UNIVERSIDADE FEDERAL DE OURO PRETO – UFOP

CIÊNCIA DA COMPUTAÇÃO

BANCO DE DADOS I

RESUMO IV

Gabriel Mace dos Santos Ferreira

Marcus Vinícius Souza Fernandes

Ouro Preto

2021

Normalização para Bancos de Dados Relacionais

De acordo com o modelo relacional, temos que o esquema de um banco de dados consiste em um número de esquemas de relação, em que cada esquema é composto por um número de atributos que são agrupados de acordo com o planejamento de um projetista de banco de dados. Porém, é necessário um método para verificar a qualidade de um esquema de relação.

1. Introdução

Nesse sentido, surgiu um processo, denominado normalização, para escolher “bons” esquemas de relação, ou seja, avaliar porque um conjunto de atributos para um esquema de relação é melhor do que outro.
São utilizados quatro quesitos que compõem normas para a realização de uma avaliação informal:

- Semântica dos atributos;
- Valores redundantes em tuplas;
- Valores nulos em tuplas;
- Possibilidade de geração de tuplas espúrias.

2. Quesitos para Avaliação

2.1. Semântica dos Atributos de Relações

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 um única relação; assim, a relação não será confusa em termos semânticos.

Em suma, consiste em estabelecer nomes coerentes (significativos semanticamente) para atributos e relações, de forma com que seja simples e claro o entendimento. Além disso, também é importante se atentar a quais atributos estão presentes em uma determinada relação, caso algum deles estejam causando confusão, provavelmente não deveriam estar contidos.

2.2. Valores Redundantes em Tuplas

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 suma, consiste em evitar anomalias e redundâncias no armazenamento. Em casos de redundância controlada bem articulada visando ganho de eficiência, esta norma não se aplica.

2.3. Valores Nulos em Tuplas

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.

Em suma, consiste em evitar ao máximo a existência/presença de valores nulos. Permiti-los apenas em casos essenciais e inevitáveis, não sendo maioria em uma determinada relação. De forma, evitar a presença de valores nulos é possível criar uma nova relação composta pela chave primária da relação original e o atributo sobre o qual o valor nulo era repetido.

2.4. Possibilidade de Geração de Tuplas Espúrias

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.

De forma, a melhor compreender o tema suponha duas relações que possuam atributos semelhantes, por exemplo gênero, no entanto esses não são chaves primárias, ao realizar o casamento sobre essa informação ocorrerá a geração de tuplas espúrias dado que a união será feita sobre um atributo comum, ignorando os demais que caracterizariam unicamente uma relação.

3. Dependência Funcional

Dependência funcional consiste na dependência entre valores, contextualizando através de um exemplo, dado dois conjuntos de atributos, sejam eles X e Y, pertencentes a um esquema R, para as tuplas t1 e t2, caso

$t_1[X] = t_2[X]$, então $t_{1[}Y] = t_2[Y]$.
Isso implica que os valores de Y dependem ou são determinados pelos valores de X nas tuplas do esquema. Uma observação interessante é que X→Y não significa que Y→X.

4. Normalização

A normalização pode ser compreendida como um processo de análise de esquemas de relações com base em suas respectivas dependências funcionais e chaves primárias, no intuito de reduzir redundância. Este processo nos fornece uma série de testes de forma normal que podem ser feitos em esquemas de relações individuais. A forma normal de uma relação se refere a sua condição normal mais elevada, em suma, o grau que ela atinge é equivalente ao seu próprio grau de normalização.

5. Primeira Forma Normal

Uma determinada relação se encontra na primeira forma normal (1FN) se o domínio de todos os seus atributos possuírem apenas valores atômicos, e o valor de qualquer atributo dentre suas tuplas for único no domínio deste atributo. Em suma, seus atributos não podem ser multivalorados, compostos e/ou complexos.

Soluções:

1. Criar uma nova relação para receber o atributo multivalorado e o remover da relação atual, deixando apenas uma chave estrangeira. Desvantagem: Necessidade de junção na pesquisa.

2. Caso o valor multivalorado possua uma quantidade máxima **n** predefinida, criar na relação **n** colunas, para recebê-los. Desvantagem: Introdução de valores nulos na relação.

