Processamento de Transações em BD

Banco de Dados II

Bianca Maria Pedrosa

Índice

Transações

“Transação é uma unidade lógica de processamento de um SBD que acessa e, possivelmente, atualiza vários itens de dados”.

[Silberschatz99]

Exemplo

- Transferir $100 da conta A para a conta B

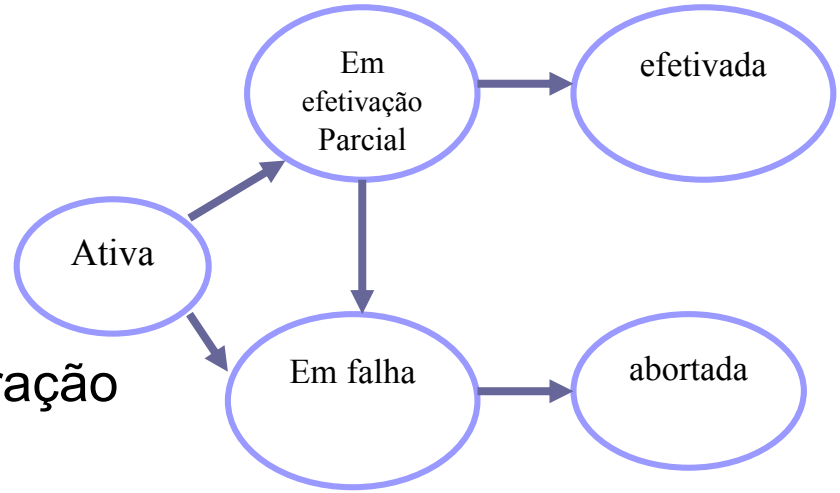
```
read(A);
A:=A-100;
write(A);
read(B);
B:=B+100;
write(B);
```

Propriedades (ACID)

- Atomicidade
  - Todas ou nenhuma das operações são refletidas no BD
- Consistência
  - O BD deve refletir um estado do mundo real
- Isolamento
  - Uma transação não interfere em outra executada concorrente
- Durabilidade
  - Depois de realizada com sucesso, uma transação persiste, mesmo se houver falhas no sistema

Estados

- Ativa
  - em execução
- Em efetivação parcial
  - após execução da última declaração
- Em Falha
  - execução normal não pode se realizar
- Em efetivação
  - após a conclusão com sucesso
- Abortada
  - após transação desfeita, BD = estado incial



Suporte a Transações em MySQL

- **Para ter suporte a transações em MySQL crie Tabelas InnoDB**
  - O InnoDB provê o MySQL com um mecanismo de armazenamento seguro com transações (compatível com ACID) com commit, rollback, e recuperação em caso de falhas. InnoDB faz bloqueio a nível de registro e também fornece uma leitura sem bloqueio em SELECT em um estilo consistente com Oracle. Estes recursos aumentam a performance e a concorrência de multi-usuários.
  - Não há a necessidade de escalonamento de bloqueios em InnoDB, pois o bloqueio a nível de registro no InnoDB cabe em um espaço muito pequeno.
  - Para ler mais sobre os tipos de tabelas do mysql acesse o capítulo 7 do manual de referência do MySQL.

SQL e Transações

- Para iniciar uma transação use:
  - Start transaction | Begin
- Para concluir uma transação use:
  - Commit (efetivação)
  - Rollback(reverter/desfazer)

SQL: Exemplo 1

```
Start transaction;
    Update Conta
        Set saldo = saldo - 100;
        Where idCliente = ‘A’;
     Update Conta
        Set saldo = saldo + 100;
        Where idCliente = ‘B’;
Commit;
```

SQL: Exemplo 2

```
Start transaction;
    Update Conta
        Set saldo = saldo - 100;
        Where idCliente = 'A';
    Update Conta
        Set saldo = saldo + 100;
        Where idCliente = 'B';
Rollback;
```

Obs: todas as operações a partir do begin serão desfeitas

Exemplos de Transações

T_1 transfere 100 da conta A para a conta B

T_1

```
read(A);
A:=A-100;
write(A);
read(B);
B:=B+100;
write(B);
```

T_2 atualiza A e B, bom base no valor antigo de A

T_2

```
read(A);
temp:=A*0,1;
A:=A-temp;
write(A);
read(B);
B:=B+temp;
write(B);
```

Escala de Execução Seqüencial

T_1	T_2
read(A);	
A:=A-100;	
write(A);	
read(B);	
B:=B+100;	
write(B);	
	read(A);
	temp:=A*0,1;
	A:=A-temp;
	write(A);
	read(B);
	B:=B+temp;
	write(B);

Escala 1

T_1	T_2
	read(A);
	temp:=A*0,1;
	A:=A-temp;
	write(A);
	read(B);
	B:=B+temp;
	write(B);
read(A);	
A:=A-100;	
write(A);	
read(B);	
B:=B+100;	
write(B);	

