

Processamento de Transações em BD

Banco de Dados II

Bianca Maria Pedrosa

Última atualização 16/05/2016



Índice

- Processamento de Transações
- Controle de Concorrência
- Sistema de Recuperação



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 concorrentemente

- 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**

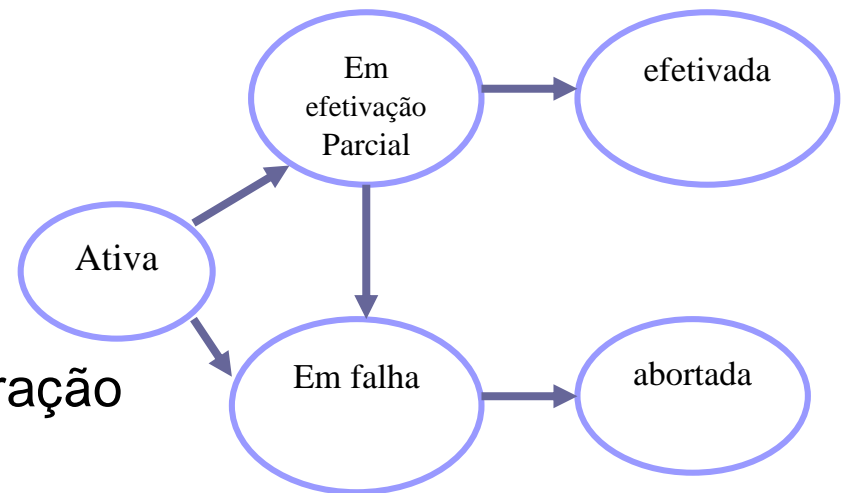
- ☐ execução normal não pode se realizar

- **Em efetivação**

- ☐ após a conclusão com sucesso

- **Abortada**

- ☐ após transação desfeita, BD = estado inicial



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 Start Transaction serão desfeitas

Exemplos de Transações

T_1 transfere 100 da conta A para a conta B

```
T1  
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

```
T2  
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); 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 2

