das transações (“vítima”) é abortada e as outras continuarão

Prevenção de Deadlocks com Timestamps

- Cada transação T recebe um *timestamp* TS(T) único
  - Se T1 inicia antes de T2, então TS(T1) < TS(T2)
- wait-die:
  - se TS(T1) < TS (T2), T1 pode esperar, senão T1 é abortada e reiniciada com o mesmo timestamp
- wound-wait:
  - se TS(T1) < TS(T2), abortar T2 e restartar depois como mesmo timestamp, senão T1 pode esperar
- A transação mais nova é sempre abortada

Prevenção de Deadlocks sem Timestamps

- Sem espera (no waiting) :
  - Se uma transação não pode obter um lock, ela aborta depois de um tempo pre-definido
- Espera cautelosa (cautions wait)
  - Considere que Ti tentar bloquear um item X que já está bloqueado por Tj
  - Se Tj não está bloquado por algum outro item,
    - então Ti pode esperar
    - senão Ti é abortada

Transações em SQL

Inaniação (starvation)

- Uma transação sofre de inaniação se ela é impedida de progredir por outras transações durante um longo periodo de tempo
- Possíveis causas:
  - Esquemas de bloqueio/desbloqueio injustos que priorizam determinadas transações sobre outras
  - Esquemas de prevenção de deadlock que vitimam a mesma transação repetidas vezes
- Prevenção: wait-die e wound-wait

Transações em SQL

- Em interfaces SQL genéricas
  - Cada comando é implicitamente considerado como uma transação
- SQL embutido
  - Comandos em linguagens de programação
  - `BEGIN_TRANSACTION` é implícito
  - Fim de transações é explícitamente declarado
    - Comandos `COMMIT` ou `ROLLBACK`

Transação em SQL – Exemplo

- Comandos SQL embutidos em C
- Declaração de variáveis compartilhadas

```
EXEC SQL BEGIN DECLARE SECTION;
  int conta1, conta2;
  int saldo1;
  int valor;
EXEC SQL END DECLARE SECTION;
```

Transação em SQL – Exemplo

```
void transfer() {
EXEC SQL SELECT saldo INTO :saldo1
       FROM   Contas
       WHERE  conta_num = :conta1;
if (:saldo1 >= :valor){
  EXEC SQL UPDATE Contas
           SET    saldo = saldo + :valor;
           WHERE  conta_num = :conta2;
  EXEC SQL UPDATE Contas
        SET    saldo = saldo - :valor;
        WHERE  conta_num = :conta1;
  EXEC SQL COMMIT;}
  else { EXEC SQL ROLLBACK;}
}
```

Transações de leitura X escrita

- Transações leitura e escrita são default
  - `SET TRANSACTION READ WRITE;`
- Transações de leitura somente
  - `SET TRANSACTION READ ONLY`
  - O SGBD pode tirar vantagem de que não há escritas e aumentar o grau de concorrência

Níveis de Isolamento

- Definem comportamento da transação com respeito ao controle de concorrência
- `SET TRANSACTION ISOLATION LEVEL` <opções>
  - <opções> :
    - `READ UNCOMMITTED`
    - `READ COMMITTED`
    - `REPEATABLE READ`
    - `SERIALIZABLE`
- Default é `SERIALIZABLE`
  - Produz escalonamentos serializáveis

READ UNCOMMITTED

- Permite leituras “sujas”
- Dados “sujos” são dados escritos por transações que ainda não tenha comitado
- Leituras “sujas” são leituras de dados sujos

REPEATABLE READ

- Se uma tupla é recuperada em uma consulta na transação, então ela será garantidamente recuperada em consultas subsequentes da transação.
- No entanto, novas tuplas inseridas por outra transação entre as duas consultas podem aparecer
- Estas são chamadas “tuplas fantamas” (phantom tuples)

Níveis de Isolamento – Resumo

	Tipos de Violação		
Nível de Isolamento	Leitura suja	Leitura dupla	Tupla Fantasma
READ UNCOMMITTED	sim	sim	sim
READ COMMITTED	não	sim	sim
REPEATABLE READ	não	não	sim
SERIALIZABLE	não	não	não