3. Expandir a chave primária, unindo os atributos em questão. Desvantagem: Introdução de valores redundantes na relação.

6. Segunda Forma Normal

Uma determinada relação se encontra na segunda forma normal (2FN) se um

atributo estiver na 1FN e se todo atributo não chave depende **totalmente** da chave primária. Caso a chave primária seja composta por mais de um atributo é necessário que os atributos não chave dependem de toda a chave primária, não apenas uma de suas partes.

Soluções:

1. Criar novas relações para os atributos que dependem parcialmente da chave primária da relação em questão. Desta forma, na relação inicial, é mantido apenas o atributo que satisfaz a Segunda Forma Normal.

2. Em alguns casos não há necessidade de criar novas relações, alguma relação já existente pode satisfazer essa necessidade.

7. Terceira Forma Normal

Uma determinada relação se encontra na terceira forma normal (3FN) se ela se adequar à segunda forma normal (2FN) e também se não existir dependência funcional indireta entre um determinado atributo com a chave primária da relação que ele se encontra, ou seja, todo atributo não chave deve depender diretamente a chave primária.

Soluções:

1. Separar o atributo que gera transitividade na relação, criando uma nova relação ou fazendo uso de alguma já existente que satisfaça essa necessidade.

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

1. Transações

Transação corresponde a um pacote de operações que desejamos executar em cima de um banco de dados. Ela precisa ser executada integralmente para assegurar consistência e precisão.

1.1. Possíveis instruções de transação

- Uma ou mais instruções DML (Data Manipulation Language): Instruções da linguagem de manipulacao de dados (por exemplo: uma operação de escrita sobre o banco e uma operação de consulta sobre os dados alterados);

- Uma instrução DDL (Data Definition Language): Linguagem de definição de dados, geralmente forma uma determinada transação (por exemplo quando se deseja criar uma nova tabela sobre o banco);

- Uma instrução DCL (Data Control Language): Linguagem de controle de dados (por exemplo ao desfazer uma transação);

Uma transação geralmente inicia quando a primeira instrução SQL é executada e se encerra com o surgimento de algum dos seguintes eventos:

- Comando **COMMIT** (efetivação de transação) ou **ROLLBACK** (desfazendo transação);

- Instrução DDL ou DCL é executada (commit automático);

- Desconexão do banco de dados (commit automático);

- Falha no sistema (rollback automático);

Obs.: Ao finalizar uma transação, o próximo comando inicia automaticamente a próxima transação.

1.2. Instruções de controle de transação

- **COMMIT:** Finaliza a transação atual, fazendo com que as alterações pendentes se tornem permanentes;

- **SAVEPOINT <nome_savepoint>:** Ponto de gravação, em suma, divide a transação em partes;

- **ROLLBACK [TO SAVEPOINT <nome_savepoint>]:** Finaliza a transação atual, encerrando todas as alterações pendentes; Ocorre em duas partes, o ROLLBACK finaliza a transação atual e descarta as alterações pendentes, ao acrescentar o TO SAVEPOINT é descartado o ponto de gravação e as alterações que ocorrem após o savepoint e previamente a chamada do ROLLBACK.

2. Processamento de transações

Quando várias transações são realizadas no mesmo instante, as denominamos transações concorrentes, nestes casos ocorre um entrelaçamento de operações destas transações.
O entrelaçamento funciona da seguinte maneira, uma certa quantidade de operações de uma transação são executadas, daí o processo é suspenso e as operações de outra transação se iniciam. Essa intercalação persiste até que todas elas se encontrem concluídas, vale ressaltar que durante a intercalação, as operações são retomadas do ponto de parada.

2.1. Propriedades de transações

Durante o processo de transações concorrentes devem ser mantidas algumas propriedades das transações, são elas:

- **Atomicidade:** A transação é realizada integralmente, caso contrário, não é realizada, visto que é uma unidade atômica.

- **Consistência:** A transação leva o banco de um estado consistente para outro consistente, ou seja deve ser garantida a consistência do banco.

- **Isolamento:** A execução de uma transação não pode sofrer interferência de outras concorrentes.

- **Durabilidade (ou persistência):** As alterações de uma transação não podem ser perdidas por falha, devem persistir no banco de dados.