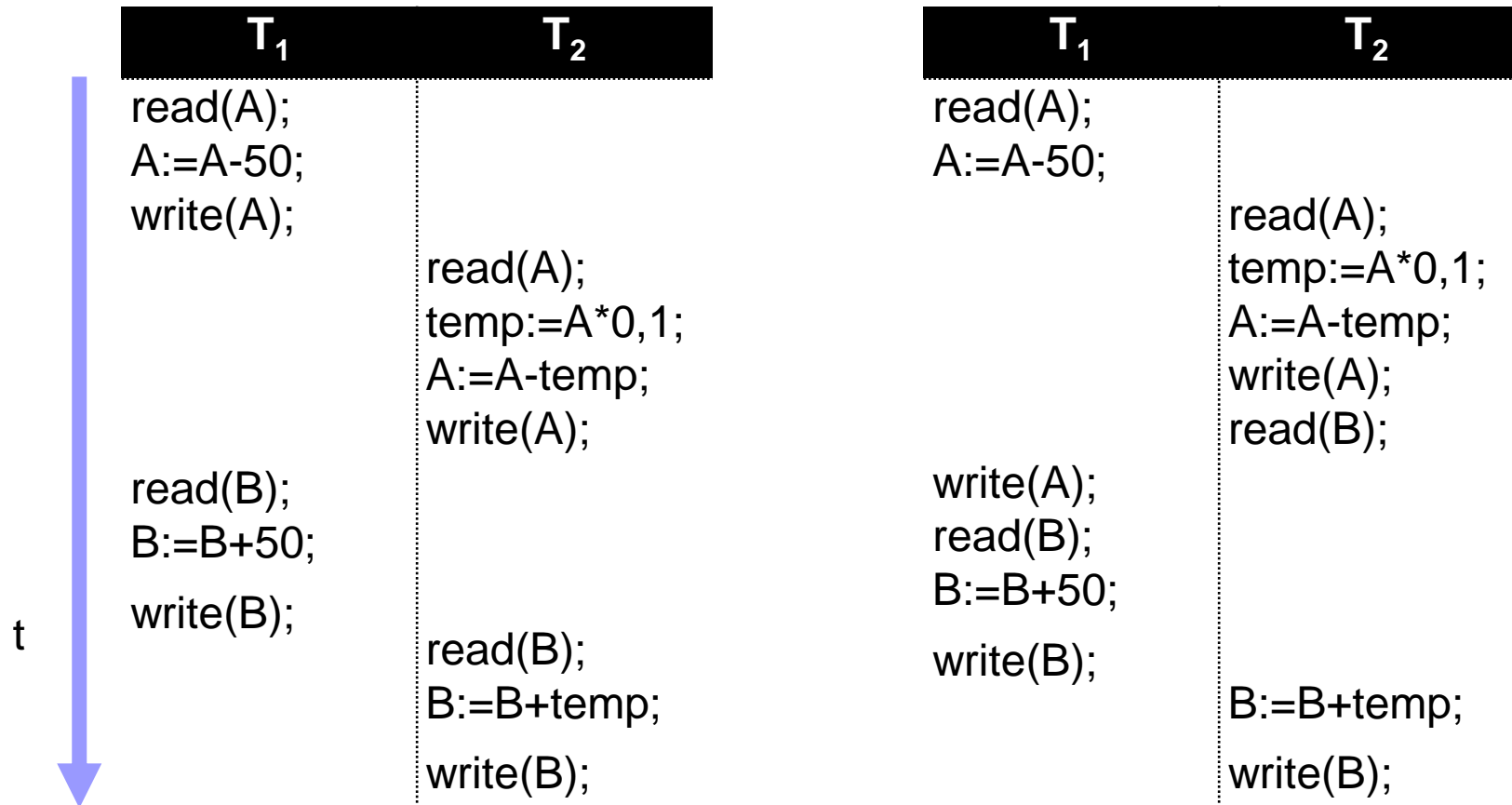


Execuções Concorrentes

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

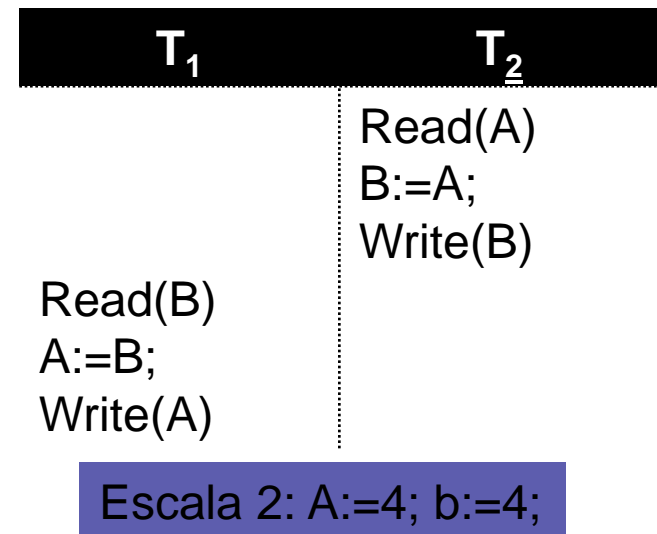
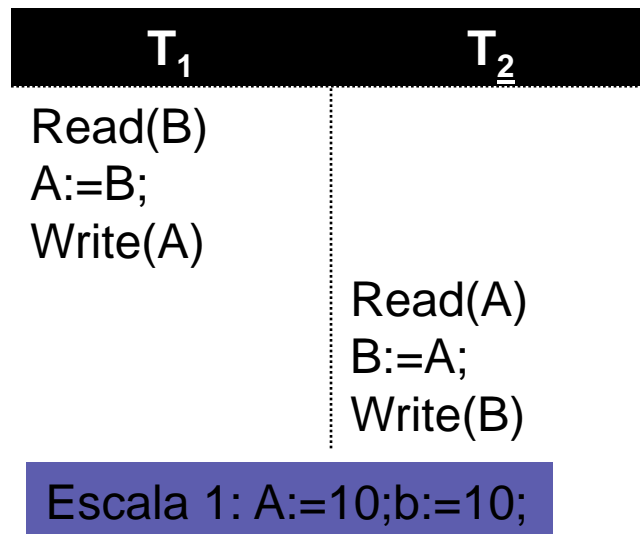


