CONTROLE DE CONCORRÊNCIA - PROTOCOLO TWO-PHASE LOCKING

Sérgio Mergen

Versão modificada de Database System Concepts, 5th Ed.

Tópicos

- Protocolos de Controle de Concorrência
  - Schedules concorrentes
  - Schedules recuperáveis
  - Schedules livres de rollback em cascata
- Protocolos baseados em lock
  - Mecanismo de Lock
  - Protocolo Two-Phase Locking
  - Protocolo Rigorous Two-Phase Locking

Protocolos de Controle de Concorrência

- Quando as aplicações enviam comandos SQL para o SGBD, esses comandos devem obter acesso aos dados
  - Esses comandos podem ser vistos como instruções de READ e WRITE

- Ex. considere que a conta bancária 890-01 seja identificada pelo apelido A
  - o comando *SELECT * FROM conta WHERE numConta = '890-01'*
    - seria traduzido para *READ(A)*

Protocolos de Controle de Concorrência

- Quando muitas instruções de READ e WRITE desejam acessar os itens de dados, como controlar o acesso a esses itens?

- O mecanismo usado para isso é chamado de protocolo de controle de concorrência
  - A ordem com que o acesso é concedido vem a se tornar o schedule

Protocolos de Controle de Concorrência

- O objetivo de um protocolo de controle de concorrência é gerar schedules
  - Concorrentes (e consistentes)
  - Recuperáveis
  - Sem rollbacks em cascata

Protocolos de Controle de Concorrência

- Vantagens em ter schedules concorrentes
  - **Maior utilização do disco e do processador**, levando a uma maior vazão (*throughput)* das transações
    - Ex. Uma transação pode usar o CPU enquanto a outra está lendo ou gravando do disco
  - **Redução do tempo médio de resposta**: transações curtas não esperam mais tanto atrás de transações longas.

Protocolos de Controle de Concorrência

- O objetivo de um protocolo de controle de concorrência é gerar schedules
  - Concorrentes (e consistentes)
  - Recuperáveis
  - Sem rollbacks em cascata

Protocolos de Controle de Concorrência

- **Schedule recuperável** — se uma transação ler um item de dados previamente escrito por outra transação
  - o **commit** da transação que fez a escrita deve aparecer antes do **commit** daquela que fez a leitura

- O schedule ao lado não é recuperável

T8	T9
read (A) **write(A)** **read (B)** ***Abort***	 **read(A)** ***Commit***

Protocolos de Controle de Concorrência

- **Schedule recuperável** — se uma transação ler um item de dados previamente escrito por outra transação
  - o **commit** da transação que fez a escrita deve aparecer antes do **commit** daquela que fez a leitura

- T_8 aborta e T_9 lê (e possivelmente mostra ao usuário) um banco inconsistente.

T8	T9
read (A)	
write(A)	
	read(A)
	Commit
read (B)	
Abort	

É imprescindível que o SGBD gere schedules recuperáveis.

Protocolos de Controle de Concorrência

- O objetivo de um protocolo de controle de concorrência é gerar schedules
  - Concorrentes (e consistentes)
  - Recuperáveis
  - Sem rollbacks em cascata

Protocolos de Controle de Concorrência

- Rollback é a ação que é tomada quando uma transação aborta
  - Significa que uma transação precisa ser desfeita (revertida)
  - Isso pode vir a ser necessário para assegurar o requisito de atomicidade
    - Ou tudo é executado ou nada é

- Rollbacks em cascata
  - Ocorre quando o **abort** em uma transação leva a um **abort** de outra transação
  - que leva ao **abort** de outra transação
  - e assim por diante
  - gerando um efeito de rollbacks em cascata

Protocolos de Controle de Concorrência

- No schedule ao lado nenhuma das transações "comitou" ainda
  - ou seja, o schedule ainda é recuperável

T10	T11	T12
read (A)		
read (B)		
	read(B)	
write *(A)*		
	read(A)	
	write(C)	
		read(C)

Protocolos de Controle de Concorrência

- Se T_{10} falhar, T_{11} e T_{12} também devem ser revertidas (roll back).
  - Para garantir a consistência do banco

T10	T11	T12
read (A)		
read (B)		
	read(B)	
write *(A)*		
	read(A)	
	write(C)	
		read(C)
Abort		
	Abort (cascata)	
		Abort (cascata)

- Pode levar a reversão de uma quantidade grande de trabalho
- E a reversão tem um custo relativamente alto

Protocolos de Controle de Concorrência

- **Schedules sem cascata**
  - Se uma transação ler um item de dados previamente escrito por outra transação
    - A transação que fez a escrita deve comitar antes que a outra transação tenha feito a leitura

- Schedules sem cascata previnem rollbacks em cascata
  - É desejável restringir os schedules para os que são sem cascata

Protocolos de Controle de Concorrência

- O schedule ao lado é livre de cascata
- E também é recuperável
- Todo schedule sem cascata é também recuperável

T10	T11	T12
read (A)		
read (B)		
	read(B)	
write *(A)*		
Commit		
	read(A)	
	write(C)	
	Commit	
		read(C)
		.
		.