Obs.: As propriedades podem ser referenciadas utilizando a sigla ACID.

2.2. Modelo simplificado para processamento de transações concorrentes

Operações:

- **read_item(X):** Realiza a leitura de um item X do banco de dados e o transfere para uma variável X da memória.

- **write_item(X):** Realiza a escrita de um item X da memória em um item X do banco de dados.

3. Controle de concorrência

As técnicas de controle de concorrência tem como finalidade assegurar que transações concorrentes sejam executadas de forma adequada, evitando os seguintes problemas:

- **Perda de atualização:** Ocorre quando transações que acessam os mesmos itens do banco possuem suas operações entrelaçadas, ocasionando em atualizações incorretas no banco de dados.

  Exemplo:

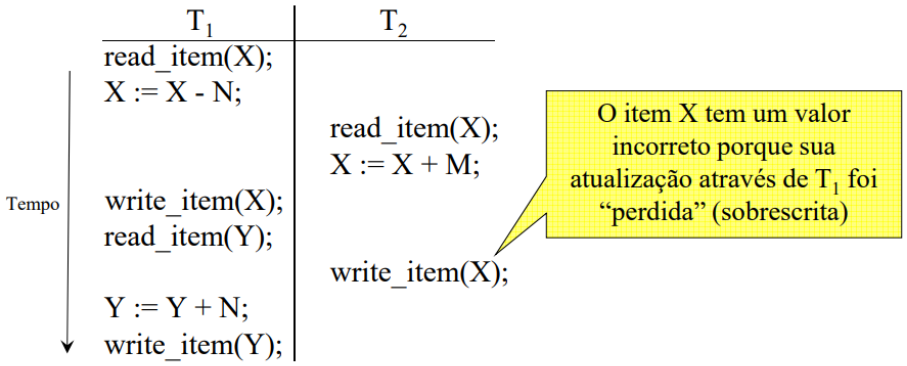

Imagem 1 - Exemplo de problema de perda de atualização (retirada dos slides do professor Guilherme Tavares de Assis).

- **Atualização temporária:** Ocorre quando uma operação falha durante uma transação, desfazendo todas as operações de escrita e leitura anteriores (rollback).

  Exemplo:

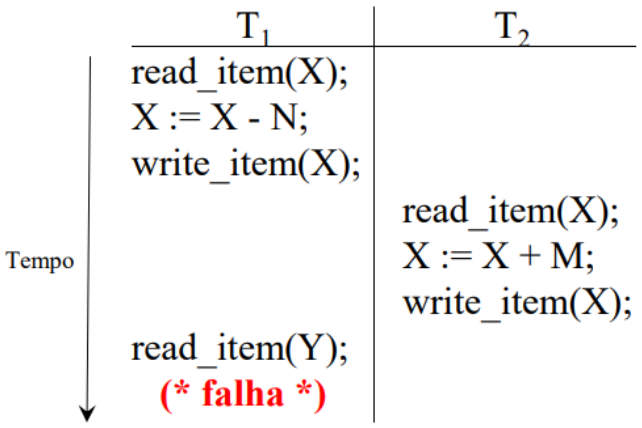

Imagem 2 - Exemplo de problema de atualização temporária (retirada dos slides do professor Guilherme Tavares de Assis).

- **Agregação incorreta:** Ocorre quando uma operação de uma determinada transação está realizando cálculos numa função de agregação e a variável que ela leu sofre alterações concorrentes.

  Exemplo:

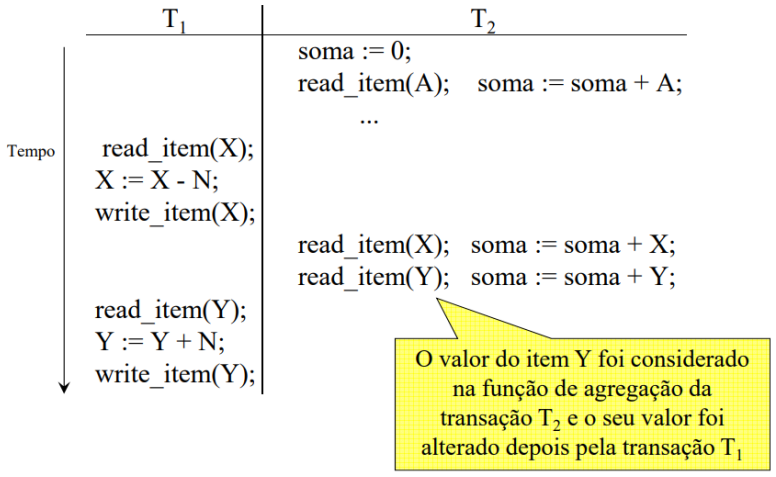

Imagem 3 - Exemplo de problema de agregação incorreta (retirada dos slides do professor Guilherme Tavares de Assis).

