Transações

Baseado em Elmasri/Navathe (Cap. 17) e Silberschatz (Cap. 15)

Sequência de operações (sendo atualização a operação principal) que desempenha uma função específica dentro de uma aplicação.
Exemplo: uma transação bancária de transferência de valores entre contas é uma operação única para o usuário porém e composta de várias operações.

É uma unidade de execução de programa que acessa, e possivelmente atualiza, vários itens de dados em um banco de dados.
Um sistema gerenciador de banco de dados (SGBD) deve garantir a execução apropriada das transações em caso de falhas e execução simultânea.

Problemas que podem ser causados por transação T se um SGBD não garantir a execução correta de T.

Atualização Perdida

Controle de reservas de passagens aéreas de uma cia aérea: cada registro contem o número de poltronas reservada para um determinado voo.
A tabela abaixo apresenta a seguinte situação: T1 transfere N reservas de um voo, cujo numero de assentos reservados está armazenado em um item de dados chamado X, para um outro voo, cujo número de de assentos reservados está armazenado em um item de dados chamado Y.

tempo	**T1**	**T2**
0	Read(X)	
1	X=X-N	
2		Read(X)
3		X=X+M
4	Write(X)	
5	Read(Y)	
6		Write(X)
7	Y=Y+N	
8	Write(Y)	

Se duas transações T1 e T2 acessarem os mesmos itens de dados (na figura, o item X) com operações intercaladas conforme apresentado na tabela acima, qual o valor final de X?
T2 lerá o valor de X antes de T1 mudá-lo no banco, portanto o valor atualizado resultando de T1 será perdido.
Cenário:
X=80
N=5
M=4
O resultado deveria ser 79 porém é 84 pois perdeu-se a remoção dos 5 assentos.
O valor do item X será incorreto pois a atualização feita por T1 no tempo 4 foi sobrescrita (**perdida**) por T2 no tempo 6 (veja tabela acima).

Leitura Suja (Dirty Read)

Suponha agora que T1 esteja executando e no tempo 6 ocorre uma situação inesperada e T1 aborta (faz um rollback). Antes de T1 abortar, T2 leu o item X (tempo 2). Como T1 abortou, o valor de X volta a ser o original porém T2 leu o valor de X que deveria ser atualizado. Neste caso, T2 leu um valor sujo.

Sumarização Incorreta

A tabela abaixo exemplifica uma situação onde ocorre uma sumarização incorreta. A transação T3 está calculando o número total de reservas em todos os voos da cia aérea.
Durante a execução de T3, T1 é disparada (tempo 30.
Se acontecer a intercalação de operações como mostrado na tabela abaixo, o resultado de T3 não contabilizará N, pois T3 leu o valor de X depois que os N assentos foram subtraídos, mas lerá o valor Y antes que esses N assentos tenham sido adicionados a Y.

tempo	**T1**	**T3**
0		Soma=0
1		Read(A)
2		Soma+=A
3	Read(X)	:
4	X=X-N	:
5	Write(X)	:
6		Read(X)
7		Soma+=X
8		Read(Y)
9		Soma+=Y
10	Read(Y)	
11	Y=Y+N	
12	Write(Y)	

Como tratar esses problemas?

Garantindo a propriedade ACID das transações:

