Processamento de Transações

Processamento de Transações

⇒ **Transações Atômicas:** Unidades lógicas de processamento sobre um banco de dados.

⇒ **Controle de Concorrência:** Garantia de que múltiplas transações ativadas por vários usuários produzirão resultados corretos quando manipulam o banco de dados.

⇒ **Recuperação de Falhas:** Garantia de que os efeitos das transações são mantidos no banco de dados mesmo com a ocorrência de falhas.

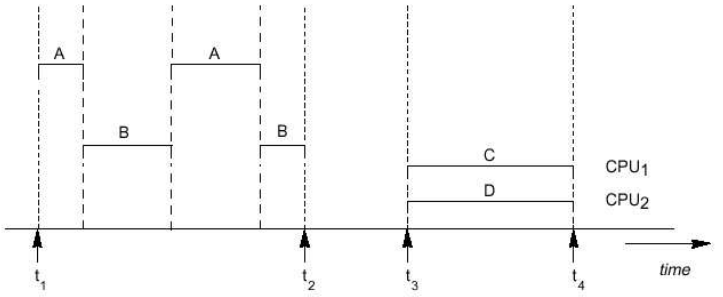
Conceitos Introdutórios

⇒ Sistemas Multiusuário X Monousuário

- Execução Intercalada :
  - Uma CPU
  - Preemptação pelo S.O.
- Simultânea: Várias CPUs

⇒ Concorrência: Vários processos disputam um recurso compartilhado

Concorrência



Processamento de Transações

⇒ Transação: Execução de um programa que acessa ou modifica o conteúdo de um banco de dados;

⇒ Modelo de Transação: Operações de leitura e escrita;

- `READ_ITEM(X)` : Leitura de um ítem de dado `X` do banco de dados a ser armazenado em uma variável de programa.
- `WRITE_ITEM(X)` : Escrita de uma variável de programa em um ítem de dados `X` do banco de dados.

Processamento de Transações

- `READ_ITEM(X)` consiste em:
  1. Encontrar o endereço do bloco de disco que contém `X`
  2. Copiar o bloco em um *buffer* de memória principal
  3. Copiar o ítem `X` do buffer em uma variável de programa
- `WRITE_ITEM(X)` consiste em:
  1. Encontrar o endereçoø do bloco de disco que contém o ítem `X`
  2. Copiar este bloco em um buffer da memória principal
  3. Copiar o conteúdo da variável de programa para a imagem do ítem `X` do buffer
  4. Armazenar o bloco atualizado no disco (imediatamente ou não)

Transações : Concorrência e Recuperação

⇒ Controle de concorrência e recuperação de falhas em um banco de dados estão relacionadas ao conceito de transação.

⇒ A execução de múltiplas transações submetidas por vários usuários deve ser controlada de tal forma que o estado final do banco de dados seja consistente.

Transações : Concorrência e Recuperação

⇒ Transação *versus* Programa : parâmetros

Transação 1	Transação 2
`read_item(X);`	`read_item(X);`
`X:=X-N;`	`X:=X+M;`
`write_item(X);`	`write_item(X);`
`read_item(Y);`	
`Y:=Y+N;`	
`write_item(Y);`	

Motivação para Controle de Concorrência

⇒ Principais problemas no controle de concorrência

- Atualização perdida
- Atualização Temporária (Leitura "Suja ")
- Sumarização Incorreta
- Leitura dupla

Motivação para Recuperação de Falhas

⇒ Quando uma transação é submetida pra execução em um SGBD, o sistema é responsável em garantir que:

a) Todas as operações da transação serão completadas com sucesso e seus efeitos serão permanentemente registrados no banco de dados, ou

b) A transação não tem efeitos no banco de dados e nem em nenhuma outra transação

⇒ Em uma transação toda ou nenhuma operação deve surtir efeito no banco de dados

Motivação para Recuperação de Falhas

⇒ Tipos de Falhas

1. Falhas de sistema de computação (hardware/software)
2. Erro na transação: erros lógicos, de programação, *overflow,*divisão por zero, interrupções etc.
3. Erros locais ou condições de execessão na aplicação
4. Manutenção do controle de concorrência
5. Falha de disco
6. Problemas fí sicos ou catástrofes