Concorrência e Consistência do BD

- 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

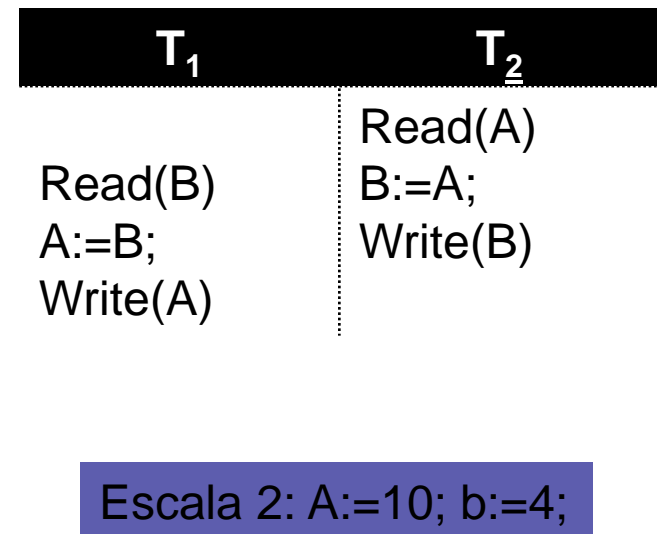
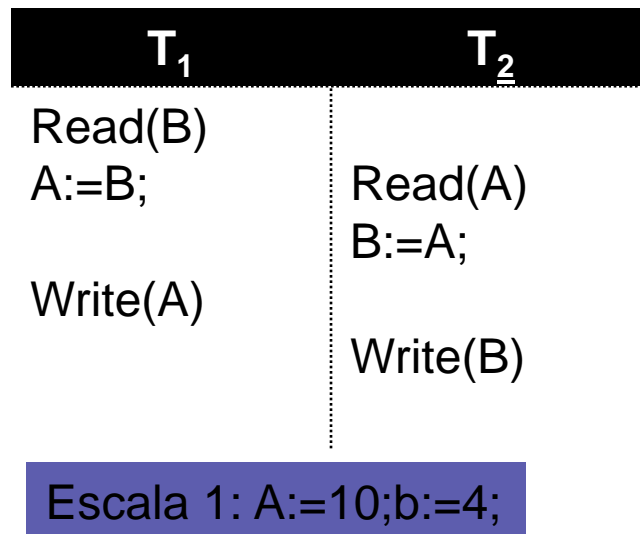
Teste de Consistência

- Sejam duas transações $T_1:A:=B$; $T_2:B:=A$;
- 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



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





Serialização

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

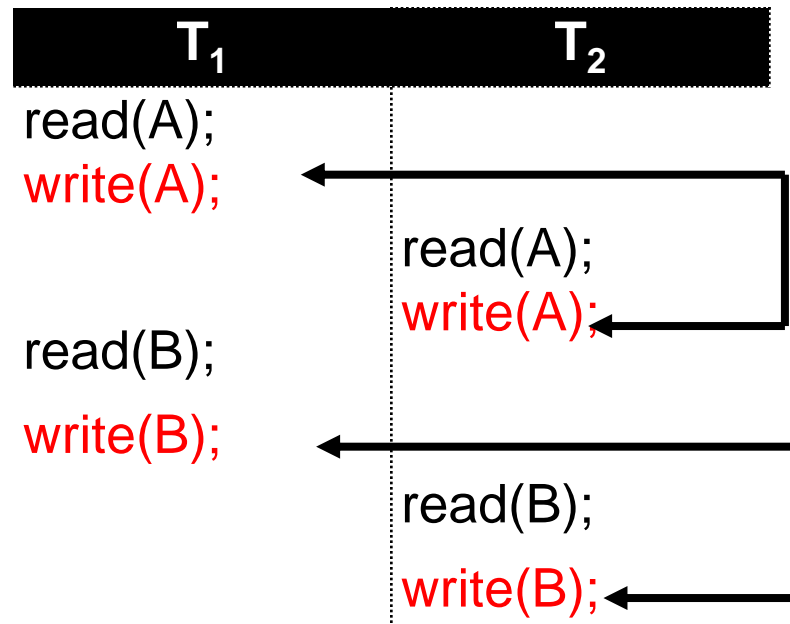
Serialização de Conflito

- Seja l_i e l_j instruções das transações T_i e T_j , respectivamente. Se l_i e l_j 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 l_i e l_j , mas importa para o Banco de Dados em si.