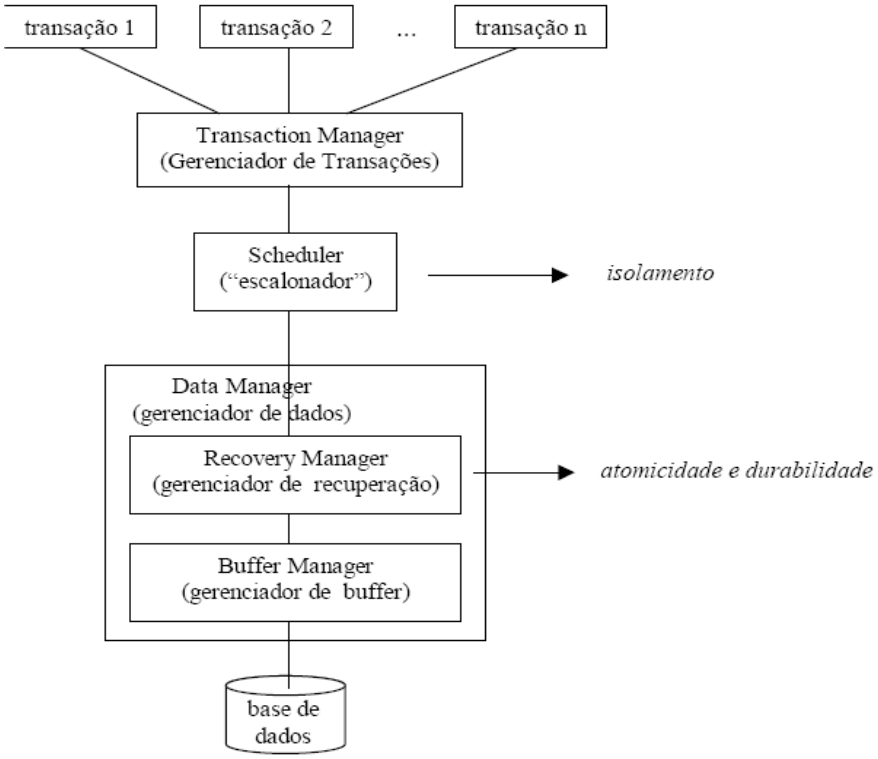
Atomicidade: a transação não pode ser dividida, ou é executada por completo ou não é executada.

Consistência: ao finalizar a transação deve manter a consistência dos dados no banco de dados, ou seja, o estado corrente do banco de dados deve respeitar o conjunto de restrições de integridade definido.

Isolamento: a execução da transação deve ocorrer sem interferência, como se a mesma estivesse executando sozinha no computador.

Durabilidade: as alterações nos dados feitas por uma transação terminada com sucesso deve ser persistido no banco de dados.

A figura abaixo apresenta um diagrama de como o gerenciador de transações pode ser implementado em um SGBD



As transações são atendidas pelo gerenciador que se apoio no escalonador para definir a forma que as transações serão executadas, garantindo o isolamento. Em seguida, o gerenciador de dados formado pelo gerenciador de recuperação e o gerenciador de *buffer*, garantem a propriedade de atomicidade e durabilidade. A consistência é garantida pelo próprio SGBD através do conjunto de restrições de integridade definido.

Definições

Consideramos apenas duas operações de acesso ao banco de dados: Read(X), que lê o item X do banco de dados (vamos abstrair de qual tabela e o tipo do dado) e armazena na variável X, e Write(X) que escreve o conteúdo da variável X no item X de uma tabela qualquer do banco de dados.

Por exemplo, dada uma transação T_1 que transfere 50 reais da conta A para B, a mesma seria implementada (na nossa notação) da seguinte forma:

Ex: T1 transfere 50 reais da conta A para a conta B:

T_1:
1. read(A)
2. A=A-50
3. Write(A)
4. Read(B)
5. B=B+50
6. Write(B)

Como as propriedades ACID são aplicadas:

Atomicidade: Se a transação falhar após o passo 3 e antes do passo 6, gerenciador de transações (GT) deve garantir que as atualizações feitas não seja refletidas no banco. Essa característica é garantida pelo gerenciador de transação.

Consistência: após a transação finalizar, o item B deve ter 50 a mais e o item A, 50 a menos, conforme definido em T_1, além de a soma de A+B após o término de T_1 ser igual à antes do início de T_1. Na maioria das vezes a consistência é garantida pelo programador da aplicação, pois se ele fizer A-30 e B+40, a consistência desses dois itens não pode ser garantida.