Controle de Transações

⇒ Operações de controle da transação:

- `BEGIN_TRANSACTION:` Marca o iní cio de uma transação;
- `READ` ou `WRITE:` Especificam a leitura ou escrita de um ítem no banco de dados;
- `END_TRANSACTION:` Marca o fim das operações de leitura/escrita de uma transação;
- `COMMIT_TRANSACTION:` Sinaliza que a transação terminou com sucesso e todas as operações que modificam o banco de dados estão comprometidas (*commited)* e não serão desfeitas;

Controle de Transações

- `ROLLBACK` ou (`ABORT`): Sinaliza que a transação deve terminar "sem sucesso " ou seja, todos os efeitos da transação sobre o banco de dados devem ser desfeitos;

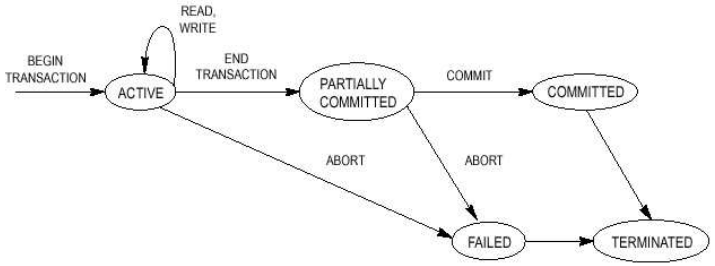
⇒ Operações para recuperação de falhas:

- `UNDO:` Usada para desfazer os efeitos de uma única operação no banco de dados;
- `REDO:` Usada quando uma operação de uma transação deve ser refeita para garantir que as alterações no banco de dados feitas por uma transação comprometida;

Estados de Uma Transação

- Ativa
- Parcialmente Comprometida
- Comprometida
- Falha
- Terminada

Estados de Uma Transação



Log (Histórico) de Sistema

⇒ Usado para permitir a recuperação de falhas em uma transação;

⇒ Registra todas as operações de uma transação que afetam os valores dos ítens de dados;

⇒ Os registros são feitos diretamente em disco;

⇒ É periodicamente copiados (*backed-up)* para um conjunto de fitas (*archieve)* que pode posteriormente ser utilizado para recuperação de dados perdidos;

Log (Histórico) de Sistema

⇒ Tipos de entrada registradas no logo de uma transação:

`T` = *Transaction-ID* — Gerado automaticamente pelo sistemas. É único para cada transação.

1. `[START_TRANSCTION,T];`
2. `[WRITE_ITEM,T,X,Valor_Antigo,Valor_Novo];`
3. `[READ_ITEM,T,X];`
4. `[COMMIT,T];`
5. `[ABORT,T];`

Pontos de Comprometimento (Commit Point)

⇒ É alcançado quando todas as operações da transação que acessam o BD são executadas com sucesso e registradas no log. A partir dai a transação é dita estar **comprometida** e assumimos que que todos os seus efeitos estão permanentemente registrados no banco de dados. Este ponto é registrado com uma entrada do tipo `[COMMIT,T]` no banco de dados.

⇒ Se para uma transação `T` uma entrada `[START_TRANSACTION,T]` é encontrada mas não uma entrada `[COMMIT,T]`, significa que a falha ocorreu antes que a transação chegasse ao ponto de comprometimento, e então todas as operações já realizadas no BD devem ser desfeitas (devem sofrer *undo).*

Pontos de Comprometimento (Commit Point)

⇒ Se a transação possui entradas `[START_TRANSACTION,T]` e `[COMMIT,T]` no log, significa que o ponto de comprometimento foi alcançado antes da falha, e deve-se verificar quais operações devem ser refeitas (devem sofrer *redo).*

⇒ *Force-Writing*: Todos os registros do log de uma transação são gravados no disco antes do commit-point.