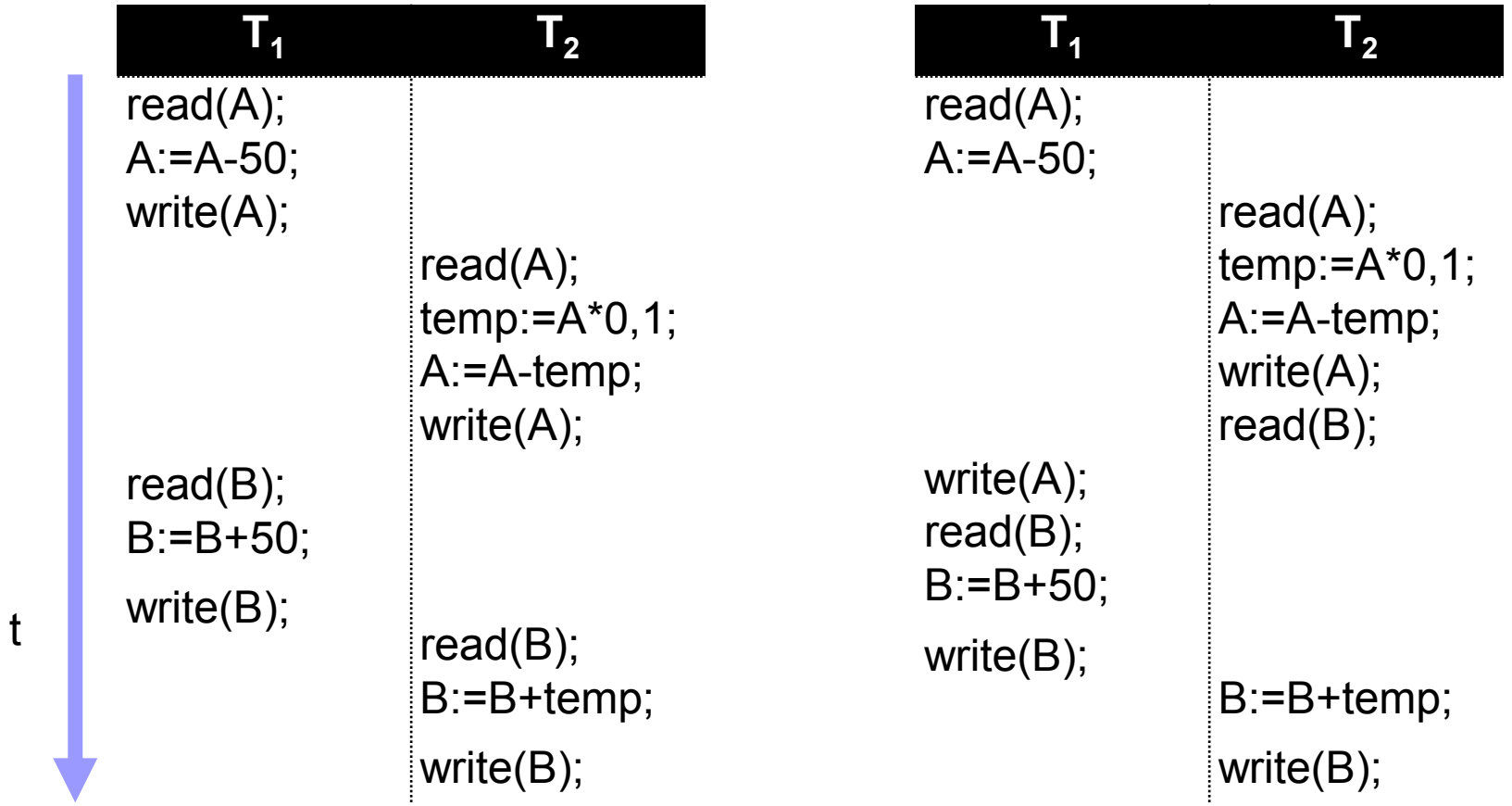
Escala 2

Execuções Concorrentes

Motivações:

- Atividade de I/O e Atividade de CPU são independentes. Logo, o paralelismo entre elas pode ser explorado
- Em um SBD existem transações curtas e longas. O processamento concorrente pode reduzir o tempo médio de resposta do sistema.