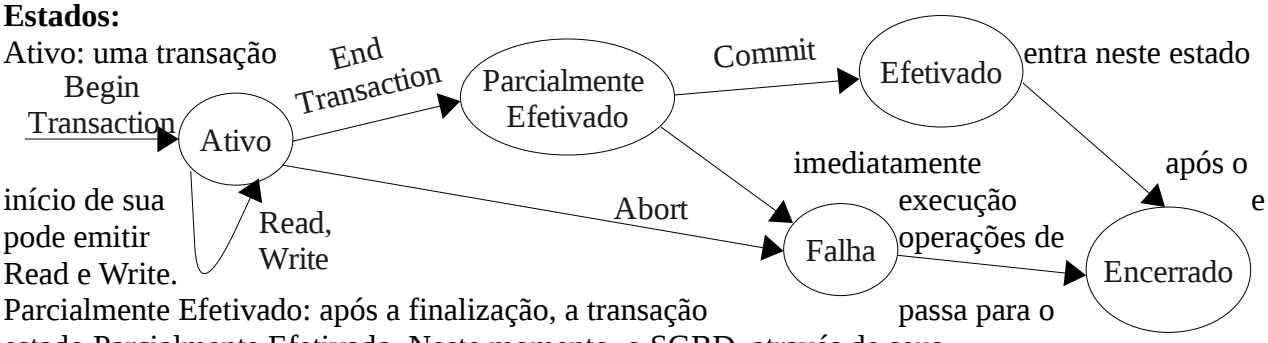
Isolamento: Se entre os passos 3 e 6, outra transação T_2 receber permissão de acessar o banco de dados parcialmente atualizado, ele verá um banco de dados inconsistente (a soma A + B será menor

do que deveria ser). O isolamento poderia ser assegurado executando as transações serialmente. Porém essa características subutilizaria o poder de processamento da máquina hospedeira. Assim, executar múltiplas transações simultaneamente oferece vantagens significativas (a forma de orquestrar essa execução será vista mais tarde). Garantida pelo gerenciador de transação e de bloqueios.
Durabilidade: após T_1 ser finalizada com sucesso, as alterações realizadas devem ser persistidas mesmos se ocorra uma falha no banco após o término. A durabilidade é garantida também pelo **gerenciador de recuperação**

Estados de uma transação

Em seu ciclo de vida, uma transação pode assumir alguns estados. Para definir os estados, vamos apresentar 4 novos comandos utilizados nas nossas transações:
Begin_Transaction: marca o início de uma transação.
End_Transaction: especifica que as operações de leitura e escrita, além dos cálculos intermediários, terminaram. Marca também o fim da execução da transação. Entretanto, neste ponto é necessário verificar se as mudanças introduzidas pela transação podem ser persistidas (efetivadas) ou se a transação deve ser abortada.
Commit_Transaction: indica o término com sucesso da transação, de forma que quaisquer alterações executadas podem ser efetivadas no banco de dados.
Rollback: indica que a transação não terminou com sucesso, de forma que as alterações realizadas pela mesma não podem ser efetivadas no banco de dados.
A figura abaixo apresenta os comandos em uma transação e os estados que a mesma atinge após a execução de cada um:

Estados:
Ativo: uma transação



Parcialmente Efetivado: após a finalização, a transação estado Parcialmente Efetivado. Neste momento, o SGBD, através de seus gerenciadores, deve garantir que uma falha no sistema não impossibilite a gravação permanente das alterações realizadas pela transação. Se tudo estiver OK, a transação entra no estado Efetivado através de um Commit.
Efetivado: Uma vez neste estado, todas as alterações realizadas pela transação são garantidas pelo SGBD (durabilidade).
Falha: o estado Falha é alcançado se alguma exceção ocorrer enquanto a mesma está no estado Ativo (bloqueios, etc.) ou se uma das verificações quando ela se encontra no estado Parcialmente Efetivado falhar.
Encerrado: este estado é alcançado quando a transação deixa o sistema, seja com sucesso ou não.

