7 Controle de Transações: a Missão!

Imagine a seguinte situação: *Você vai até o caixa eletrônico de seu banco, solicita que seja retirado um determinado valor de sua conta corrente e depositado em sua conta de poupança*. De repente, não mais que de repente, tchan..tchannnn, puf o terminal apaga! Seu cabelo se arrepia! Caramba, o que aconteceu???!!! Fica pensando: *Será que o dinheiro já saiu de minha conta?? Se saiu, será que já foi registrado na poupança???* E não sossega enquanto não puder ver o que aconteceu...

Isso já aconteceu com você? Se aconteceu, qual foi o resultado? Deu tudo certo no final das contas? Espero que sim! Pois se tudo deu certo no final é porque o gerenciador de transações do sistema que controla a sua conta estava funcionando como deveria!

Bom, quer saber um pouco mais sobre como o sistema pode garantir que tudo termine bem, apesar dos problemas que possam ocorrer no meio do caminho? Então venha, acompanhe-me em mais um emocionante capítulo de nossa querida disciplina: "Controle de Transações: a Missão!"

7.1 Conceito de Transação

A operação bancária mencionada na introdução desse capítulo parece se tratar de apenas um evento: *transferência de fundos entre contas bancárias: conta corrente para conta poupança*, não é verdade? Pois não é bem assim!

Em um sistema de banco de dados, devem ocorrem diversas operações mais simples para viabilizar a macro operação de transferência de fundos. No entanto, é razoável pensar que é desejável que ocorram todas essas operações mais simples, de modo a efetivar a transferência, ou em caso de falha, que nenhuma operação mais simples seja efetivada! Você percebe isso?!

Ora, se algumas dessas operações mais simples forem efetivadas e outras não, você corre o risco de ver seu rico dinheirinho desaparecer! Já pensou?! O dinheiro sai da conta corrente mas não chega na conta poupança! Pode isso??!!! Ah, eu não sei vocês, mas pra mim com certeza não pode!!!!

Pois é, o nome que damos para esse conjunto de operações mais simples que devem ser todas realizadas ou todas descartadas, nós chamamos de **Transações**!

Então o primeiro conceito que você deve conhecer é esse:

Uma **Transação** é uma unidade de execução do programa que acessa e possivelmente atualiza vários itens de dados.

Normalmente, essas operações estarão delimitadas em seu código pelos comandos **Begin Transaction** e **End Transaction**.

Para garantir a integridade dos dados é necessário que o sistema de banco de dados mantenha as seguintes propriedades das transações, que é comumente referida como **ACID** (não é o que você está pensando!):

Atomicidade: todas as operações da transação são refletidas corretamente no banco de dados, ou nenhuma delas;

Consistência: a execução de uma transação isolada (ou seja, sem qualquer outra transação executando simultaneamente) preserva a consistência do banco de dados.

Isolamento: embora várias transações possam ser executadas simultaneamente, o sistema garante que, para cada par de transações T_i e T_j, parece para T_i que ou T_j terminou a execução antes que T_i começasse ou T_j iniciou a execução depois que T_i terminou. Assim, cada transação não está ciente das outras transações executadas simultaneamente no sistema.

Durabilidade: Depois que uma transação for completada com sucesso, as mudanças que ela fez ao banco de dados persistem, mesmo que existam falhas no sistema.

7.2 Exemplo: Consistência

Imagine que um sistema bancário muito simples que consiste de duas operações básicas:

read(x) que transfere o item x do BD para um buffer local pertencente à transação que executou a operação read;

write(x) que transfere o item x do buffer local da transação que executou a operação write de volta para o BD;

Suponha ainda, que T_i\$ seja uma transação que transfere R$ 50,00 da conta A para a conta B. Essa transação pode ser como:

T_i: read(A);

A:= A - 50;

write(A);

read(B);

B:=B+50;

write(B).

O requisito de **consistência** é que a soma de **A+B** sejam inalteradas pela execução da transação. Traduzindo, se não fosse por essa propriedade, o dinheiro poderia ser criado ou destruído pela transação!