Tópicos

- Protocolos de Controle de Concorrência
  - Schedules concorrentes
  - Schedules recuperáveis
  - Schedules livres de rollback em cascata
- Protocolos baseados em lock
  - Mecanismo de Lock
  - Protocolo Two-Phase Locking
  - Protocolo Rigorous Two-Phase Locking

Protocolos baseados em lock

- Um **protocolo baseado em lock** é um conjunto de regras seguidas por todas as transações ao pedir e liberar locks.

- Um lock é um mecanismo que controla o acesso concorrente a um item de dados

Tópicos

- Protocolos de Controle de Concorrência
  - Schedules concorrentes
  - Schedules recuperáveis
  - Schedules livres de rollback em cascata
- Protocolos baseados em lock
  - Mecanismo de Lock
  - Protocolo Two-Phase Locking
  - Protocolo Rigorous Two-Phase Locking

Protocolos baseados em lock

- Itens de dados podem sofrer lock em dois modos:

- 1. *exclusivo (X)*
  - O item de dados pode ser tanto lido quanto escrito
  - O lock é obtido usando a instrução **lock-X**.

- 2. *compartilhado (S)*.
  - O item de dados pode ser apenas lido
  - O lock é obtido usando a instrução **lock-S**.

Mecanismo de Lock

- Os pedidos de lock são feitos para o gerenciador do controle de concorrência.
  - A transação pode prosseguir apenas depois que o pedido for aceito

Mecanismo de Lock

- Matriz de compatibilidade de Lock

Lock concedido \ Lock solicitado	S	X
S	Sim	Não
X	Não	Não

- Uma transação pode ter um pedido de lock aceito em um item se
  - O pedido de lock for **compatível** com os locks já obtidos sobre o item por outras transações

Mecanismo de Lock

- Matriz de compatibilidade de Lock

Lock concedido \ Lock solicitado	S	X
S	Sim	Não
X	Não	Não

- Várias transações podem ter locks **compartilhados** sobre um item
  - Mas se uma transação possui um lock exclusivo sobre um item
    - Nenhuma outra transação pode obter qualquer tipo de lock sobre o mesmo item

Mecanismo de Lock

- Matriz de compatibilidade de Lock

Lock concedido \ Lock solicitado	S	X
S	Sim	Não
X	Não	Não

- Se um lock não puder ser dado
  - A transação que fez o pedido deve **esperar**
    - Até que todos os locks incompatíveis das outras transações sejam liberados
    - Para então o pedido de lock ser aceito.

Mecanismo de Lock

- Ao lado, exemplo de transações realizando locks

- Isso por si só não é suficiente para garantir schedules serializáveis.
  - A gravação de A em T1 leva à soma impressa errada em T2.

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

Tópicos

- Protocolos de Controle de Concorrência
  - Schedules concorrentes
  - Schedules recuperáveis
  - Schedules livres de rollback em cascata
- Protocolos baseados em lock
  - Mecanismo de Lock
  - Protocolo Two-Phase Locking
  - Protocolo Rigorous Two-Phase Locking

Protocolo Two-Phase Locking (2PL)

- Fase 1: Fase de crescimento
  - Transações podem obter locks
  - Transações não podem liberar locks
- Fase 2: Fase de encolhimento
  - Transações podem liberar locks
  - Transações não pode obter locks

Mecanismo de Lock

- O schedule ao lado não obedece ao *two-phase locking*

- Na transação T1, um **lock** aparece depois de um **unlock**

T1	T2
lock-X*(A);*	
read (A);	
A := A – 50;	
write *(A);*	
unlock*(A);*	
	lock-S*(A);*
	lock-S*(B);*
	read(A);
	read(B);
	print(A + B);
	unlock*(A);*
	unlock*(B);*
	commit;
lock-X*(B);*	
read *(B);*	
B := B + 50;	
write *(B);*	
unlock*(B);*	
commit;	

Protocolo Two-Phase Locking (2PL)

- O schedule ao lado obedece ao *two-phase locking*
- Em cada transação, todos **locks** vêm antes de todos **unlocks**

T1	T2
lock-X*(A)*;	
read (A);	
write *(A)*;	
lock-S*(B)*;	
unlock*(A)*;	
	lock-S*(C)*;
	read (C);
read *(B)*;	
	lock-S*(A)*;
unlock*(B)*;	
	read *(A)*;
Commit	
	unlock*(C)*;
	unlock*(A)*;
	Commit