Se as transações ocorressem de forma sequencial, a propriedade ACID seria facilmente garantida. Porém, para uma melhor utilização dos recursos de uma máquina, as transações executam simultaneamente e essas execuções devem respeitar a propriedade Isolamento das transações.

Execuções Concorrentes

Diversas complicações em relação à consistência de dados, porém temos boas razões para permitir concorrência:

- Uma transação possui diversos passos (conjuntos de Reads, Writes e processamentos)
- Instruções de entrada/saída (E/S) e de processador podem operar em paralelo
    - Uma transação faz I/O, outra é processada pela CPU etc.
    - Processador e disco ficam menos tempo inativos: mais transações por tempo
    - Reduz o tempo médio de resposta: várias transações podem ser executadas concorrentemente
- Concorrência versus consistência
    - O sistema de banco de dados deve controlar a interação entre transações concorrentes para garantir a consistência dos dados
    - Identificar quais ordens de execução (escalas de execução) podem garantir a manutenção da consistência
        - esquemas de controle de concorrência (visto mais tarde)

Exemplo de execução de duas transações T_1 e T_2. T_1 transfere 50 dinheiros da conta A para a conta B, e T_2 transfere 10% do saldo da conta A para a conta B. Suponhamos que a conta A tenha 1.000 dinheiros e a conta B, 850 dinheiros. Antes T_1 de ser executada A+B=1.850, portanto, após a execução de T_1 a soma A+B deve continuar 1.850 (consistência). O mesmo caso vale para a transação T_2. Após a execução de T_1 e T_2, a soma A+B deve continuar igual antes e depois.

A tabela abaixo apresenta um escalonamento S_1 das transações que preserva a consistência.

tempo	**T_1**	**T_2**
0	Read(A)	
1	A=A-50	
2	Write(A)	
3	Read(B)	
4	B=B+50	
5	Write(B)	
6		Read(A)
7		temp=A*0.1
8		A=A-temp
9		Write(A)
10		Read(B)
11		B=B+temp
12		Write(B)

A tabela abaixo também apresenta outro escalonamento S_2 que respeita a consistência.

tempo	**T_1**	**T_2**
0		Read(A)
1		temp=A*0.1
2		A=A-temp
3		Write(A)
4		Read(B)
5		B=B+temp
6		Write(B)
7	Read(A)	
8	A=A-50	
9	Write(A)	
10	Read(B)	
11	B=B+50	
12	Write(B)	

Perceba que em ambos os escalonamentos (S_1 e S_2), T_1 e T_2 não são executadas de forma concorrente.
Em qualquer caso, a consistência é mantida, ou seja, saldo de A+B é o mesmo, antes e depois da execução das transações. S_1 e S_2 são sequenciais: a sequencia de comandos de cada transação e executada de cada vez. Assim, para um conjunto de *n* transações, há *n!* escalonamentos sequenciais válidos.
Quando várias transações são executadas simultaneamente, o escalonamento correspondente já pode não ser sequencial. Neste caso, o sistema operacional pode executar uma transação durante algum tempo, depois mudar o contexto e passar a executar a segunda transação durante algum tempo e então voltar à primeira transação durante algum tempo e assim por diante. Neste caso, várias sequências de execução são possíveis. Geralmente não é possível prever exatamente quantas instruções de uma transação serão executadas antes que a CPU alterne para outra transação.
O escalonamento S_3 apresentado na tabela abaixo é concorrente e consistente, equivale ao S_1.

tempo	**T_1**	**T_2**
0	Read(A)	
1	A=A-50	
2	Write(A)	
3		Read(A)
4		temp=A*0.1
5		A=A-temp
6		Write(A)
7	Read(B)	
8	B=B+50	
9	Write(B)	

tempo	T_1	T_2
10		Read(B)
11		B=B+temp
12		Write(B)

Já o escalonamento S_4 apresentado na tabela abaixo não é consistente.

tempo	T_1	T_2
0	Read(A)	
1	A=A-50	
2		Read(A)
3		temp=A*0.1
4		A=A-temp
5		Write(A)
6		Read(B)
7	Write(A)	
8	Read(B)	
9	B=B+50	
10	Write(B)	
11		B=B+temp
12		Write(B)

Por que S_4 não é consistente?

