Resumo 4 Banco de Dados 1

Marcos Geraldo Braga Emiliano 19.1.4012

Fonte: Slides das aulas

-Normalização para Bancos de Dados Relacionais

Introdução

De acordo com o modelo relacional: o esquema de um banco de dados relacional consiste em um número de esquemas de relação, cada esquema de relação consiste de um número de atributos e os atributos são agrupados para formar um esquema de relação, utilizando-se o bom senso do projetista de banco de dados ou mapeando-se um esquema ERE para um modelo relacional.

Entretanto, precisa-se de algum método de avaliação formal da qualidade de um esquema de relação, a normalização surgiu como um processo para se escolher "bons" esquemas de relação, ou seja, para avaliar formalmente porque um conjunto de atributos para um esquema de relação é melhor do que um outro.

Informalmente, existem normas referentes a 04 quesitos a serem avaliados quanto à qualidade dos esquemas de relação: semântica dos atributos, valores redundantes em tuplas, valores nulos em tuplas e possibilidade de geração de tuplas espúrias.

-**Semântica dos Atributos de Relações**

Sempre que se agrupa atributos para formar um esquema de relação, parte-se do pressuposto de que um certo significado está associado aos atributos, esse significado (semântica) especifica como interpretar os valores dos atributos armazenados em uma tupla da relação. Por exemplo, no esquema da relação

Empregado (NomeE, CodEmp, DtNasc, Endereco, NumDept),

cada tupla representa um empregado, com valores para nome, código, data de nascimento, endereço e número do departamento no qual o empregado trabalha.

Norma informal 1:

Projete um esquema de relação de modo que seja fácil explicar seu significado. Não combine atributos de vários tipos de entidades e relacionamentos em uma única relação, assim, a relação não será confusa em termos semânticos.

-Valores Redundantes em Tuplas

Ao se projetar esquemas de relação, deve-se minimizar o espaço de armazenamento que as relações ocupam, valores redundantes em tuplas aumentam o espaço de armazenamento das relações e valores redundantes

podem gerar anomalias de atualização, classificadas como anomalias de inclusão, de exclusão e de modificação.

Por exemplo, considere o esquema:

Emp_Depto (NomeE, CodEmp, DtNasc, Endereco, NumDept, NomeD, CodGerente)

Em Emp_Depto, os valores dos atributos referentes a departamento são repetidos para todos os empregados que trabalham no mesmo departamento.

Norma informal 2:

Projete os esquemas de relação de modo que nenhuma anomalia de inclusão, exclusão ou modificação esteja presente nas relações. Se estiverem presentes quaisquer anomalias, mencione-as com clareza e certifique-se de que os programas que atualizam o banco de dados irão operar corretamente.

Em determinadas situações, a norma precisa ser violada a fim de melhorar o desempenho de algumas consultas. Por exemplo, se uma importante consulta deve recuperar dados dos departamentos de empregados, juntamente com os dados dos empregados, pode-se utilizar o esquema de relação Emp_Depto.

-Valores Nulos em Tuplas

Se muitos atributos de um esquema de relação não se aplicarem a todas as tuplas, tal relação possuirá vários valores nulos, podendo gerar: desperdício de espaço de armazenamento, problemas na aplicação de operações de JOIN e de funções de agregação como COUNT e diferentes interpretações do valor nulo, o atributo não se aplica à tupla, o valor do atributo para esta tupla é desconhecido e o valor é conhecido, porém ausente (não foi registrado).

Norma informal 3:

Tanto quanto possível, evite colocar atributos em uma relação cujos valores possam frequentemente ser nulos. Se nulos forem inevitáveis, assegure-se de que eles se aplicam em casos excepcionais e não se aplicam à maioria das tuplas na relação.

-Possibilidade de Geração de Tuplas Espúrias

Por exemplo, considere os esquemas:

Emp_Locs (NomeE, LocalizacaoP)

Emp_Proj2 (CodEmp, CodProj, Horas, NomeP, LocalizacaoP)

Uma tupla em Emp_Locs significa que o empregado de nome "NomeE" trabalha em um projeto localizado em "LocalizacaoP". Uma tupla em Emp_Proj2 significa que o empregado de código "CodEmp" trabalha "Horas" por semana no projeto de nome "NomeP" e código "CodProj" localizado em "LocalizacaoP".

É um projeto ruim, pois se for executada uma junção natural entre as relações, o resultado produzirá, além das tuplas corretas, um grande número de tuplas espúrias (inválidas), isto ocorre pois o atributo que relaciona Emp_Locs e Emp_Proj2 ("LocalizacaoP") não corresponde a uma chave primária nem a uma chave estrangeira.

Norma informal 4:

Projete esquemas de relação de modo que possam ser juntados (JOIN) com condições de igualdade em atributos que sejam chave primária/chave estrangeira, de tal forma que garanta que nenhuma tupla espúria seja gerada. Não conserve relações que possuam atributos iguais, além das combinações de chave primária/chave estrangeira; se tais relações forem inevitáveis, não junte as mesmas nesses atributos.