- **Leitura não-repetitiva:** Ocorre quando uma operação de uma transação lê um item duas vezes e o item é alterado, fazendo com que receba diferentes valores para suas duas leituras do mesmo item.

  Exemplo:

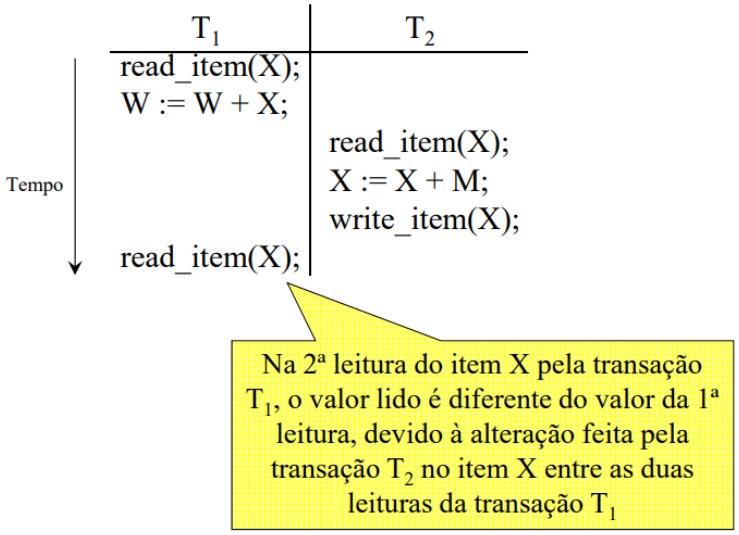

Imagem 4 - Exemplo de problema de leitura não-repetitiva (retirada dos slides do professor Guilherme Tavares de Assis).

4. Recuperação de falhas

O SGBD não deve permitir que parte de operações de uma determinada transação seja aplicada ao banco até que todas as operações em execução naquele momento sejam finalizadas. Isso pode vir a ocorrer devido a falha na execução de uma operação, possíveis tipos de falha são:

- Falha no computador;
- Erro na transação ou sistema;
- Imposição do controle de concorrência;
- Falha no disco;
- Problemas físicos;

Para os três primeiros tipos mencionados, o sistema deve manter uma quantidade suficiente de informações com o intuito de se recuperar da falha, para os demais não é possível. Para as demais falhas, é necessário o uso do mecanismo de backup que visa a recuperação de uma forma aproximada do banco de dados.

4.1. Histórico de operações que afetam o banco

Visando manter a consistência do banco, o gerenciador de recuperação sempre registra no log (histórico) cada transação por meio da notação:

- **[start_transaction, T]:** Indica que a transação dita como T iniciou a execução.

- **[write_item, T, X, old_value, new_value]:** Indica que a transação dita como T alterou o valor X do banco de dados, atualizando old_value para new_value. O registro é realizado, caso seja necessário refazer ou desfazer uma operação de escrita, sendo possível retornar ao seu valor anterior (old_value) ou alterá-lo novamente para o valor desejado (new_value).

- **[read_item, T, X]:** Indica que a transação dita como T leu o valor do item X do banco de dados. O registro é realizado, com o intuito de verificar se ao desfazer uma operação de escrita será gerada uma leitura “suja”.

- **[commit, T]:** Indica que a transação dita como T foi finalizada com sucesso. O registro é realizado, com o intuito de verificar se as alterações das transações realizadas pré-commit são permanentes.

- **[abort, T]:** Indica que a transação dita como T foi abortada. O registro é realizado, com o intuito de marcar as transações que necessitam receber o tratamento de recuperação de falhas.