Checkpoint

⇒ A intervalos regulares que podem ser medidos em tempo ou em números de transações comprometidas, o SGBD executa um *checkpoint* que consiste em:

1. Suspender temporariamente a execução de transações
2. Atualizar todas as operações de escrita realizadas no banco de dados
3. Registrar no log uma entrada do tipo `[CHECKPOINT]`
4. Liberar as execuções de transação

⇒ Desta forma, toda transação `T` que possui uma `[COMMIT,T]` antes de um checkpoint esta garantidamente correta no disco e não precisa ter suas operações refeitas no caso de falha no sistema.

Propriedades de Uma Transação (ACID)

⇒ Os métodos de controle de concorrência e recuperação de falhas em um SGBD devem garantir as seguintes propriedades de uma transação:

1. Atomicidade
2. Consistência Preservada
3. Isolamento
4. Durabilidade ou Permanência

Schedules de Transações

⇒ *Schedule:* Ordem para a execução de operações de transações concorrentes.

⇒ Um schedule S de n transações $T_1, T_2, T_3, \ldots, T_{n-1}, T_n$ é uma ordenação das operações das transações sujeitas a restrição de que, para cada transação T_i em S devem aparecer na mesma ordem em que elas ocorrem em T_i.

⇒ Exemplos:

S_a: $r_1(X); r_2(X); w_1(X); r_1(Y); w_2(X); c_2; w_1(Y); c_1;$

S_b: $r_1(X); w_1(X); r_2(X); w_2(X); c_2; r_1(Y); w_1(Y); c_1;$

Schedules de Transações

⇒ Duas operações em um schedule são conflitantes se:

1. Pertencem a transações diferentes
2. Acessam o mesmo ítem de dados X
3. Pelo menos uma das duas é uma escrita em X

No schedule S_a, $r_1(X)$ e $w_2(X)$ são conflitantes

$w_1(X)$ e $w_2(X)$ são conflitantes

Schedules de Transações

⇒ Um schedule S de n transações $T_1, T_2, T_3, \ldots, T_{n-1}, T_n$ é dito ser **completo** se as seguintes condições ocorrem:

1. As operações de S são exatamente aquelas em $T_1 \ldots, T_n$ inclusive as operações de commit e abort como sendo as ultimas operações de cada transação no schedule;
2. Para cada par de operações da mesma transação T_i no schedule, temos que a ordem em que estas operações na transação é preservada no schedule;
3. Para quaisquer duas operações conflitantes, uma das duas deve necessariamente ocorrer antes da outra no schedule.

Schedules de Transações

⇒ As operações não conflitantes não precisam ser ordenadas

⇒ Projeção Comprometida `C(S)` de um schedule `S`: É um schedule que contém somente as operações de `S` que pertencem a transações comprometidas.

Recuperação de Falhas

⇒ Um Schedule é dito ser recuperável se nenhuma transação `T` em `S` é comprometida até que todas as transações `T'` que tenham escrito um ítem que `T` deve ler estejam comprometidas.

⇒ Dizemos que uma transação `T` lê de uma transação `T'` em um schedule `S` se algum ítem `X` é primeiro escrito por `T'` e depois lido por `T`.

Recuperação de Falhas

⇒ No schedule `S`, `T'` não deve ser abortada antes que `T` leia o ítem `X` e não deve haver nenhuma transação que escreve `X` depois que `T'` escreva em `X` e `T` o leia (a menos que esta transação seja abortada).

NR: S_c: $r_1(X); w_1(X); r_2(X); r_1(Y); w_2(X); c_2; a_1;$

R: S_d: $r_1(X); w_1(X); r_2(X); r_1(Y); w_2(X); w_1(Y); c_1; c_2;$

⇒ Em um schedule recuperável, não é necessário se aplicar rollback em nenhuma transação comprometida.

Recuperação de Falhas

⇒ Rollback em Cascata

S_e: $r_1(X); w_1(X); \boxed{r_2(X)}; r_1(Y); w_2(X); w_1(Y); c_2; a_1; \boxed{r_2(X)}$

Em S_e, T_2 deve sofrer rollback porque T_1 foi abortada e T_2 lê de T_1.

⇒ Dizemos que um schedule **evita roolback em cascata**, se cada transação no schedule só lê ítens que foram escritos por transações comprometidas;

Recuperação de Falhas

⇒ Schedule Estrito:

- Nenhuma transação no schedule pode ler ou escrever um ítem `X` até que a última transação que escreveu `X` seja comprometida ou abortada.
- O processo de recuperação de operações de escrita passa ser uma questão de recuperar o valor anterior do ítem `X`.