-Dependência Funcional

Uma dependência funcional entre dois conjuntos de atributos (X e Y) de uma relação, cujo esquema é R, é uma restrição que estabelece que, para quaisquer tuplas t1 e t2 de R, se t1[X] = t2[X], então t1[Y] = t2[Y]. Isso significa que os valores de Y dependem ou são determinados univocamente pelos valores de X nas tuplas de R, ou seja, Y é funcionalmente dependente de X (X → Y), se X for uma chave candidata de R, então X → Y para qualquer subconjunto de atributos Y de R, X → Y em R não implica que Y → X.

Considere os seguintes esquemas:

Emp_Depto (NomeE, CodEmp, DtNasc, Endereco, NumDept, NomeD, CodGerente)

Emp_Proj (CodEmp, CodProj, Horas, NomeE, NomeP, LocalizacaoP)

A partir da semântica dos atributos, existem as seguintes dependências funcionais:

CodEmp → {NomeE, DtNasc, Endereço, NumDept}

NumDept → {NomeD, CodGerente}

{CodEmp, CodProj} → Horas

CodEmp → NomeE

CodProj → {NomeP, LocalizacaoP}

-Normalização

A normalização de dados pode ser vista como um processo para analisar esquemas de relação com base em suas dependências funcionais e chaves primárias, no intuito de diminuir redundância. O processo de normalização fornece uma série de testes de forma normal que podem ser realizados em esquemas de relação individuais, permitindo a normalização de esquemas relacionais no grau desejado. A forma normal de uma relação refere-se à condição da forma normal mais elevada que ela atinge e, portanto, indica o

grau para o qual ela foi normalizada, é desejável que todas as relações de um esquema relacional se encontrem na 3ª forma normal.

-Primeira Forma Normal

Uma relação está na primeira forma normal (1FN) se o domínio de todos os seus atributos possuir apenas valores atômicos, e o valor de qualquer atributo em suas tuplas ser um único valor do domínio desse atributo, informalmente, para que uma relação esteja na 1FN, seus atributos não podem ser multivalorados, compostos ou complexos. Considere o seguinte esquema de relação:

Departamento (NomeD, NumDept, CodGerente, LocaisD)
Existem duas maneiras de se enxergar o atributo "LocaisD": o domínio de "LocaisD" permite conjuntos de valores e, portanto, não é atômico. Neste caso, "LocaisD" é funcionalmente dependente de "NumDept", porque cada conjunto é considerado único do domínio do atributo, o domínio de "LocaisD" permite apenas valores atômicos, mas algumas tuplas possuem o conjunto desses valores (um valor por tupla), neste caso, "LocaisD" não é funcionalmente dependente de "NumDept".

Soluções para se alcançar a 1FN são: remover o atributo "LocaisD", colocando-o em uma nova relação juntamente com a chave primária de Departamento, a chave primária da nova relação é {NumDept, LocaisD}, desvantagem: necessidade de junção na pesquisa. Se um número máximo de valores para o atributo "LocaisD" for conhecido (por exemplo 3), substituir o mesmo por três novos atributos atômicos na relação Departamento, desvantagem: introdução de valores nulos na relação. Expandir a chave primária de Departamento para {NumDept, LocaisD}, de tal forma que haja uma tupla em separado para cada localização de um departamento, desvantagem: introdução de redundância na relação.

-Segunda Forma Normal

Uma relação está na segunda forma normal (2FN) se estiver na 1FN e se todo atributo não chave possuir dependência funcional total em relação à chave primária da relação. $X \rightarrow Y$ é uma dependência funcional total se a remoção de qualquer atributo A de X significar que a dependência não mais se mantém; caso contrário, é uma dependência funcional parcial. Informalmente, para que uma relação esteja na 2FN, todo atributo não chave necessita de toda a chave primária para ser identificado. Considere o seguinte esquema de relação:

Emp_Proj (CodEmp, CodProj, Horas, NomeE, NomeP, LocalizacaoP)

Para tal esquema, existem as dependências funcionais:

{CodEmp, CodProj} → Horas

CodEmp → NomeE

CodProj → {NomeP, LocalizacaoP}

Tal esquema está na 1FN mas não está na 2FN. Os atributos não chaves "NomeE", "NomeP" e "LocalizaçãoP" possuem dependência funcional parcial em relação à chave primária da relação. Para colocar a relação na 2FN, cada dependência funcional parcial dará origem a uma nova relação, em substituição à relação Emp_Proj:

Emp_Proj1 (CodEmp, CodProj, Horas)

Emp_Proj2 (CodEmp, NomeE)

Emp_Proj3 (CodProj, NomeP, LocalizacaoP)

-Terceira Forma Normal

Uma relação está na terceira forma normal (3FN) se estiver na 2FN e se todo atributo não chave não possuir dependência funcional transitiva em relação à chave primária da relação. X → Y é uma dependência funcional transitiva na relação R se existir um conjunto de atributos Z em R, que não seja chave candidata de R, de tal forma que as dependências X → Z e Z → Y existem.