Então supondo que os valores iniciais de A e B, sejam respectivamente 100 e 50. Portanto, antes de iniciar a transação o valor de A+B era de 150. A transação após terminada deve manter esse mesmo valor para A+B. Acompanhe:

T_i: A+B = 150

read(A);	100
A:= A - 50;	A := 100 - 50
write(A);	A:=50
read(B);	50
B:=B+50;	B := 50 + 50
write(B).	B:= 100

A+B = 150 (mantido o mesmo valor antes da execução da transação)

A tarefa de manutenção de consistência cabe ao programador da aplicação garantir!

7.3 Exemplo: Atomicidade

Agora suponha que, durante a execução da transação T_i, aconteça uma falha (falta de energia elétrica, falhas de hardware ou erros de software) que impede T_i de completar sua execução com sucesso.

Suponha ainda que a falha ocorreu após a operação write(A), mas antes da operação write (B). Vejamos o que ocorreu:

read(A);	100
A:= A - 50;	A := 100 - 50
write(A);	A:=50
read(B);	50
B:=B+50;	B := 50 + 50 //logo após ocorre falha!
write(B).	50 // O novo valor é perdido!!!

Resultado: sumiram os R$ 50,00 que foram retirados de A (valor atual de 50) mas que não foram computados em B (valor atual igual ao inicial, ou seja, manteve os 50). Ou seja, a soma A+B não é mais preservada!

Ou seja, o estado do banco de dados encontra-se **inconsistente**!

Se a propriedade de atomicidade estiver presente, todas as ações da transação são refletidas no banco de dados ou nenhuma delas é refletida. É 8 ou 80, ou consegue realizar a transação como um todo ou desfaz o que estava feito até o momento da falha!

O sistema de banco de dados deve acompanhar os valores antigos de quaisquer dados em que uma tranasação realiza uma escrita e, se a transação não completar com sucesso sua execução, então o sistema restaura os valores antigos para que pareça que a transação nunca sequer começou.

Porisso, o problema mencionado no inicio desse capítulo não deve gerar problemas para você! O sistema de banco de dados deve garantir que, independente de qual foi o momento em que ocorreu a falha dentro da transação, os seus dados vão ficar restaurados ao momento anterior ao do inicio da transação. Sendo assim, seu dinheiro não sairá da conta corrente.

7.4 Exemplo: Durabilidade

Quando a execução da transação termina com sucesso, e o usuário que iniciou a transação foi notificado de que a transferência de fundos ocorreu, é preciso que nenhuma falha no sistema resulte em uma perda dos dados correspondentes a essa transferência.

A propriedade de durabilidade garante que, quando uma transação é executada com sucesso, todas as atualizações que ela executou no banco de dados persistem, mesmo que haja uma falha no sistema após a transação terminar sua execução.

Garantir a durabilidade é tarefa do sistema de banco de dados.

7.5 Exemplo: Isolamento

Mesmo que as propriedades de consistência e atomicidade sejam garantidas em cadas transação, se várias transações forem executadas simultaneamente, suas operações podem intercalar de alguma maneira indesejável, resultando em um estado inconsistente.

Suponha que além de T_1 que já havia sido mencionado tenhamos também T_2 que transfere 10% do saldo da conta A para a conta B.

Se executarmos T_1 e depois executarmos T_2, teremos:

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)

A sequencia em que são realizadas as operações no tempo é chamado de **schedule**. Na figura anterior temos um exemplo de schedule serial, ou seja, uma transação é executada após o término da execução da outra. Dessa forma, sempre se garante a integridade do banco, pois se efetuarmos a soma de A+B podem notar que o valor é preservado.

Mas, e se o sistema executa várias transações simultaneamente, o schedule pode ser simultâneo. Ou seja, transações iniciarão antes do término da outra. Imagine o seguinte schedule simultâneo que é equivalente ao schedule serial apresentado acima. Esse schedule é dito equivalente pois reproduz exatamente o mesmo resultado que o anterior. Portanto, o estado do banco de dados é dito estar consistente.