Execuções Concorrentes



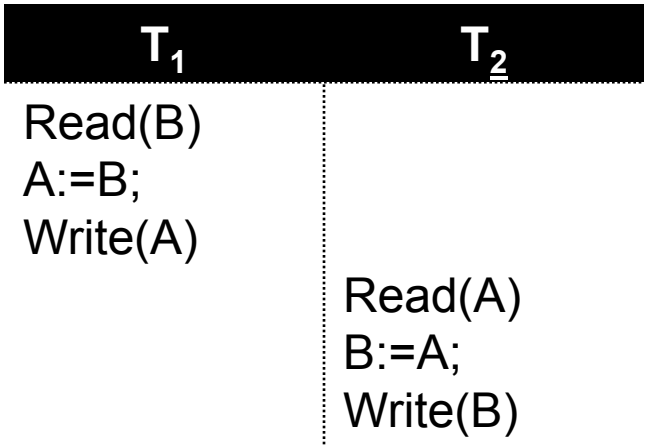
Processamento entrelaçado

Concorrência e Consistência do BD

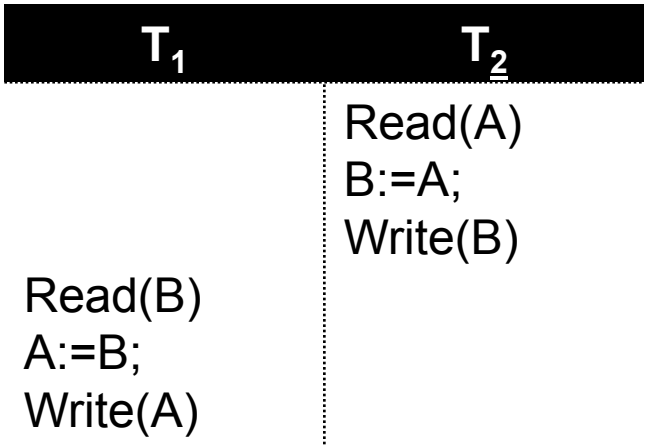
- Qualquer escala concorrente deve levar o BD a um estado consistente
- Um BD está consistente se seu estado reflete o resultado de qualquer escala sequencial de transações

Teste de Consistência

- Sejam duas transações T1:A:=B; T2:B:=A;
- Suponha que A=4 e B=10 antes da execução das transações.
- Qualquer execução concorrente das duas transações deve levar A e B a valores iguais



Escala 1: A:=10;b:=10;



Escala 2: A:=4; b:=4;

Teste de Consistência

- Uma escala concorrente *por si só* não garante isolamento
- Lembre-se que A=4 e B=10 antes da execução das transações.
- Nenhuma execução concorrente das duas transações levaria A e B a valores trocados

T_1	T_2
Read(B)	
A:=B;	
	Read(A)
	B:=A;
Write(A)	
	Write(B)

Escala 1: A:=10;b:=4;

T_1	T_2
	Read(A)
Read(B)	B:=A;
A:=B;	Write(B)
Write(A)	

Escala 2: A:=10; b:=4;

Serialização

- Uma escala de execução concorrente deve ser equivalente a uma escala seqüencal. Para isto, realiza-se a serialização, que pode ser:
  - Serialização de conflito
  - Visão serializada

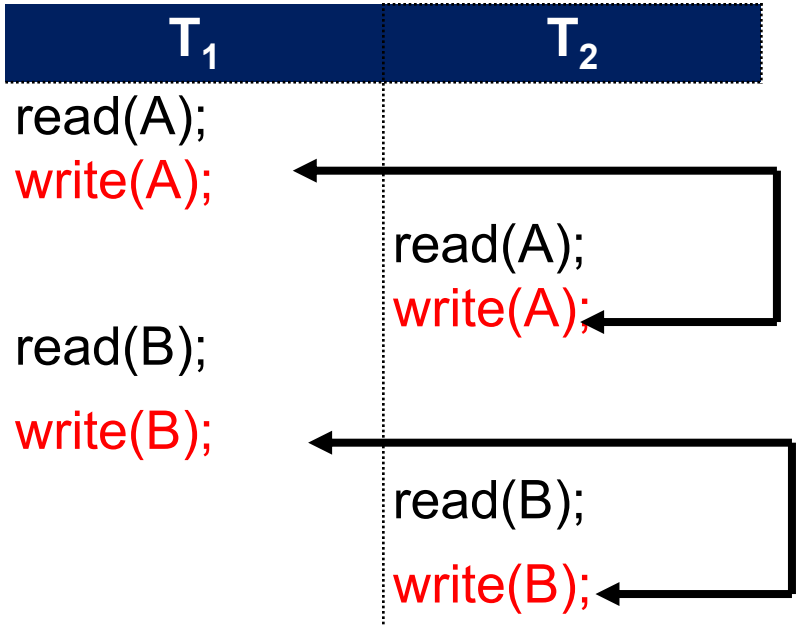
Serialização de Conflito

- Seja Ii e Ij instruções das transações Ti e Tj, respectivamente.Se Ii e Ij referem-se a dados diferentes:
  - então podem ser alternadas;
  - caso contrário, a ordem das instruções pode fazer diferença. Neste caso, há 4 possibilidades:

L_i	L_j	
read(Q)	read(Q)	A ordem NÃO importa
Read(Q)	Write(Q)	A ordem importa
Write(Q)	Read(Q)	A ordem importa
Write(Q)	Write(Q)	A ordem não importa para Ii e Ij, mas importa para o Banco de Dados em si.