Informalmente, para que uma relação esteja na 3FN, não pode existir uma dependência funcional indireta entre um determinado atributo não chave com a chave primária da relação, por meio de um conjunto de atributos não chaves. Considere o seguinte esquema de relação:

Emp_Depto (NomeE, CodEmp, DtNasc, Endereco, NumDept, NomeD, CodGerente)

Para tal esquema, existem as dependências funcionais:

CodEmp → {NomeE, DtNasc, Endereco, NumDept}

NumDept → {NomeD, CodGerente}

Tal esquema está na 2FN mas não está na 3FN. Existe uma dependência transitiva dos atributos não chaves "NomeD" e "CodGerente" em relação à chave primária "CodEmp", por meio do atributo não chave "NumDept".

Para colocar a relação na 3FN, em substituição à relação Emp_Depto, a dependência funcional transitiva dará origem a duas novas relações, com uma ligação de chave primária/chave estrangeira entre elas:

Emp_Depto1 (NomeE, CodEmp, DtNasc,Endereco, NumDept)

Emp_Depto2 (NumDept, NomeD, CodGerente)

-Noções de Processamento de Transações, Controle de Concorrência e Recuperação de Falhas

-Transações

Transação é uma unidade lógica de processamento de operações sobre um banco de dados, uma transação é formada por uma sequência de operações que precisam ser executadas integralmente para garantir a consistência e a

precisão.

Geralmente, uma transação consiste de uma das seguintes instruções: uma ou mais instruções DML (Data Manipulation Language), uma instrução DDL (Data Definition Language) e uma instrução DCL (Data Control Language).

Uma transação começa quando for executada a 1ª instrução SQL executável e termina com um dos seguintes eventos: comando COMMIT ou ROLLBACK é emitido, instrução DDL ou DCL é executada (commit automático), o usuário desconecta do banco de dados (commit automático) e o sistema falha (rollback automático).

Quando uma transação termina, o próximo comando SQL inicia automaticamente a próxima transação. As instruções de controle de transações são: **COMMIT**: finaliza a transação atual tornando permanentes todas as alterações de dados pendentes, **SAVEPOINT** <nome_savepoint>: marca um ponto de gravação dentro da transação atual, sendo utilizado para dividir uma transação em partes menores, **ROLLBACK [TO SAVEPOINT <nome_savepoint>]**: **ROLLBACK** finaliza a transação atual, descartando todas as alterações de dados pendentes. **ROLLBACK TO SAVEPOINT** descarta o ponto de gravação determinado e as alterações seguintes ao mesmo. Exemplo:

INSERT INTO Departamento (ID_Depto, NomeDepto, ID_Gerente) **VALUES** (10, 'Marketing', 3);

UPDATE Funcionario **SET** salario = salario * 1.05

WHERE ID_Depto = 5;

COMMIT;

DELETE FROM Funcionario;

ROLLBACK;

No exemplo, o departamento 10 é inserido, os salários dos funcionários do departamento 5 são atualizados, mas nenhum funcionário é excluído.

-Processamento de Transações

Quando várias transações são emitidas ao mesmo tempo (transações concorrentes), ocorre um entrelaçamento de operações das mesmas. Algumas operações de uma transação são executadas, em seguida, seu processo é suspenso e algumas operações de outra transação são executadas, depois, um processo suspenso é retomado a partir do ponto de interrupção, executado e interrompido novamente para a execução de uma outra transação.

As propriedades de uma transação (propriedades ACID) devem ser garantidas ao processar transações concorrentes: atomicidade, uma transação é uma unidade atômica de processamento, é realizada integralmente ou não é realizada, consistência, uma transação é consistente se levar o banco de

dados de um estado consistente para outro estado também consistente, isolamento, a execução de uma transação não deve sofrer interferência de quaisquer outras transações que estejam sendo executadas concorrentemente, durabilidade (ou persistência), as alterações aplicadas ao banco de dados, por meio de uma transação confirmada, devem persistir no banco de dados, não sendo perdidas por nenhuma falha.

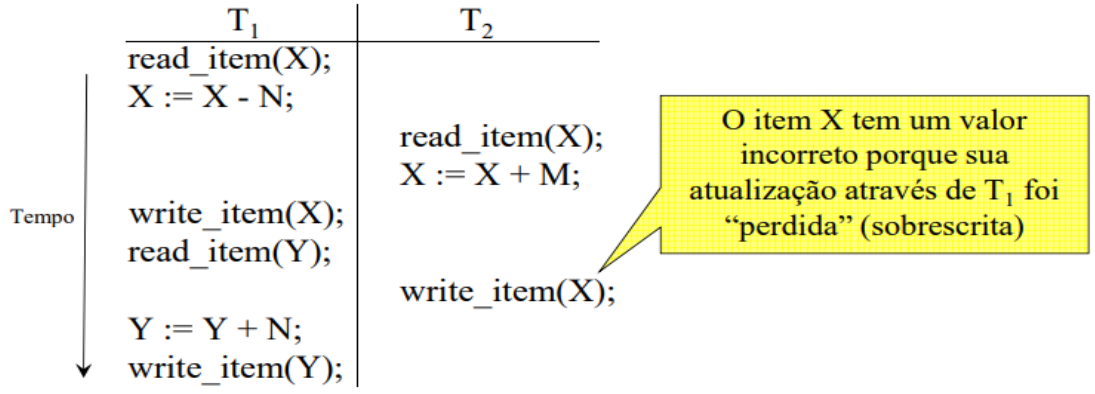
O modelo simplificado para processamento de transações concorrentes envolve as seguintes operações: read_item(X), lê um item X do banco de dados e transfere para uma variável X de memória e write_item(X), escreve o valor de uma variável X de memória em um item X do banco de dados. Exemplo:

T_1	T_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);	

-Controle de Concorrência

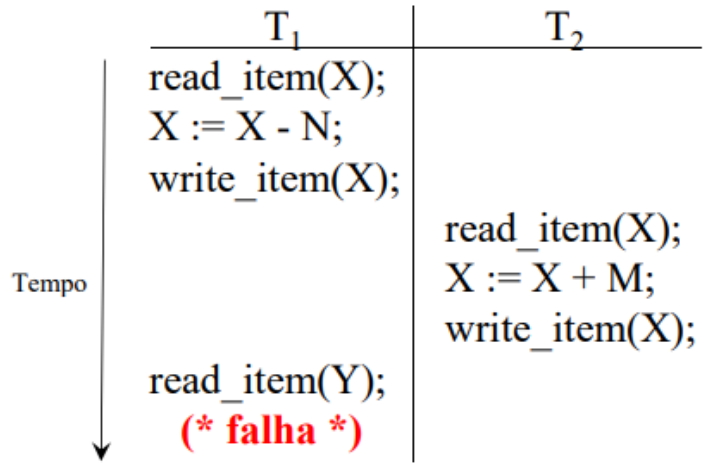
Técnicas de controle de concorrência são utilizadas para garantir que transações concorrentes sejam executadas adequadamente. Muitos problemas podem ocorrer quando transações concorrentes são executadas sem controle, a saber, problema da perda de atualização, problema da atualização temporária (leitura suja), problema da agregação incorreta e problema da leitura não-repetitiva.

Problema da perda de atualização, ocorre quando duas transações que acessam os mesmos itens do banco de dados possuem operações entrelaçadas, de modo que torne incorreto o valor de algum item do banco de dados.



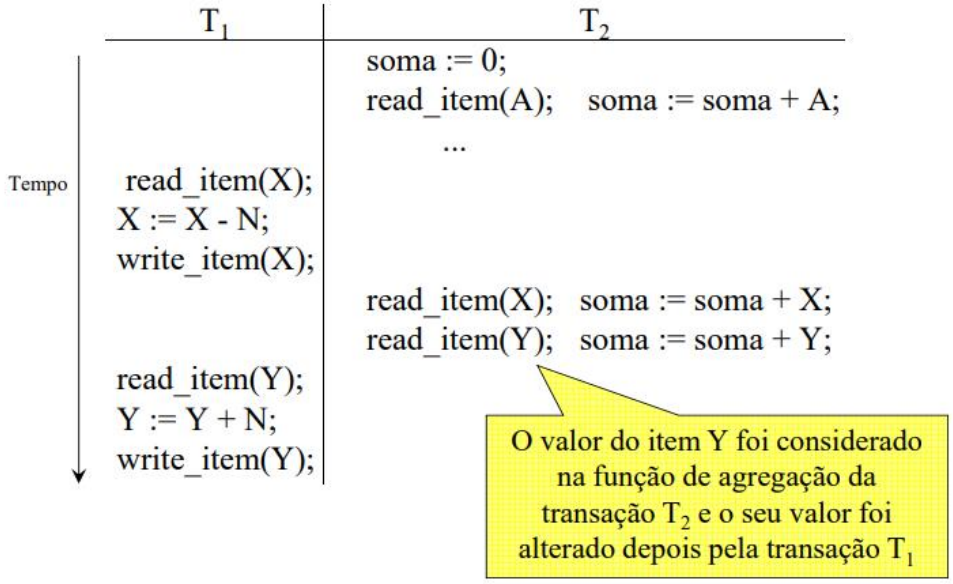
O valor final do item X em T2 estará incorreto, porque T2 lê o valor de X antes que T1 o altere no banco de dados e, portanto, o valor atualizado resultante de T1 será perdido.

Problema da atualização temporária (leitura suja), ocorre quando uma transação atualiza um item do banco de dados e, por algum motivo, a transação falha, no caso, o item atualizado é acessado por uma outra transação antes do seu valor ser retornado ao valor anterior.