T_1	T_2
read(A);	
A:= A - 50;	
write(A);	

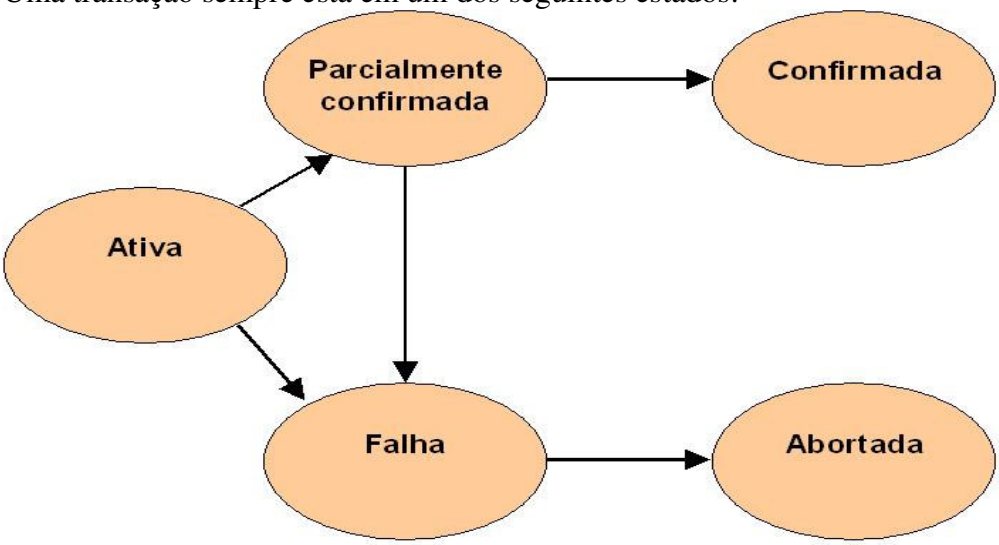
	read(A);
	temp:= A * 0,1
	A:= A - temp
	write(A)
read(B);	
B:=B+50;	
write(B)	
	read(B)
	B:= B + temp
	write (B)

Porém, o schedule simultâneo pode levar a resultados indesejável, como no exemplo seguinte, onde a soma de A+B não é preservada pela execução das 2 transações:

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

7.6 Estado de Transação

Uma transação sempre está em um dos seguintes estados:



- **Ativa,** é o estado inicial; a transação permanece nesse estado enquanto está sendo executada;
- **Parcialmente confirmada,** depois que a instrução final foi executada;
- **Falha,** depois da descoberta de que a execução normal não pode mais prosseguir;
- **Abortada,** depois que a transação foi revertida e o BD foi restaurado ao seu estado anterior ao início da transação;
- **Confirmada,** após o término bem sucedido.

Sempre que uma transação estiver no estado **abortada,** o sistema de banco de dados para garantir a atomicidade deve desfazer qualquer mudança que a transação tiver realizado até o momento da falha. É dito que a transação foi **revertida (rolled back).**

Uma transação que completa sua execução com sucesso é considerada **confirmada (committed).**

8 Bibliografia Recomendada

[ELMASRI 2005] Elmasri, R. & Navathe, S. **Sistemas de Banco de Dados**, 4a. edição, Editora Pearson/Addison Wesley, 2005

[HEUSER 2009] Heuser, C.A. **Projeto de Banco de Dados**, 6a. edição, Editora Bookman, 2009

[PRESSMAN 2006] Pressman, R.S. **Engenharia de Software**, 8a. edição, Editora MacGraw-Hill, 2006

[RAMAKRISHNAN 2002] Ramakrishnan, R. & Gehrke, J. **Database Management Systems**, Third Edition, 2002

[SILBERCHATZ 2006] Silberchatz, A.; Korth, H.F. & Sudarshan, S. **Sistema de Banco de Dados**, 5a. edição, Editora Campus, 2006