Ao ocorrer uma falha, as transações que não tiveram seus registros de commit gravados no log devem ser desfeitas. Já as transações que gravaram podem ser refeitas a partir dos registros presentes no log.

4.2. Operações do processo de recuperação de falhas

No processo de recuperação de falhas relativa a uma transação T, usam-se as seguintes operações:

- **UNDO (desfazer):** Percorre o log no sentido final->começo (retroativo), desfazendo todas as alterações (write) realizadas para que retornem ao estado antigo.

- **REDO (refazer):** Percorre o log no sentido começo->final, refazendo as operações de escrita, considerando os novos valores.

5. Escalonamento e recuperabilidade

Um escalonamento **S** de **n** transações é a ordenação das operações dessas transações que são restringidas pela lógica de que para cada transação $\mathbf{T_i}$ que participa de **S**, é necessário que as operações de $\mathbf{T_i}$ apareçam na mesma ordem que ocorrem em $\mathbf{T_i}$.

Vale destacar que duas operações em um escalonamento são conflitantes quando:

- Pertencem a diferentes transações;
- Possuem acesso ao mesmo item X;
- Pelo menos uma delas é uma operação write_item(X).

5.1. Notação simplificada:

- $r_i(X)$: 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.

5.2. Exemplos

De posse da notação acima, observe os exemplos:

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

   Na situação acima está ocorrendo a perda de atualização, logo é possível deduzir que o escalonamento utilizado não está correto.

2. **S_b: $r_1(X)$; $w_1(X)$; $r_2(X)$; $w_2(X)$; $r_1(Y)$; a_1;**

   Na situação acima a transação 2 está lendo X, após ele ser alterado durante a transação 1, no entanto ocorre uma falha no final do escalonamento, acarretando em uma leitura suja.

No exemplo é possível observar todas as características de operações conflitantes nas operações $\mathbf{r_1}$ e $\mathbf{w_2}$ do primeiro exemplo.

5.3. Recuperabilidade

Um escalonamento é recuperável quando nenhuma transação **T** em **S** entrar no estado confirmado, enquanto todas as transações **T'** (transações que tenham escrito um item que **T** tenha lido) entrem no estado confirmado.

- **Exemplos**

1. **S_a: $r_1(X)$; w1(X); r2(X); r1(Y); w2(X); w1(Y); c1; $r_2(Y)$; a_1;**

   O exemplo acima é recuperável dado que a transação 2 lê o valor atualizado de X pela transação 1, no entanto essas alterações são commitadas primeiro.

2. **S_b: $r_1(X)$; w1(X); r2(X); r1(Y); w2(X); c2; a_1;**

   O exemplo acima não é recuperável dado que a transação 2 lê o item X atualizado, no entanto é feito o commit dessa antes da confirmação da transação 1.

5.4. Rollback em cascata

Um *rollback* em cascata pode ocorrer em um escalonamento recuperável, nessa situação uma transação não-confirmada precisa ser desfeita, pois leu um item de uma transação que falhou.

- **Exemplo**

1. **S_e: $r_1(X)$; $w_1(X)$; $r_2(X)$; $r_1(Y)$; $w_2(X)$; w1(Y); a1; a2;**

A fim de evitar um rollback em cascata é necessário que todas as transações no escalonamento leiam itens de transações já confirmadas ou abortadas (dependendo da situação). No exemplo anterior isso pode ser obtido com a leitura dos itens de T_1 por T_2, após a confirmação ou aborto da primeira.

6. Seriabilidade de escalonamentos

Um escalonamento S é denominado serial quando todas as transações participantes são executadas consecutivamente, caso contrário o

escalonamento é não-serial, usualmente é desejado que um escalonamento seja desse tipo. Outras características de um escalonamento serial são:

- 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;
- Na prática, é inaceitável, dado que limita a concorrência;

Obs.: Escalonamentos seriais são denominados escalonamentos totalmente corretos, dado que não sofrem com problemas de concorrência.

6.1. Escalonamento serializável

Um escalonamento de n transações é serializável quando for equivalente a um escalonamento serial das n transações, ou seja, é possível afirmar que ele é correto dado que equivale a um escalonamento serial que é considerado correto.
É importante destacar que dois escalonamentos são equivalentes ao possuírem a mesma ordem de quaisquer operações conflitantes.