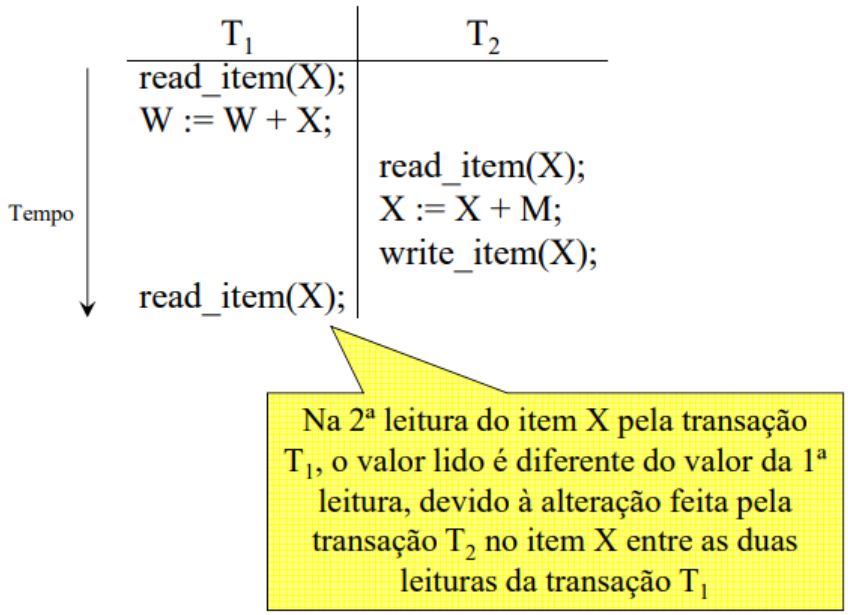


A transação T1 falha e o sistema deve alterar o valor do item X para o seu valor anterior; porém, T2 já leu o valor temporário "incorreto" do item X.

Problema da agregação incorreta, se uma transação estiver calculando uma função de agregação em um número de itens, enquanto outras transações estiverem atualizando alguns desses itens, a função agregada pode considerar alguns itens antes que eles sejam atualizados e outros depois que tenham sido atualizados.



Problema da leitura não-repetitiva, ocorre quando uma transação T lê um item duas vezes e o item é alterado por uma outra transação T' entre as duas leituras de T. Portanto, T recebe diferentes valores para suas duas leituras do mesmo item.



-Recuperação de Falhas

O SGBD não deve permitir que algumas operações de uma transação T sejam aplicadas ao banco de dados enquanto outras operações de T não sejam, isso pode acontecer quando uma transação falha após executar algumas de suas operações (e não todas).

Os tipos de falhas possíveis são: falha no computador, erro de transação ou de sistema imposição do controle de concorrência, falha no disco e problemas físicos e catástrofes. Para os 3 primeiros tipos de falhas, o sistema deve manter informações suficientes para se recuperar da falha.

Para manter a consistência do banco de dados, o gerenciador de recuperação registra no histórico (log), para cada transação, as operações que afetam os valores dos itens do banco: [start_transaction, T], indica que a transação T iniciou sua execução, [write_item, T, X, old_value old_value, new_value], indica que a transação T alterou o valor do item X do banco de dados de old_value (valor antigo) para new_value (novo valor), [read_item, T, X], indica que a transação T leu o valor do item X do banco de dados, [commit, T], indica que a transação T foi finalizada com sucesso e [abort, T], indica que a transação T foi abortada.

Quando ocorre uma falha: as transações inicializadas, mas que não gravaram seus registros de commit no log, devem ser desfeitas e as transações que gravaram seus registros de commit no log podem ter que ser refeitas a partir

dos registros do log.

Para tanto, no processo de recuperação de falhas relativa a uma transação T, usam-se as operações: UNDO (desfazer): desfaz a transação, ou seja, percorre o log de forma retroativa, retornando todos os itens alterados por uma operação write de T aos seus valores antigos e REDO (refazer): refaz a transação, ou seja, percorre o log para frente, ajustando todos os itens alterados por uma operação write de T para seus valores novos.

-Escalonamento e Recuperabilidade

Um escalonamento S de n transações é uma ordenação das operações dessas transações sujeita à restrição de que, para cada transação Ti que participa de S, as operações de Ti em S devem aparecer na mesma ordem em que ocorrem em Ti. Notação simplificada para escalonamento: $r_i(X)$: read_item(X) na transação T_i, $w_i(X)$: write_item(X) na transação T_i, c_i: commit na transação T_i, a_i: abort na transação T_i. Exemplos de escalonamento:

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

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

Duas operações em um escalonamento são ditas conflitantes se: pertencem a diferentes transações, possuem acesso ao mesmo item X e pelo menos uma delas é uma operação write_item(X).

Um escalonamento S é dito ser recuperável se nenhuma transação T em S entrar no estado confirmado até que todas as transações T', que tenham escrito um item que T tenha lido, entrem no estado confirmado.

Sa: r1(X); w1(X); r2(X); r1(Y); w2(X); w1(Y); c1; r2(Y); c2; S é recuperável. a

Sc: r1(X); w1(X); r2(X); r1(Y); w2(X); c2; a1; Sc não é recuperável, porque T2 lê o item X atualizado por T1, e então T2 é confirmado antes que T1 se confirme.