Protocolo Two-Phase Locking (2PL)

- O protocolo two-phase locking puro tem inconvenientes
  - Não previne rollbacks em cascata
  - Gera schedules não recuperáveis
  - Depende do programador para solicitar bloqueio e desbloqueio de itens

Protocolo Two-Phase Locking (2PL)

- **Exemplo de rollback em cascata**

- No schedule ao lado, ao abortar T1
  - T2 também precisaria abortar

T1	T2
lock-X*(A)*; read (A); write *(A)*; lock-S*(B)*; unlock*(A)*; read *(B)*; unlock(B); abort	 lock-S*(A)*; read *(A)*; abort (cascata)

Protocolo Two-Phase Locking (2PL)

- **Exemplo de schedule não recuperável**

- No schedule ao lado, ao abortar T1
  - T2 precisaria ser revertida
  - Mas ela já foi confirmada

T1	T2

lock-X*(A)*;
read (A);
write *(A)*;
lock-S*(B)*;
unlock*(A)*;
 	
read *(B)*;
 	
unlock(B);
 	
abort
	
	lock-S*(A)*;
	
	read *(A)*;
	
	print(A);
	unlock(A);
	commit

Tópicos

- Protocolos de Controle de Concorrência
  - Schedules concorrentes
  - Schedules recuperáveis
  - Schedules livres de rollback em cascata
- Protocolos baseados em lock
  - Mecanismo de Lock
  - Protocolo Two-Phase Locking
  - Protocolo Rigorous Two-Phase Locking

Protocolo Rigorous Two-Phase Locking

- Nessa versão, uma transação deve segurar todos os locks até a instrução final de commit/abort

- Ao contrário da versão pura, o protocolo rigoroso
  - Previne rollbacks em cascata
  - Gera somente schedules recuperáveis
  - Realiza bloqueio e desbloqueio de itens automaticamente

Protocolo Rigorous Two-Phase Locking

- Os pedidos de lock são feitos **automaticamente** quando a transação tenta usar o item pela primeira vez

T1	T2
read (A) / lock-S*(A)* write *(A)* / lock-X*(A)* read *(B)* / lock-S*(B)* commit / unlock(A,B)	read(C) / lock-S*(C)* read *(A)* / lock-S*(A)*; commit / unlock(A,C)

Protocolo Rigorous Two-Phase Locking

- Os unlocks são realizados **automaticamente** quando a transação é encerrada
  - Por commit
  - Ou abort

T1	T2
read (A) / lock-S*(A)* write *(A)* / lock-X*(A)* read *(B)* / lock-S*(B)* commit / unlock(A,B)	read(C) / lock-S*(C)* read *(A)* / lock-S*(A)*; commit / unlock(A,C)

Encerrando

- O protocolo 2PL e suas variações geram muitos bloqueios

- Na prática, os bancos de dados usam um protocolos menos bloqueantes, baseados em múltiplas versões de um registro (MVCC - Multiversion Concurrency Control).
  - Nesses protocolos
    - Leitura não bloqueia escrita
    - Escrita não bloqueia leitura

- Mas, dependendo da configuração
  - alguns SGBDs podem adotar um controle igual ou muito semelhante ao 2PL

Atividade Individual

- O script disponibilizado possui um esquema relacional composto por
  - Tabelas:
    - Cliente, Evento, Ingresso
  - Um script de geração de ingressos
    - Criar_ingressos
  - Alguns dados

```
insert into evento values (1, 'show 1', now());

insert into cliente values (1, 'cliente 1’);
insert into cliente values (2, 'cliente 2’);

CALL criar_ingressos(1, 10);
```

Atividade Individual

- O objetivo da atividade é criar uma procedure com a seguinte assinatura:

```
CREATE PROCEDURE reservar_ingressos(
   IN p_eventoID INT,    IN p_clienteID INT,
   IN ingresso1 INT,    IN ingresso2 INT)
```

- A procedure deve realizar a reserva de até dois ingressos (ingresso1, ingresso2) para a mesma pessoa (p_clienteID) no mesmo evento (p_eventoID).

Atividade Individual

- Caso não seja possível reservar algum ingresso, por não existir ID ou por reserva pre-existente
  - as duas reservas devem ser canceladas, e uma mensagem de erro exibida
  - Ex.

```
-- cliente 1 reservando os ingressos 1 e 2 para o evento 1
call reservar_ingressos(1, 1, 1, 2);

-- cliente 2 reservando os ingressos 5 e 1 para o evento 1
-- aqui precisa dar problema, pois o ingresso 1 já foi reservado
call reservar_ingressos(1, 2, 5, 1);
```

  - O controle da procedure reservar_ingressos deve ser realizado por meio de uma transação