- **Exemplos**

1. **S_a: $r_1(X)$; $w_1(X)$; $r_1(Y)$; $w_1(Y)$; c_1; $r_2(X)$; $w_2(X)$; c_2;** (serial);
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;** (serializável);
3. **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);

Obs.: Foram marcadas as operações conflitantes, no entanto é importante destacar que $w_1(X)$ é conflitante com $r_2(X)$ e $w_2(X)$.

7. Teste de seriabilidade

Uma forma de testar a seriabilidade de um escalonamento é por meio da construção de um grafo de precedência. A constatação se um conjunto S é serializável ou não é feita a partir da verificação se o grafo possui um ciclo ou não, caso não possua nenhum ciclo o escalonamento é considerado serializável.

7.1. Grafo de precedência

Um grafo de precedência de um escalonamento S é um grafo dirigido G = (N, E), onde N corresponde ao conjunto de nodos $\{T_1, T_2, \ldots, T_n\}$ composto pelas transações de S e E corresponde ao conjunto de arcos dirigidos $\{e_1, e_2, \ldots, e_n\}$ composto pela ligação de uma transição T_j que possui uma operação conflitante com T_k.

Exemplo:

Sa: r1(X); w1(X); r1(Y); w1(Y); c1; r2(X); w2(X); c2;
Sb: r1(X); w1(X); r2(X); w2(X); r1(Y); w1(Y); c1; c2;
Sc: r1(X); r2(X); w1(X); r1(Y); w2(X); w1(Y); c1; c2;

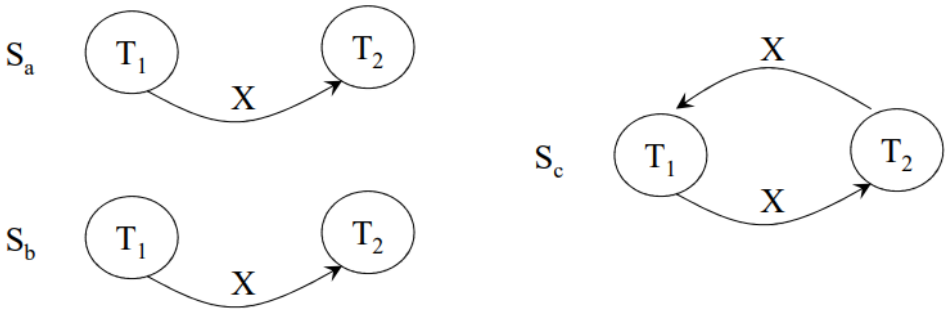

*Imagem 5 - Exemplos para os escalonamentos de **Sa**, **Sb** e **Sc**, resultando em seus respectivos grafos de precedência (retirada dos slides do professor Guilherme Tavares de Assis).*

8. Técnicas de bloqueio (Locking)

Um bloqueio é utilizado com o intuito de sincronizar o acesso de transações concorrentes aos itens de dados. Usualmente ele é constituído por uma variável associada a um item de dado, que descreve o status do item em relação às operações que podem ser realizadas sobre ele.

Obs.: Em geral, existem bloqueios para cada item de dado no banco de dados.

Nos subtópicos seguintes seguem algumas técnicas de bloqueio:

8.1. Bloqueio binário

O bloqueio binário é o mecanismo mais simples e mais restrito do controle de concorrência, usualmente é constituído por dois estados bloqueado (*locked*) ou desbloqueado (*unlocked*), por possuir apenas dois estados duas operações são necessárias à ele:

- **lock_item(X):** bloqueia o item X;
- **unlock_item(X):** desbloqueia o item X.

A partir dessas duas operações é possível impor a exclusão mútua sobre o item de dado, ou seja, caso o item seja bloqueado por uma transação nenhuma outra transação terá acesso a esse item, enquanto a transação que o bloqueou o desbloqueie. Enquanto o item não é desbloqueado, as transações que querem acessá-lo são colocadas em uma fila de espera para o item.