Em um escalonamento recuperável, pode ocorrer um fenômeno conhecido como *rollback* em cascata, no qual uma transação não-confirmada tenha que ser desfeita porque leu um item de uma transação que falhou.

Se: r1(X); w1(X); r2(X); r1(Y); w2(X); w1(Y); a1; a2;

Um escalonamento evita *rollbacks* em cascata se todas as transações no escalonamento lerem somente itens que tenham sido escritos por transações já confirmadas, no escalonamento S_e anterior, $r_2(X)$ deve ser adiada até que T_1 tenha sido confirmada (ou abortada), retardando T_2.

-Seriabilidade de Escalonamentos

Um escalonamento S é denominado serial se, para todas as transações T participantes do escalonamento, todas as operações de T forem executadas consecutivamente no escalonamento, caso contrário, o escalonamento é

denominado não-serial. Um escalonamento serial: possui somente uma transação ativa de cada vez, não permite nenhum entrelaçamento de transações, é considerado correto, independente da ordem de execução das transações, limita a concorrência e na prática, é inaceitável.

Um escalonamento S de n transações é serializável se for equivalente a algum escalonamento serial das n transações, dizer que um escalonamento não-serial S é serializável equivale a dizer que ele é correto, já que equivale a um escalonamento serial que é considerado correto, dois escalonamentos são ditos equivalentes se a ordem de quaisquer duas operações conflitantes for a mesma nos dois escalonamentos.

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

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

S_c: $r_1(X)$; $r_2(X)$; $w_1(X)$; $r_1(Y)$; $w_2(X)$; $w_1(Y)$; c_1; c_2; (não serializável)

-Teste de Seriabilidade

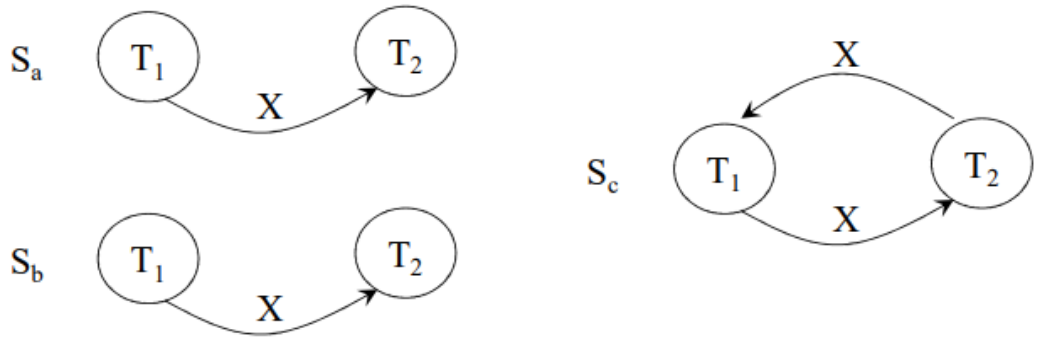
Uma forma de testar a seriabilidade de um escalonamento é através da construção de um grafo de precedência, um grafo de precedência de um escalonamento S é um grafo dirigido G = (N, E), onde N é um conjunto de nodos {T_1, T_2, ..., T_n}e E é um conjunto de arcos dirigidos {e_1, e_2, ...,e_m} tal que: cada nodo T_i corresponde a uma transação de S, cada arco e_j liga uma transição T_j que possui uma operação conflitante com uma transição T_k. Um escalonamento S é serializável se e somente se o grafo de precedência não tiver nenhum ciclo.

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

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

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

• Para os escalonamentos Sa, Sb e Sc, os grafos de precedência são:



-Técnicas de Bloqueio (Locking)

Bloqueios são usados como um meio de sincronizar o acesso aos itens do banco de dados por transações concorrentes, um bloqueio consiste em uma

variável associada a um item de dado, que descreve o status do item em relação a possíveis operações que podem ser aplicadas ao mesmo, em geral, existe um bloqueio para cada item de dado no banco de dado. Duas técnicas de bloqueio são: bloqueio binário e bloqueio múltiplo.

-Bloqueio Binário

Um bloqueio binário possui dois estados: bloqueado (locked) e desbloqueado (unlocked). As operações necessárias são: lock_item(X), bloqueia o item X e unlock_item(X), desbloqueia o item X. O bloqueio binário impõe a exclusão mútua no item de dado, se o item X for bloqueado por uma transação T_i, nenhuma outra transação T_j poderá acessar o item X até que a transação T_i o desbloqueie, ficando esperando por tal desbloqueio. O processo de espera coloca a transação T_j em uma fila de espera pelo item X até que o mesmo seja desbloqueado por T_i.

Para que a técnica de bloqueio binário possa ser usada, uma transação T deve obedecer às seguintes regras: T deve emitir um lock_item(X) antes que qualquer read_item(X) ou write_item(X) seja executado, T deve emitir um unlock_item(X) depois que todos os read_item(X) e write_item(X) tenham sido completados em T, T não poderá emitir lock_item(X) se X estiver bloqueado por T e T poderá emitir um unlock_item(X) apenas se tiver bloqueado X.