⇒ Exemplo: sendo 9 o valor inicial de `X`

S_f:W_1(X,5);w_2(X,8);a_1;

Controle de Concorrência

⇒ Schedules Seriais: As operações das transações são executadas sem intercalação.

Serial: S_d:r_1(X);w_1(X);r_1(Y);w_1(Y);c_1;r_2(X);w_2(X);c_2;

Não serial: S_d:r_1(X);w_1(X);r_2(X);r_1(Y);w_2(X);w_1(Y);c_1;c_2;

⇒ Se considerarmos que cada transação é independente, a ordem em que elas são executadas não é importante, assim, **todo schedule serial é considerado correto;**

⇒ Problema principal: mau uso da CPU e perda de concorrência.

Exemplo

Schedule A

	T_1	T_2
t_1	read_item(X);	
t_2	X:=X-N;	
t_3	write_item(X);	
t_4	read_item(Y);	
t_5	Y:=Y+N;	
t_6	write_item(Y);	
t_7		read_item(X);
t_8		X:=X+M;
t_9		write_item(X);

Exemplo

Schedule B

	T_1	T_2
t_1		read_item(X);
t_2		X:=X+M;
t_3		write_item(X);
t_4	read_item(X);	
t_5	X:=X-N;	
t_6	write_item(X);	
t_7	read_item(Y);	
t_8	Y:=Y+N;	
t_9	write_item(Y);	

Exemplo

Schedule C

	T_1	T_2
t_1	read_item(X);	read_item(X);
t_2	X:=X-N;	X:=X+M;
t_3		
t_4		
t_5	write_item(X);	
t_6	read_item(Y);	
t_7		write_item(X);
t_8	Y:=Y+N;	
t_9	write_item(Y);	

Exemplo

Schedule D

	T_1	T_2
t_1	read_item(X);	
t_2	X:=X-N;	
t_3	write_item(X);	
t_4		read_item(X);
t_5		X:=X+M;
t_6		write_item(X);
t_7	read_item(Y);	
t_8	Y:=Y+N;	
t_9	write_item(Y);	

Schedules Serializáveis

⇒ Um schedule S de n transações é serializável se é equivalente a algum schedule serial das mesmas n transações.

⇒ Um schedule serializável pode ser considerado correto uma vez que é equivalente a um schedule serial que é correto

⇒ Quando dois schedules são equivalentes ?

⇒ Equivalência de resultado: Produzem o mesmo estado no banco de dados

Exemplo: Seja X=100;

S_1	S_2
read_item(X);	read_item(X);
X:=X+10;	X:=X*1.1;
write_item(X);	write_item(X);

Schedules Serializáveis

⇒ Equivalência por conflito: A ordem de quaisquer duas operações conflitantes é a mesma em ambos os schedules

⇒ Um schedule S é serializável por conflito se este schedule é equivalente por conflito a algum schedule serial S'.

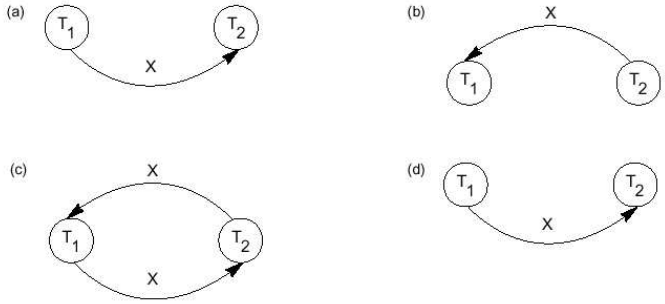
⇒ Teste de serializabilidade por conflito:

Grafo de Precedências: É um grafo dirigido $G = (N, E)$ que consiste de um conjunto de nodos $N = \{T_1, T_2, \ldots, T_n\}$ e um conjunto de arcos dirigidos $E = \{e_1, e_2, \ldots, e_n\}$. Existe um nodo no grafo para cada transação T_i no schedule. Cada arco e_i no grafo é da forma (T_j, T_k), $1 \leq j, k \leq n$, tal que uma operação em T_j, aparece no schedule antes de alguma operação conflitante em T_k.