Noção de Conflito

- Duas instruções entram em conflito caso elas sejam operações pertencentes a diferentes transações, agindo no mesmo dado, e pelo menos uma destas instruções é um write.



Escala Conflito Serializável

- Se uma escala S pode ser transformada em outra S', por uma série de trocas de instruções não conflitantes, dizemos que S e S' são equivalentes no conflito
- Uma escala S é conflito serializável se ela é equivalente no conflito a uma escala de execução seqüencial

T_1	T_2
read(A);	
write(A);	
read(B);	
write(B);	
	read(A);
	write(A);
	read(B);
	write(B);

T_1	T_2
read(A);	
write(A);	
	read(A);
	write(A);
read(B);	
write(B);	
	read(B);
	write(B);

Visão Serializada

- Duas escalas S e S' são esquivalentes na visão se :
  - cada transação lê os mesmos valores em ambas as escalas e executa a mesma computação.
  - resultam o mesmo resultado final.

T_1	T_2
read(A);	
write(A);	
read(B);	
write(B);	
	read(A);
	write(A);
	read(B);
	write(B);

T_1	T_2
read(A);	
write(A);	
	read(A);
	write(A);
read(B);	
write(B);	
	read(B);
	write(B);

Visão Serializada

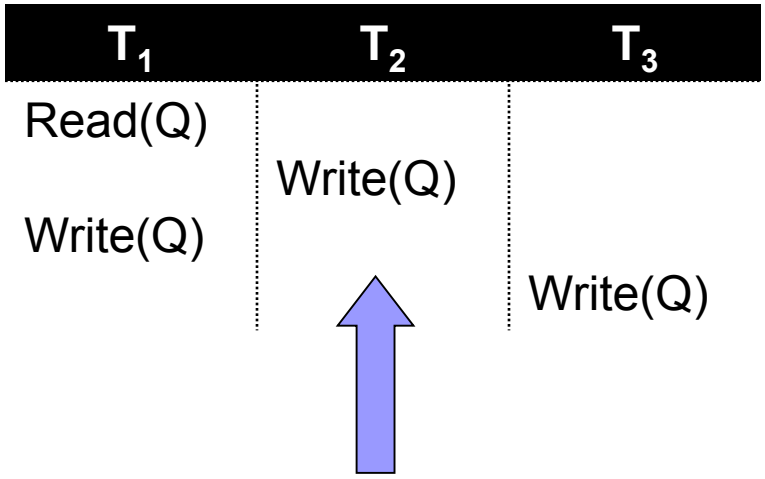
- Exemplo de escalas **não equivalentes** na visão:

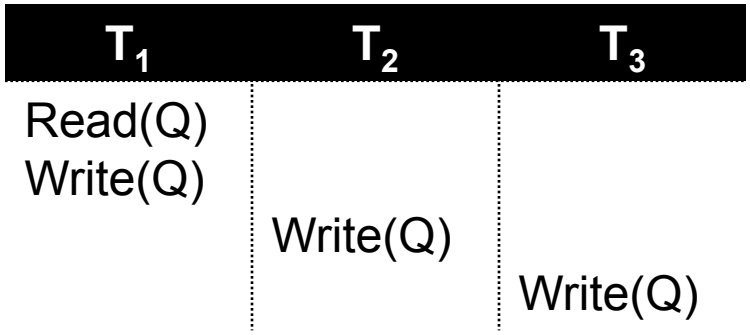
T_1	T_2
read(A);	
A:=A-50;	
write(A);	
read(B);	
B:=B+50;	
write(B);	
	read(A);
	temp:=A*0,1;
	A:=A-temp;
	write(A);
	read(B);
	B:=B+temp;
	write(B);

T_1	T_2
	read(A);
	temp:=A*0,1;
	A:=A-temp;
	write(A);
	read(B);
	B:=B+temp;
	write(B);
read(A);	
A:=A-50;	
write(A);	
read(B);	
B:=B+50;	
write(B);	

Visão Serializada

- Uma escala tem visão serializada se for equivalente em visão a um escala de execução seqüencial