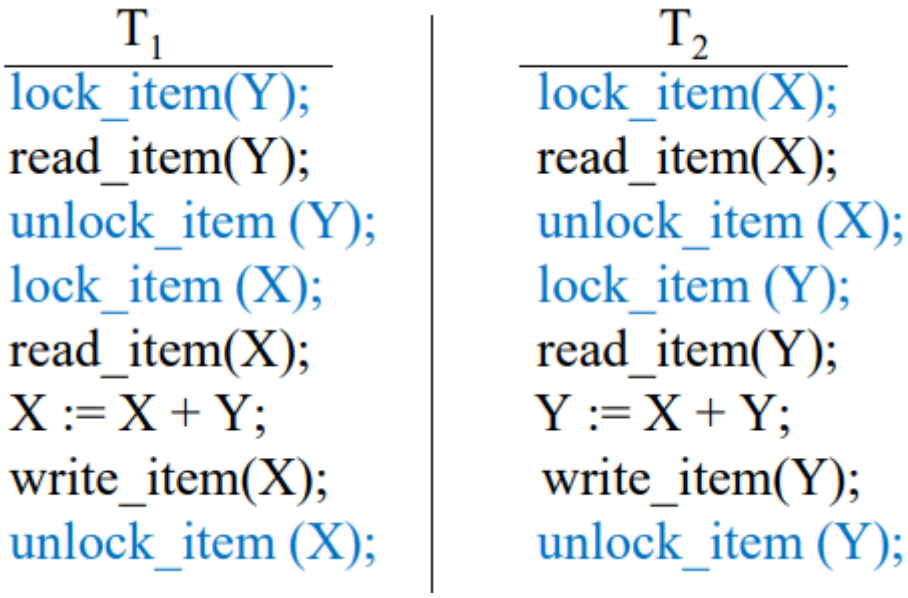
O bloqueio binário é o mecanismo mais simples e mais restrito de controle de concorrência. A implementação requer uma tabela de bloqueios (<nome do item de dado, lock, transação>) e uma fila de espera.

Algoritmos de bloqueio e desbloqueio do item X:

```
lock_item(X):
  B: se LOCK(X) = 0 então  (* item desbloqueado *)
        LOCK(X) ← 1        (* bloquear o item *)
     senão início
        esperar até (LOCK(X) = 0 e o gerenciador de bloqueio despertar
                                                    a transação);
        goto B;
     fim;

unlock_item(X):
  LOCK(X) ← 0;      (* desbloquear o item *)
  se alguma transação estiver esperando então
        despertar uma das transações em espera;
```

Considere as seguintes transações:



-Bloqueio Múltiplo

Um esquema de bloqueio múltiplo (read/write ou compartilhado/exclusivo) permite que um item de dado seja acessado por mais de uma transação para leitura. As operações necessárias são: read_lock(X): bloqueia o item X para leitura, permitindo que outras transações leiam o item X (bloqueio compartilhado), write_lock(X): bloqueia o item X para gravação, mantendo o bloqueio sobre o item X (bloqueio exclusivo) e unlock(X): desbloqueia o item X. A implementação do bloqueio múltiplo requer uma tabela de bloqueios (<nome do item de dado, lock, número de leituras, transações de bloqueio>) e uma fila de espera.

Para que a técnica de bloqueio múltiplo possa ser usada, uma transação T deve obedecer às seguintes regras: T deve emitir um read_lock(X) ou write_lock(X) antes que qualquer read_item(X) seja executado em T, T deve emitir um write_lock(X) antes que qualquer write_item(X) seja executado em T, T deve emitir um unlock(X) depois que todos os read_item(X) e write_item(X) tenham sido executados em T, T não emitirá nenhum read_lock(X) ou write_lock(X) se X já estiver bloqueado por T (de forma compartilhada ou exclusiva) e T poderá emitir um unlock(X) apenas se tiver bloqueado X (de forma compartilhada ou exclusiva).

Algoritmos de bloqueio e desbloqueio do item X:

read_lock(X):

```
B: se LOCK(X) = "unlocked" então início  (* item desbloqueado *)
        LOCK(X) ← "read-locked";    (* bloquear o item p/ leitura *)
        num_de_leituras(X) ← 1;
   fim
   senão se LOCK(X) = "read-locked" então   (* bloqueado p/ leitura *)
        num_de_leituras(X) ← num_de_leituras(X) + 1;
   senão início
        esperar até (LOCK(X) = "unlocked" e o gerenciador de bloqueio
                                                despertar a transação);
        goto B;
   fim;
```

write_lock(X):