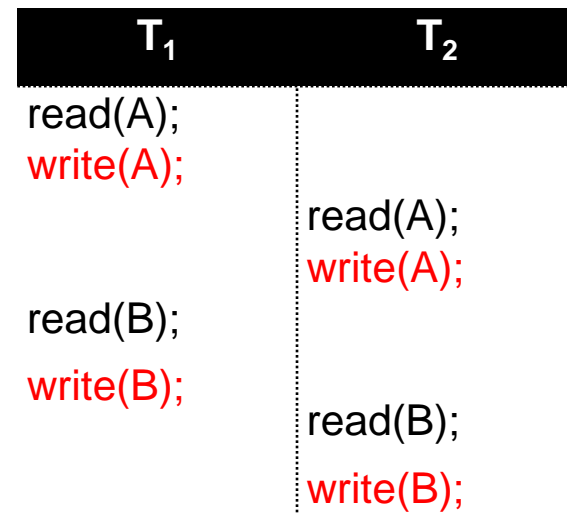
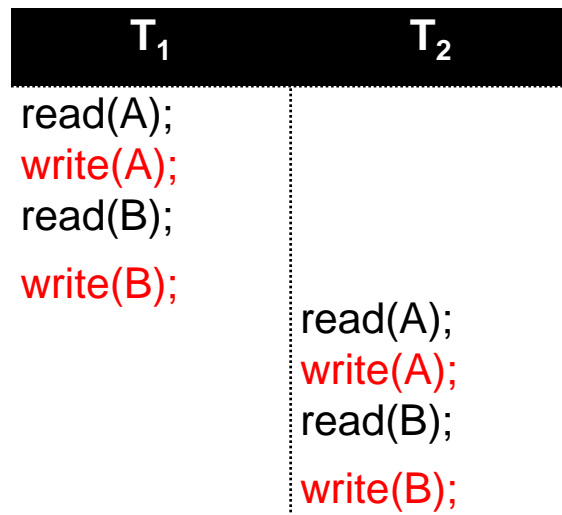
Noção de Conflito

- 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.



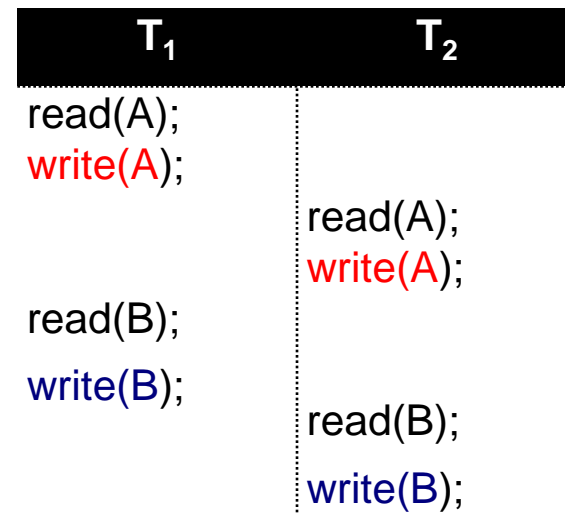
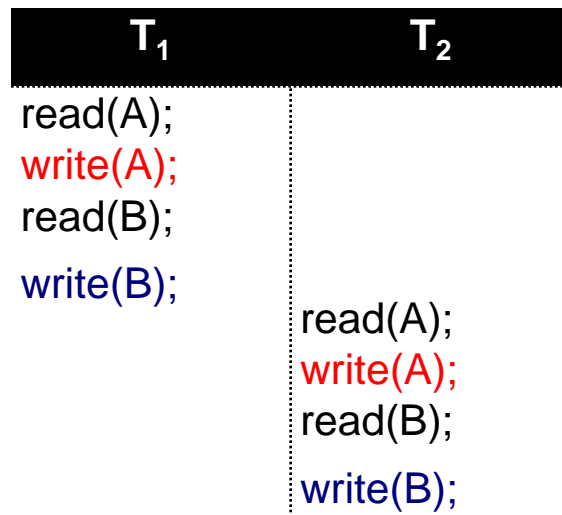
Escala Conflito Serializável

- 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



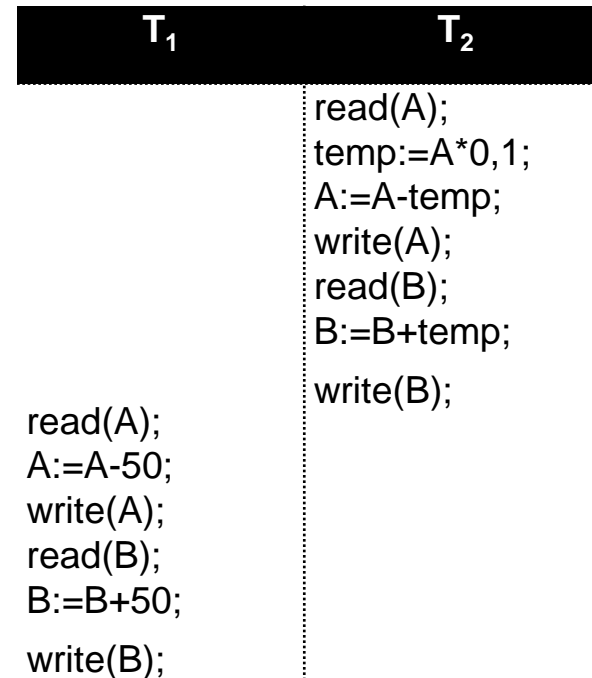
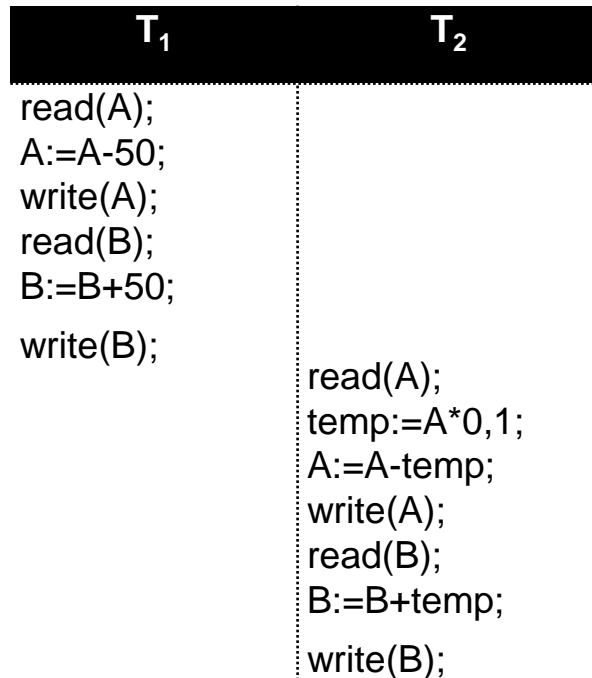
Visão Serializada