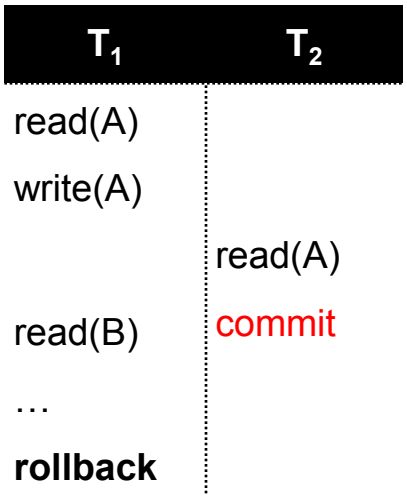




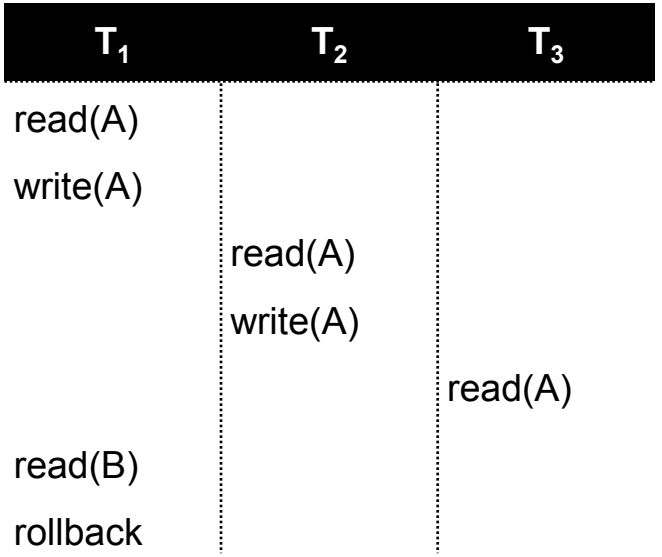
Escrita cega é quando uma transação escreve no BD sem levar em conta o valor antigo de um dado

Problemas em Escalas

Escala não recuperável



Escala com retorno em cascata



Em **escalas recuperáveis**, nenhuma transação pode ser efetivada antes que todas as outras que leem seus dados o sejam

Um **retorno em cascata** ocorre quando uma transação tem que ser desfeita porque leu um dado de uma transação que falhou

Controle de Concorrência

"Quando diversas transações são executadas de modo concorrente em um BD, a propriedade de isolamento pode não ser preservada. É necessário que o sistema controle a interação entre transações concorrentes; esse controle é alcançado por meio de uma larga gama de mecanismos chamados esquemas de controle de concorrência".

[Silberschatz99]

Controle de Concorrência

- Protocolos com Base em Bloqueios (lock)
- Protocolos com Base em *Timestamp*
- Protocolos com Base em Validação
- Esquemas de Multiversão

Bloqueio

- Impede o acesso a um determinado item de dado;
- Pode ser:
  - Compartilhado (S) - pode ler, mas não escrever
  - Exclusivo (X) - pode ler e escrever

Compatibilidade de Bloqueio

	Compartilhado	*Exclusivo*
Compartilhado	**sim**	não
Exclusivo	não	não

Exemplo de Transações com Bloqueios

T_1	T_2

lock-X(B);
read(B);
B:=B-50;
write (B):
unlock(B);
Lock-X(A);
read(A);
A:=A+50;
write(A);
unlock(A);
	lock-S(A);
	read(A);
	unlock(A);
	lock-S(B);
	read(B);
	unlock(B);
	display(A+B);

Escalas Concorrentes com Bloqueios

T_1	T_2
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);
	display(A+B);
Lock-X(A);	
read(A);	
A:=A+50;	
write(A);	
unlock(A);	

Deadlock

T_1	T_2
lock-X(B);	
read(B);	
B:=B-50;	
write (B):	
	lock-S(A);
	read(A);
	unlock(A);
	lock-S(B);
unlock(B);	
Lock-X(A);	
read(A);	
A:=A+50;	
write(A);	read(B);
unlock(A);	unlock(B);
	display(A+B);

DEADLOCK

Deadlock

- São impasses, situações em que uma transação não pode ser executada porque espera a liberação de dados que estão bloqueados por outra.

- São tratados através de reversões (*rollbacks*), em que uma das transações é desfeita. Quando uma transação é desfeita, seus itens de dados são liberados e podem, então, ser avaliados por outras transações, que podem continuar suas execuções.

Política de Concessão de Bloqueios

- Uma transação obtém bloqueio sobre um dado quando:
  - não existe nenhuma outra transação com bloqueio sobre este mesmo dado, de modo conflitante;
  - não existe nenhuma outra transação que esteja esperando um bloqueio sobre o dado e que tenha feito sua solicitação de bloqueio antes desta transação, que está solicitando bloqueio.