Uma observação relevante é que caso todos os desbloqueios se encontrem ao final de uma operação pode ocorrer um deadlock, ou seja, uma transação necessita de um item de dado para continuar suas operações, enquanto outra transação está utilizando o item de dado que a primeira transação deseja, mas necessita do item que a primeira utiliza. Dessa forma, ocorre um deadlock que fará com que o sistema interrompa as duas transações.

8.1.1. Condições para a criação do bloqueio binário

As seguintes regras devem ser obedecidas para que o bloqueio binário seja utilizado:

- **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**;

- **T** poderá emitir um **unlock_item(X)** apenas se tiver bloqueado **X**.

Além dessas regras, é necessária a implementação de uma tabela de bloqueios (<nome do item de dado, *lock*, transação>) e uma fila de espera.

Outra observação relevante, em relação ao bloqueio múltiplo, ele não garante o escalonamento serializável dado que caso ocorra o desbloqueio de dois itens de dados gerando um grafo de precedência com ciclo, que como estudado anteriormente indica um escalonamento não-serializável.

8.1.2. Algoritmos de bloqueio e desbloqueio

```
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;
```

Imagem 6 - Exemplo de um algoritmo de bloqueio (retirada dos slides do professor Guilherme Tavares de Assis).

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

Imagem 7 - Exemplo de um algoritmo de desbloqueio(retirada dos slides do professor Guilherme Tavares de Assis).

8.2. Bloqueio múltiplo

O bloqueio múltiplo, também conhecido como *read*/*write* ou compartilhado/exclusivo, permite que um item do banco de dados seja acessado por mais de uma transação para leitura. Para realizar tais funcionalidades são necessárias as operações:

- **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);

- **unlock(X):** desbloqueia o item X.

8.2.1. Condições para a criação do bloqueio múltiplo

As seguintes regras devem ser obedecidas para que o bloqueio binário seja utilizado:

- **T** deve emitir um **read_lock(X)** ou **write_lock(X)** antes que qualquer **read_item(X)** seja executado;

- **T** deve emitir um **write_lock(X)** antes que qualquer **read_item(X)** seja executado;

- **T** deve emitir um **unlock(X)** depois que todos os **read_item(X)** e **write_item(X)** tenham sido executados;

- **T** não emitirá um **read_lock(X)** ou **write_lock(X)** se X já esteve bloqueado por T (seja de forma compartilhada ou exclusiva);

- **T** poderá emitir um **unlock(X)** apenas se tiver bloqueado X (seja de forma compartilhada ou exclusiva);

Além dessas regras é necessária a implementação de uma tabela de bloqueios (<nome do item de dado, *lock*, número de leituras, transações de bloqueio>) e uma fila de espera.

Vale ressaltar, o número de leituras corresponde ao número de transações acessando o item naquele momento, ou seja, caso esteja sendo utilizado um read_lock o valor poderá ser maior ou igual a um, enquanto se estiver sendo utilizado o write_lock será apenas a transação que realizou o bloqueio.

8.2.2. Algoritmos de bloqueio e desbloqueio

```
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;
```

Imagem 8 - Exemplo de um algoritmo de bloqueio para leitura (retirada dos slides do professor Guilherme Tavares de Assis).

```
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;
```

Imagem 9 - Exemplo de um algoritmo de bloqueio para escrita (retirada dos slides do professor Guilherme Tavares de Assis).

```
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;
```

Imagem 10 - Exemplo de um algoritmo de desbloqueio (retirada dos slides do professor Guilherme Tavares de Assis).

8.3. Bloqueio em duas fases

As operações de bloqueio e desbloqueio devem seguir protocolos para manterem escalonamentos serializáveis. Usualmente, o protocolo mais utilizado é o de bloqueio em duas fases (*Two-Phase Locking*).
Nesse protocolo todas as operações de bloqueio (**read_lock** e **write_lock**) precedem a primeira operação de desbloqueio (*unlock*), como o nome indica as transações são divididas em duas fases:

- **Expansão:** quando são emitidos todos os bloqueios;
- **Contração:** quando os desbloqueios são emitidos e nenhum novo bloqueio pode ser emitido.

Como dito anteriormente os protocolos garantem que o escalonamento será serializável, dessa forma caso todas as transações sigam o protocolo de bloqueio em duas fases é garantido que o escalonamento será serializável, no entanto ainda poderá ocorrer um *deadlock*.