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



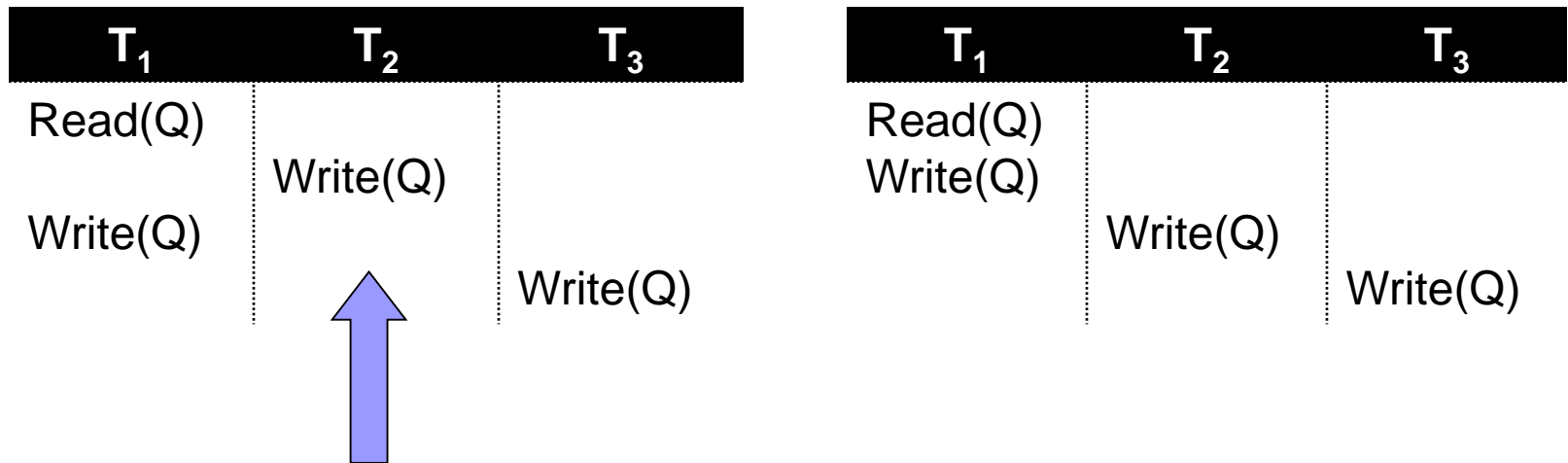
Visão Serializada

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



Visão Serializada

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

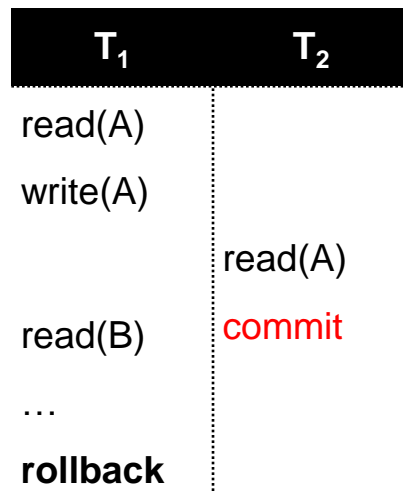


Escrita cega é quando uma transação escreve no BD sem levar em conta o valor antigo de um dado

Escalas serializáveis em conflito são sempre serializáveis em visão, mas não o contrário. Este é o caso das escalas com escritas cegas.