Protocolo de Bloqueio em duas Fases

- Fase de Expansão
  - Uma transação pode obter bloqueios, mas não pode liberar nenhum
- Fase de Encolhimento
  - Uma transação pode liberar bloqueios, mas não consegue obter nenhum bloqueio novo

Exemplo de Protocolos de Bloqueios

Com 2 fases

T_1
lock-X(B);
read(B);
B:=B-50;
write (B);
Lock-X(A);
read(A);
A:=A+50;
write(A);
unlock(B);
unlock(A);

Sem 2 fases

T_1
lock-X(B);
read(B);
B:=B-50;
write (B):
unlock(B);
Lock-X(A);
read(A);
A:=A+50;
write(A);
unlock(A);

Variações do bloqueio em duas fases:

- Severo
  - Todos os bloqueios em modo exclusivo tomados por uma transação devem ser mantidos até que a transação seja efetivada.
- Rigoroso
  - Todos os bloqueios devem ser mantidos até que a transação seja efetivada

Refinamentos do Protocolo de Bloqueio em 2 fases:

- Conversão de bloqueio
  - Upgrade

    Mecanismo para promover um bloqueio compartilhado em exclusivo

  - Downgrade

    Mecanismo para rebaixar um bloqueio exclusivo para compartilhado

Exemplo

T_1	T_2
read(a_1);	read(a_1);
read(a_2);	read(a_2);
...	display(a_1+a_2);
read(a_n);	
write (a_1);	

T_1	T_2
lock-S(a_1)	
	lock-S(a_1)
lock-S(a_2)	
	lock-S(a_2)
lock-S(a_3)	
lock-S(a_4)	
	unlock(a_1)
	unlock(a_2)
lock-S(a_n)	
upgrade(a_1)	

Deadlock

“Um sistema está em deadlock se há um conjunto de transações, tal que toda transação desse conjunto está esperando outra transação nele contida. “

Exemplo de Deadlock

Transação A	Tempo	Transação B
Lock-x(A)	T_1	
	T_2	Lock-x(B)
Lock-x(B)	T_3	
	T_4	Lock-x(A)

Métodos para manuseio de Deadlock

- Prevenção de deadlock
- Tempo esgotado

Prevenção de Deadlock

- Nenhum ciclo de espera poder ocorrer
  - ordenação de solicitações de bloqueios
  - bloqueios concedidos de uma vez só
- Timestamps
  - esperar-morrer
  - ferir-esperar

Esperar-Morrer

- Técnica não-preemptiva
- Uma **transação pode esperar pelos dados de outra**, somente se possuir *timestamp* menor, isto é, **se for mais antiga** que a outra transação
- Transações com *timestamp* maiores, nunca esperam; ao contrário, são revertidas (mortas)
- Uma transação revertida não recebe um novo *timestamp*

Ferir-esperar

- Técnica preemptiva
- Uma **transação pode esperar** pelos dados de outra, somente se possuir *timestamp* maior, isto é, **se for mais nova que a outra transação**
- Transações com *timestamp* menores, nunca esperam, são desfeitas (feridas)
- Uma transação desfeita recebe um novo *timestamp* (reversões)

Tempo Esgotado (*timeout*)

- Uma transação pode esperar por um dado por um determinado tempo;
- Caso o bloqueio não seja concedido dentro deste intervalo de tempo, a transação é revertida e reiniciada
- Difícil determinar o intervalo de tempo apropriado
- Pode ocorrer inanição, isto é, uma transação é sempre desfeita, sendo impedida de continuar seu processamento

MySQL e Bloqueios

- Mysql trabalha com bloqueio de tabelas e de registros, dependendo do tipo de tabela usado
- Para testar o isolamento a nível de tabelas use:

  LOCK TABLE tabela **READ|WRITE;**

- **Exemplo:**

  LOCK TABLE ALUNO READ;

  UPDATE ALUNO SET CODCURSO='SS' WHERE RA='123'
- **Erro obtido:**

  Table 'aluno' was locked with a READ lock and can't be updated

Nível de Isolamento das Transações

- Para isolamento a nível de registro, o MySQL oferece 4 níveis de isolamento para transações:
  - Read uncommited
  - Read commited
  - Repeatable read
  - Serializable

READ UNCOMMITED

- Conhecida também como *“dirty read”*
- SELECTs sem bloqueio são realizados de forma a não procurar por uma possível versão mais nova de um registro; assim as leituras não são 'consistentes' sob este nível de isolamento; de outra forma este nível funciona como READ COMMITTED.

READ COMMITED

- Todas as instruções SELECT ... FOR UPDATE e SELECT ... LOCK IN SHARE MODE só travam o registro de índice, não a lacuna antes dele e assim permite livre inserção de novos registros próximo ao registro travado.
- Mas ainda no tipo de faixa UPDATE e DELETE, o InnoDB deve definir lock da chave seguinte ou da lacuna e bloquear inserções feitas por outros usuários nas lacunas cobertas pela faixa. Istó é necessário já que deve se bloquear ``linhas fantasmas'' para a replicação e recuperação no MySQL funcionar.
- Suportam Leituras consistentes (Consistent reads), i.e., cada leitura consistente, mesmo dentro da mesma transação, configura e lê a sua própria cópia recente.