Algoritmo de Verificação

1. *Para cada transação* T_i *que participa do schedule* S *crie um nodo* T_i *no grafo de precedência*
2. *Para cada caso em* S *onde* T_j *executa um* `read_item(X)` *depois de um* `write_item(X)` *executado por* T_i *criar um arco* (T_i,T_j) *no grafo de precedência*
3. *Para cada caso em* S *onde* T_j *executa um* `write_item(X)` *depois de um* `read_item(X)` *executado por* T_i *criar um arco* (T_i,T_j) *no grafo de precedência*
4. *Para cada caso em* S *onde* T_j *executa um* `write_item(X)` *depois de um* `write_item(X)` *executado por* T_i *criar um arco* (T_i,T_j) *no grafo de precedência*
5. *O schedule* S *é serializável se e somente se o grafo de precedências não possui ciclos;*

Grafo de Precedência



Exemplos

T_1	T_2	T_3
`read_item(X);`	`read_item(Z);`	`read_item(Y);`
`write_item(X);`	`read_item(Y);`	`read_item(Z);`
`read_item(Y);`	`write_item(Y);`	`write_item(Y);`
`write_item(Y);`	`read_item(X);`	`write_item(Z);`
	`write_item(X);`	

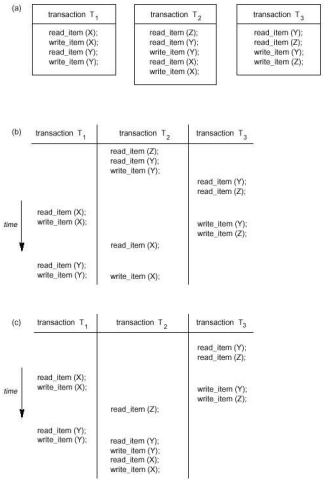
Schedule E

	T_1	T_2	T_3
t_1		`read_item(Z);`	
t_2		`read_item(Y);`	
t_3		`write_item(Y);`	
t_4			`read_item(Y);`
t_5			`read_item(Z);`
t_6	`read_item(X);`		
t_7	`write_item(X);`		
t_8			`write_item(Y);`
t_9			`write_item(Z);`
t_{10}		`read_item(X);`	
t_{11}	`read_item(Y);`		
t_{12}	`write_item(Y);`		
t_{13}		`write_item(X);`	

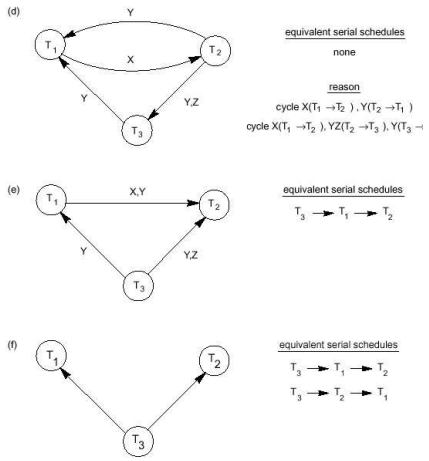
Schedule F

	T_1	T_2	T_3
t_1			read_item(Y);
t_2			read_item(Z);
t_3	read_item(X);		
t_4	write_item(X);		
t_5			write_item(Y);
t_6			write_item(Z);
t_7		read_item(Z);	
t_8	read_item(Y);		
t_9	write_item(Y);		
t_{10}		read_item(Y);	
t_{11}		write_item(Y);	
t_{12}		read_item(X);	
t_{13}		write_item(X);	

Serializabilidade - Exemplo



Serializabilidade - Exemplo



Considerações sobre Serializabilidade

⇒ Novas transações são continuamente submetidas ao longo do tempo

⇒ Na prática não é possível "rearranjar" a ordem de execução das operações após a submissão

⇒ As transações devem ser construídas de tal forma que a serializabilidade de qualquer schedule em que elas participem esteja garantida

⇒ Construção de transações deve se basear em protocolos (conjuntos de regras).