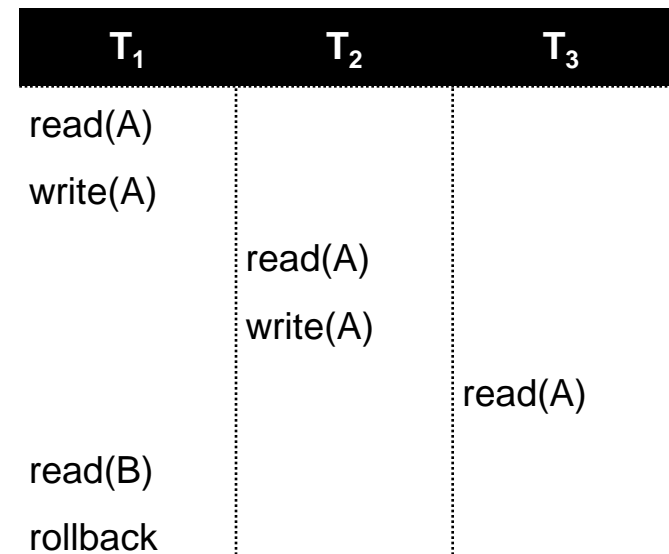
Problemas em Escalas

Escala não recuperável

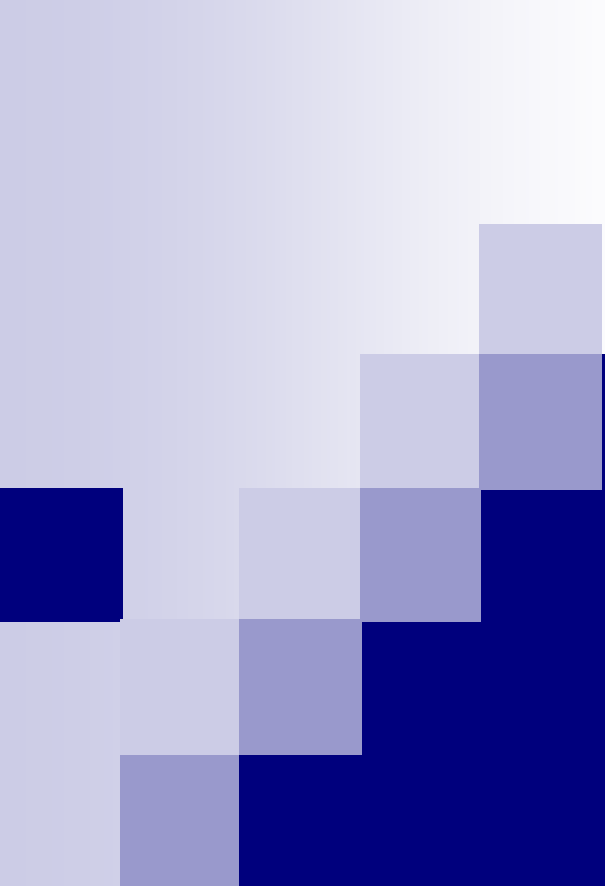


Em **escalas recuperáveis**, nenhuma transação pode ser efetivada antes que todas as outras que leem seus dados o sejam

Escala com retorno em cascata



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]

Mecanismos para controle de concorrência

- Bloqueios (*lock*)
 - Restrições de acesso aos dados manipulados por uma transação. É o mecanismo mais usado em SGBD.
- *Marcas de tempo (Timestamp)*
 - *Existem mecanismos que se baseiam na ordenação de timestamps das transações para assegurar a serialização das transações*
- Multiversão
 - Alguns algoritmos para controle de concorrência usam múltiplas versões de um item de dado e são capazes de determinar a versão mais apropriada para cada transação



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

	<i>Compartilhado</i>	<i>Exclusivo</i>
<i>Compartilhado</i>	sim	não
<i>Exclusivo</i>	não	não

Exemplo de Transações com Bloqueios

T ₁	T ₂
<pre>lock-X(B); read(B); B:=B-50; write (B); unlock(B); Lock-X(A); read(A); A:=A+50; write(A); unlock(A);</pre>	<pre>lock-S(A); read(A); unlock(A); lock-S(B); read(B); unlock(B); display(A+B);</pre>