REPEATABLE READ

- Este é o **nível de isolamento** padrão do InnoDB. SELECT ... FOR UPDATE, SELECT ... LOCK IN SHARE MODE, UPDATE, e DELETE que utilizam um índice único com uma condição de busca única, travam apenas o registro de índice encontrado, e não a lacuna antes dele. Nos outros casos, estas operações empregam travamento de registro seguinte, bloqueando a faixa de índice varrida com trava de chave seguinte ou de lacuna e bloqueando novas inserções feitas por outros usuários.
- Em leituras consistentes (**consistent reads)** existe uma diferença importante do nível de isolamento anterior: neste nível todas as leituras consistentes dentro da mesma transação lêem a mesma cópia estabelecida pela primeira leitura. Esta conversão significa que se você executa diversas SELECTs dentro da mesma transação, elas também são consistentes entre elas.

SERIALIZABLE

- Este nível é como o REPEATABLE READ, mas todos os SELECTs são convertidos implicitamente para SELECT ... LOCK IN SHARE MODE.

Leitura Consistente

- Uma leitura consistente significa que o InnoDB utiliza *multi-versioning* para apresentar a uma consulta uma cópia do banco de dados em um dado momento.
- A consulta verá as mudanças feitas por aquelas transações que fizeram o *commit* antes daquele momento e não verá nenhuma mudança feita por transações posteriores.
- A exceção a esta regra é que a consulta verá as mudanças feitas pela transação que executar a consulta.
- Uma leitura consistente não configura nenhuma trava em tabelas que ela acessa e assim outros usuários estão livres para modificar estas tabelas ao mesmo tempo que uma leitura consistente esta sendo feita na tabela.

Lock de Leitura

- SELECT ... LOCK IN SHARE MODE
  - Realizar uma leitura em modo compartilhado significa que lemos o dado disponível por último e configuramos travas de leitura nos registros lidos. Se este dado pertencer a uma transação de outro usuário que ainda não fez commit, esperaremos até que o commit seja realizado. Uma trava em modo compartilhado previne que ocorra atualizações ou deleções de registros já lidos
- SELECT ... FOR UPDATE
  - Executa o mesmo bloqueio que uma instrução UPDATE SQL

Instruções SQL e Locks

Instrução SQL	Lock
SELECT ... FROM ...	Sem travas
SELECT ... FROM ... LOCK IN SHARE MODE	atribui travas de **chave seguinte** compartilhadas em todos os registros de índices que a leitura encontrar
SELECT ... FROM ... FOR UPDATE	atribui travas de chave seguinte compartilhadas em todos os registros de índices que a leitura encontrar
INSERT INTO ... VALUES (...)	atribui uma trava exclusiva em registros inseridos
UPDATE ... SET ... WHERE	atribui trava de chave seguinte exclusiva em todos os registros que a busca encontrar.

Como lidar com Deadlocks em MySQL

- SHOW INNODB STATUS
- Esteja preparado para reexecutar uma transação se ela falhar em um deadlock.
- Commit sua transações com frequência.
- Prefira transações pequenas, pois elas têm menos chances de colidir.
- Utilize níveis de isolamento baixos em locks de leitura
- Use, mas não abuse dos locks

Tuning locks

- Usar facilidades para leituras longas
- Eliminar lock desnecessários

Sistema de Recuperação

“Uma parte integrante de um sistema de banco de dados é o esquema de recuperação que é responsável pela restauração do banco de dados para um estado consistente existente antes da ocorrência de uma falha”

Falha de Transação

- Erro Lógico (detectada no programa)
  - entrada inadequada
  - dado não encontrado
  - overflow/limite de recurso excedido
- Erro de Sistema
  - deadlock

Queda do Sistema

- Mau funcionamento do hardware
- Bug no SGBD/SO que cause a perda do conteúdo no armazenamento volátil

Falha de Disco

- Falha durante operação de transferência de dados

Tipos de Armazenamento

- Armazenamento volátil
- Armazenamento não-volátil
- Armazenamento estável
  - extremamente improvável a perda de dados

Recuperação Baseada em LOG

- LOG = Seqüência de registros de log
- Registro de log descreve uma única escrita no banco de dados, através das seguintes informaçõs:
  - identificador da transação
  - identificador do item de dado
  - valor antigo
  - valor novo

Exemplos de registros de log

- $< T_1$ start$>$
- $<T_1, X_i, V_1, V_2>$
- $<T_1$ comit$>$
- $<T_1$ abort$>$

Modo de atualização de BD

- Modificação adiada/postergada
  - adia todas os writes de uma transação até sua efetivação parcial
- Modificação imediata
  - realiza os writes a medida que a transação vai sendo executada (estado ativo)

Modificações adiadas

T_1
read(A);
A:=A-50;
write(A);
read(B);
B:=B+50;
write(B);

T_2
read(C);
C:=C-100;
write(A);

Log	Banco de Dados A=1000;B=2000;C=700
$<T_1$ start>	
$<T_1$, A, 950>	
$<T_1$, B, 2050>	
$<T_1$ commit>	
	A=950
	B=2050
$<T_2$ start>	
$<T_2$, C, 600>	
$<T_2$ commit>	
	C=600

Esquema de Recuperação REDO

- Refaz a transação
- Ajusta os valores de todos os dados atualizados pela transação para seus valores novos
- Idempotente
  - executá-la várias vezes deve ser equivalente a executá-la uma vez só.