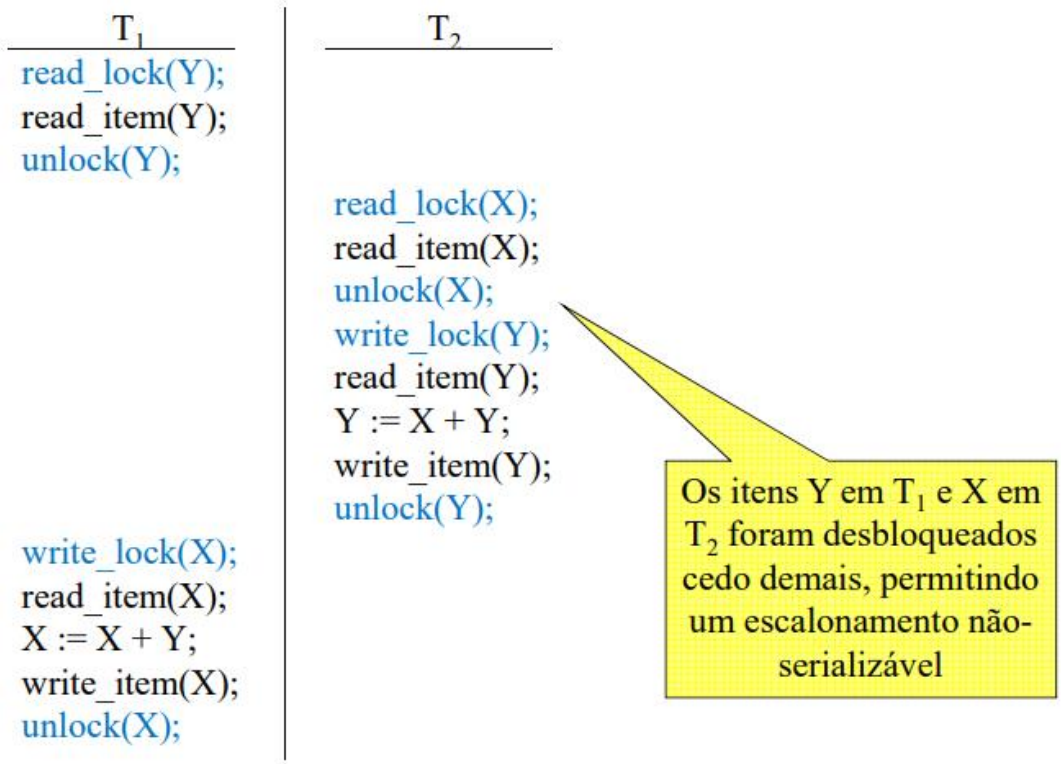
```
B: se LOCK(X) = "unlocked" então    (* item desbloqueado *)
        LOCK(X) ← "write-locked"   (* bloquear o item p/ gravação *)
   senão início
        esperar até (LOCK(X) = "unlocked" e o gerenciador de bloqueio
                                                desperta a transação);
        goto B;
   fim;
```

unlock(X):

```
se LOCK(X) = "write_locked" então início  (* bloqueado p/ gravação *)
     LOCK(X) ← "unlocked";    (* desbloquear o item *)
     despertar uma das transações em espera, se houver alguma;
fim
senão se LOCK(X) = "read-locked" então início  (* bloqueado p/ leitura *)
     num_de_leituras(X) ← num_de_leituras(X)  - 1;
     se num_de_leituras = 0 então início
          LOCK(X) ← "unlocked";    (* desbloquear o item *)
          despertar uma das transações em espera, se houver alguma;
     fim;
fim;
```

Considere as seguintes transações:

T_1	T_2
read_lock(Y);	read_lock(X);
read_item(Y);	read_item(X);
unlock(Y);	unlock(X);
write_lock(X);	write_lock(Y);
read_item(X);	read_item(Y);
X := X + Y;	Y := X + Y;
write_item(X);	write_item(Y);
unlock(X);	unlock(Y);



-Bloqueio em Duas Fases

Para garantir escalonamentos serializáveis, as operações de bloqueio e desbloqueio nas transações devem seguir protocolos. O protocolo mais usado é o protocolo de bloqueio em duas fases (Two-Phase Locking), todas as operações de bloqueio (read_lock e write_lock) precedem a primeira operação de desbloqueio (unlock). As transações são divididas em duas fases: expansão: quando são emitidos todos os bloqueios e contração: quando os desbloqueios são emitidos e nenhum novo bloqueio pode ser emitido.

Considere as seguintes transações:

T_1	T_2
read_lock(Y);	read_lock(X);
read_item(Y);	read_item(X);
unlock(Y);	unlock(X);
write_lock(X);	write_lock(Y);
read_item(X);	read_item(Y);
X := X + Y;	Y := X + Y;
write_item(X);	write_item(Y);
unlock(X);	unlock(Y);

Para seguir o protocolo, as transações foram alteradas para:

T_1'	T_2'
read_lock(Y);	read_lock(X);
read_item(Y);	read_item(X);
write_lock(X);	write_lock(Y);
unlock(Y);	unlock(X);
read_item(X);	read_item(Y);
X := X + Y;	Y := X + Y;
write_item(X);	write_item(Y);
unlock(X);	unlock(Y);

Se todas as transações em um escalonamento seguirem o protocolo de bloqueio em duas fases, o escalonamento é garantidamente serializável. Porém, não previne *deadlock*.