Escalas Concorrentes com Bloqueios

T ₁	T ₂
<pre>lock-X(B); read(B); B:=B-50; write (B); unlock(B);</pre>	<pre>lock-S(A); read(A); unlock(A); lock-S(B); read(B); unlock(B); display(A+B);</pre>
<pre>Lock-X(A); read(A); A:=A+50; write(A); unlock(A);</pre>	



Granularidade de bloqueio

- Bloqueio de tabela
- Bloqueio de linha /registro



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

T₁

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

T₁

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

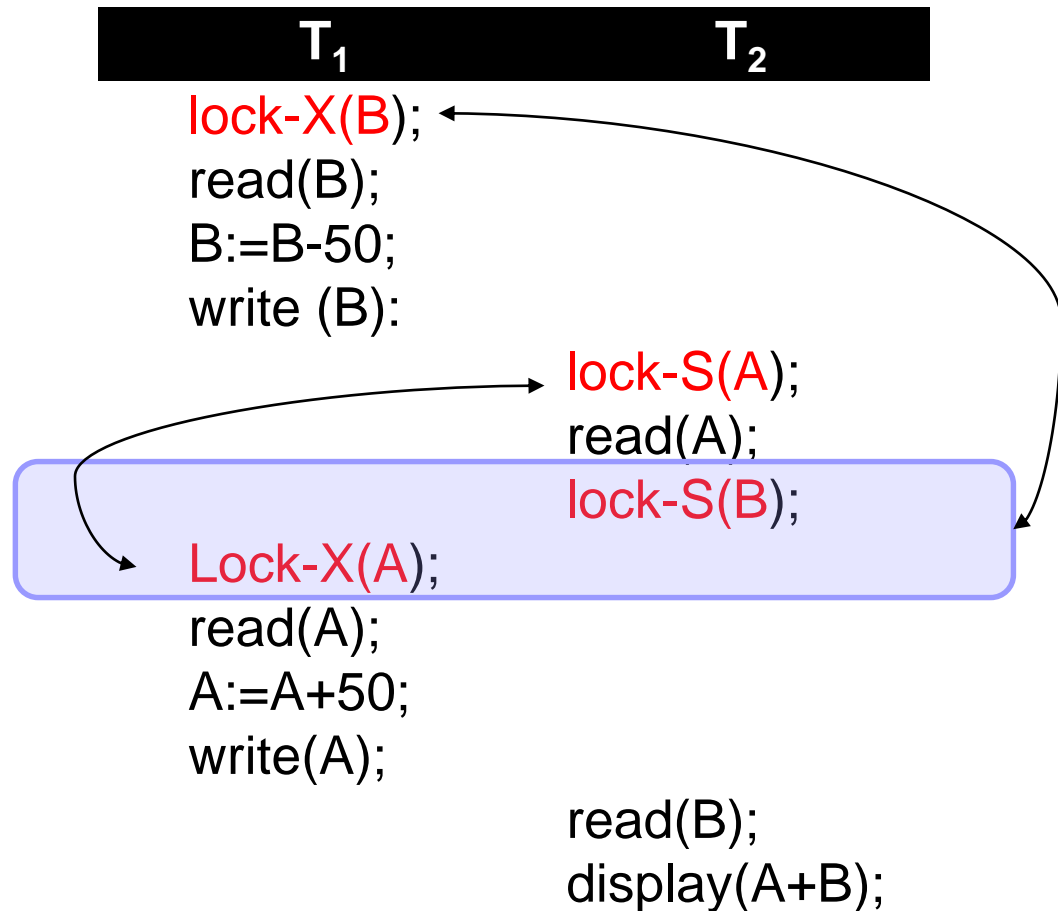
“Um sistema está em deadlock se há um conjunto de transações, tal que toda transação desse conjunto está esperando outra transação nele contida. “



Deadlock

- São impasses, situações em que cada transação, de um conjunto de transações, espera a liberação de dados que estão bloqueados por outra transação.
- 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.

Deadlock





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



Problemas em controle de concorrência

- Atualização perdida
- Leitura suja
- Leitura sem repetição
- Sumário incorreto

Atualização Perdida

- A atualização perdida ocorre quando duas transações concorrentes acessam e tentam atualizar os mesmos dados, entretanto são intercaladas de forma que uma das atualizações é perdida.

Transação 1	Tempo	Transação 2
Read P	T1	
	T2	Read P
Write P	T3	
	T4	Write P

Leitura Suja

- Leitura suja ocorre quando uma transação atualiza um dado e depois vem a falhar, mas outra transação lê este dado antes que o seu valor volte ao valor original.