Exemplos de Recuperação/REDO

Log	BD
	A=1000
	B=2000
	C=700
$<T_1$ start>	
$<T_1$, A, 950>	
$<T_1$, B, 2050>	

NÃO FAZ NADA

Log	BD
	A=1000
	B=2000
	C=700
$<T_1$ start>	
$<T_1$, A, 950>	
$<T_1$, B, 2050>	
$<T_1$ commit>	
	A=950
	B=2050
$<T_2$ start>	
$<T_2$, C, 600>	

redo(T_1)

Log	BD
	A=1000
	B=2000
	C=700
$<T_1$ start>	
$<T_1$, A, 950>	
$<T_1$, B, 2050>	
$<T_1$ commit>	
	A=950
	B=2050
$<T_2$ start>	
$<T_2$, C, 600>	
$<T_2$ commit>	
	C=600

redo(T_1), redo (T_2)

Modificação Imediata de BD

T_1
read(A);
A:=A-50;
write(A);
read(B);
B:=B+50;
write(B);

T_2
read(C);
C:=C-100;
write(A);

Log	Banco de Dados A=1000;B=200;C=700
$<T_1$ start>	
$<T_1$, A, 1000, 950>	
	A=950
$<T_1$, B, 2000, 2050>	
	B=2050
$<T_1$ commit>	
$<T_2$ start>	
$<T_2$, C, 700, 600>	
	C=600
$<T_2$ commit>	

Esquema de Recuperação UNDO

- Desfaz a transação
- Retorna os valores antigos de todos os dados atualizados pela transação para seus valores novos
- Idempotente
  - executá-la várias vezes deve ser equivalente a executá-la uma vez só.

Esquema UNDO/REDO

- UNDO
  - apenas $<T_i$ start$>$
- REDO
  - $<T_i$ start$>$ + $<T_i$ commit$>$

Exemplos de Recuperação UNDO/REDO

Log	BD
	A=1000
	B=2000
	C=700
<T_1 start>	
<T_1, A, 1000, 950>	
<T_1, B, 2000,2050>	

undo(T_1)

Log	BD
	A=1000
	B=2000
	C=700
<T_1 start>	
<T_1, A, 1000, 950>	
<T_1, B, 2000, 2050>	
<T_1 commit>	
	A=950
	B=2050
<T_2 start>	
<T_2 , C, 700,600>	

redo(T_1), undo (T_2)

Log	BD
	A=1000
	B=2000
	C=700
<T_1 start>	
<T_1, A, 1000, 950>	
<T_1, B, 2000, 2050>	
<T_1 commit>	
	A=950
	B=2050
<T_2 start>	
<T_2 , C, 700, 600>	
<T_2 commit>	
	C=600

redo(T_1), redo (T_2)

Checkpoint

- Registros de log que indicam a saída para armazenamento estável de todos os registros residentes na memória principal
- Dinamiza consideravelmente o esquema de recuperação, pois as transações efetivadas antes do último checkpoint não precisam ser avaliadas

Exemplo de Checkpoint

Log

<T_1 start>

<T_1, A, 950>

<T_1 commit>

<T_2 start>

<T_1, B, 2050>

<T_2 commit>

<checkpoint>

<T_3 start>

<T_3 , C, 600>

<T3 commit>

redo ou undo T_3, dependendo do método de atualização do BD

Tuning do Subsistema de Recuperação

1. Colocar os *logs* num disco dedicado para evitar buscas
2. Adiar escritas no BD o máximo possível
3. Ajustar o tempo de recuperação desejado quando configurar os intervalos de *dump* e *chekpoint* do BD
4. Reduzir o tamanho de transações de atualização

Tuning de Escrita em BD

- Alguns dados nunca precisam ser escritas em discos
  - Suponha um item x tem um valor inicial 3.
  - Uma transação T_1 atualiza o valor de x para 5 e efetiva.
  - O buffer do BD e arquivo de log de x*, mas o BD ainda guarda o valor 3 para x*
  - Uma transação subsequente *T_2 muda o valor de x para 11 e efetiva*, atualizando então o buffer do BD e arquivo de log de x para 11 também
  - Quando x é escrito no BD, será escrito com o valor 11, sem nunca ter recebido em disco o valor 5

this is a virtue, since we have avoided an entire write to the database disks without jeopardizing recoverability

Considerações sobre o SO

- Agendamento de processos/threads
- Controla acesso concorrente ao BD pelos threads de usuários
- Gerencia a memória física e virtual
- Gerencia arquivos