Se o controle da execução concorrente é deixado completamente sob a responsabilidade do SO, muitos escalonamentos de execução possíveis podem ser escolhidos, inclusive aqueles que deixam a base de dados em um estado inconsistente. É tarefa do sistema de BD garantir que qualquer escala executada deixe o BD num estado consistente: componente de controle de concorrência. Qualquer escala executada deve ter o mesmo efeito de outra que tivesse sido executada sem concorrência: um escalonamento concorrente deve ser equivalente a um escala sequencial.

O controle de concorrência é necessário pois se as transações fossem executadas apenas em série, teríamos um BD consistente, na execução concorrente, as operações das transações se entrelaçam e se ambas atualizaram o mesmo dado pode haver inconsistência.

Assim, o controle de concorrência de um SGBD evita os problemas apresentados no início do texto:

	Atualização Perdida		Leitura Suja		Sumarização Incorreta	
	T_1	T_2	T_1	T_2	T_1	T_2
1.	Read(A)	Read(A)	Read(A)		Read(A,B,C)	Read(B)
2.	A=A+10		A=A+10		T=A+B+C	B=B+5
3.	Write(A)	A=A+20	Write(A)			Write(B)
4.		Write(A)		Read(A)	Write(T)	
5.				A=A+20		
6.				Commit		
7.	Abort					

Planos de Execução (Escalonamento)

Quando transações são executadas concorrentemente, de maneira intercalada, a ordem de execução das operações das várias transações é conhecida como plano de execução.

Um plano *S* de *n* transações T_1, T_2, …, T_n é a ordenação das operações das transações sujeitas a uma restrição tal que, para cada transação T_i em *S*, as operações de T_i deverão aparecer na mesma ordem em que elas foram definidas em T_i. As operações das outras transações T_j em *S* poderão ser intercaladas com as operações de T_i .

Vamos escrever w para Write, r para Read, c para Commit e a para Abort. Também cada operação possuirá um índice que corresponderá ao número da transação que ela pertence. Assim, abaixo temos um plano S_a.

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

Ou seja, T_1 lê o item X, em seguida T_2 também lê o item X, T_1 grava X (possivelmente alterado), T_1 lê o item Y, T_2 também grava X (possivelmente alterado), e finalmente T_1 grava Y (possivelmente alterado).

Outro plano S_b:

S_b:$r_1(X);w_1(X);r_2(X);w_2(X);a_1$

Duas operações em um plano são ditas em conflito se:

1. pertencem a diferentes transações
2. Acessem o mesmo item X
3. Pelo menos uma das operações ser um Write (w).

Por exemplo, em S_a, as operações $r_1(X)$ e $w_2(X)$conflitam, bem como as operações $r_2(X)$ e $w_1(X)$,e as operações $w_1(X)$ e $w_2(X)$. As operações $r_1(X)$ e $r_2(X)$ não conflitam pois são ambas de leitura e $w_2(X)$ e $w_1(Y)$ também pois são sobre itens diferentes. Finalmente, $r_1(X)$ e $w_1(X)$ não conflitam pois pertencem a mesma transação (T_1).

Um plano *S* de *n* transações T_1, T_2, …, T_n é dito completo se as seguintes condições forem garantidas:

1. As operações em S são exatamente as operações T_1, T_2, …, T_n tendo um Commit ou Abort como última operação de cada transação.
2. Para quaisquer pares de operações de T_i, a ordem dessas operações em *S* deve ser a mesma definida em T_i.
3. Para quaisquer duas operações conflitantes, um das duas precisa aparecer antes da outra no plano (a ordem não é determinada).