Transação 1	Tempo	Transação 2
Read P	T1	
Write P	T2	
	T3	Read P
	T4	Write P
Read P	T5	
	T6	

Leitura não-repetível

- Leitura sem repetição ocorre quando uma transação lê um item de dado duas vezes e este item muda de valor entre as duas leituras

Transação 1	Tempo	Transação 2
Read P	T1	
	T2	Read P
	T3	Write P
Read P	T4	
Write P	T5	
	T6	



Fantasma

- Uma transação pode ler dados de uma tabela enquanto outra transação insere ou exclui dados na mesma tabela. Se esta transação for repetida, lerá dados que não fantamas, isto é, que não existiam anteriormente.

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 uncommitted
 - ☐ Read committed
 - ☐ Repeatable read
 - ☐ Serializable

READ UNCOMMITTED

- Neste nível de isolamento as transações podem visualizar os resultados de transações que já receberam *commit*.
- Este nível é raramente usado na prática, porque seu desempenho não é muito melhor do que os outros níveis, que possuem muitas vantagens.
- Ler dados não “comitados “ leva ao problema de leitura suja.

Leitura Consistente

- Uma leitura consistente significa que o InnoDB utiliza *multi-versionamento* 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.



READ COMMITED

- Uma transação verá somente alterações feitas por transações que já foram efetivadas quando ela iniciou e suas alterações não serão visíveis para outras até que ela seja efetivada.
- Permite o que é conhecido como leitura não repetível. Isso significa que você pode executar a mesma expressão duas vezes e ver dados diferentes.



REPEATABLE READ

- Este nível de isolamento garante que qualquer linha que uma transação ler vai “parecer a mesma” em leituras subsequentes dentro da mesma transação, mas na teoria ainda permite outro problema chato: leituras fantasmas.
- É o nível de isolamento de transação padrão do MySQL, mas isto pode ser alterado.

SERIALIZABLE

- É o mais alto nível de isolamento e resolve o problema da leitura fantasma forçando as transações a serem ordenadas para que eles não possam possivelmente se conflitar. Em poucas palavras, SERIALIZABLE coloca um bloqueio em cada linha que lê.
- Neste nível, muitos intervalos e contenção de bloqueio podem ocorrer. Nós raramente vemos pessoas usarem este nível de isolamento, mas as necessidades da sua aplicação podem te forçar a aceitar a baixa concorrência em favor da estabilidade dos dados em que resulta.

Níveis de Isolamento e violações

Nível de Isolamento	Leitura suja	Não-repetível	Fantasma
Read Uncommitted	X	X	X
Read Committed	-	X	X
Repeatable Read	-	-	X
Serializable	-	-	-

X ACEITA

- NÃO ACEITA

Mysql: verificando nível de isolamento

- Verificando o nível de isolamento

SELECT @@Global.tx_isolation, @@tx_isolation

- Alterando o nível de isolamento

SET SESSION tx_isolation='READ-UNCOMMITTED';

SET SESSION tx_isolation='SERIALIZABLE';

SET SESSION tx_isolation='REPEATABLE-READ';

SET SESSION tx_isolation='READ-COMMITTED';

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 suas 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

- ☐ 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 informações:
 - ☐ identificador da transação
 - ☐ identificador do item de dado
 - ☐ valor antigo
 - ☐ valor novo

Exemplos de registros de log

- $\langle T_1 \text{ start} \rangle$
- $\langle T_1, X_i, V_1, V_2 \rangle$
- $\langle T_1 \text{ comit} \rangle$
- $\langle T_1 \text{ abort} \rangle$



Modo de atualização de BD

- **Modificação adiada**

- ☐ 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

```
<T1 start>  
<T1, A, 950>  
<T1, B, 2050>  
<T1 commit>
```

```
<T2 start>  
<T2, C, 600>  
<T2 commit>
```

Banco de Dados

A=1000;B=2000;C=700

A=950
B=2050

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(T₁)	

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(T₁), redo (T₂)	

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

< T_1 start>
< T_1 , A, 1000, 950>
< T_1 , B, 2000, 2050>

< T_1 commit>
< T_2 start>
< T_2 , C, 700, 600>

< T_2 commit>

Banco de Dados
A=1000;B=200;C=700

A=950
B=2050

C=600

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 $\langle T_i \text{ start} \rangle$

- REDO

- $\langle T_i \text{ start} \rangle + \langle T_i \text{ commit} \rangle$

Exemplos de Recuperação **imediata** 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

<T₁ start>

<T₁, A, 950>

<T₁ commit>

<T₂ start>

<T₁, B, 2050>

<T₂ commit>

<checkpoint>

<T₃ start>

<T₃, C, 600>

<T3 commit>

redo ou undo T₃



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 *checkpoint* 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