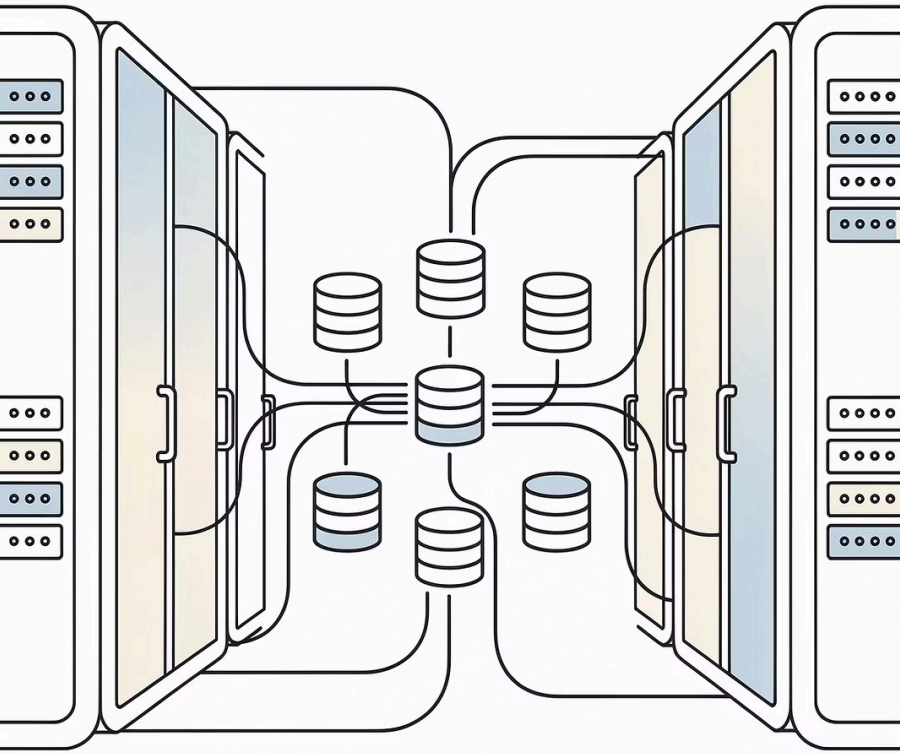


CEFET/RJ

BANCOS DE DADOS

Controle de Concorrência em Sistemas de Bancos de Dados

O controle de concorrência é um dos pilares fundamentais para garantir a integridade e consistência dos dados em sistemas de bancos de dados modernos. Este conjunto de técnicas permite que múltiplas transações sejam executadas simultaneamente, mantendo a correção dos resultados.

Eduardo Ogasawara

eduardo.ogasawara@cefet-rj.br
https://eic.cefet-rj.br/~eogasawara

Objetivo do Controle de Concorrência

Em ambientes de produção modernos, múltiplas transações precisam ser executadas simultaneamente para atender diversos usuários e aplicações. A execução concorrente traz benefícios significativos ao sistema.

A principal vantagem está na **melhoria da utilização do sistema**, aproveitando melhor os recursos de hardware disponíveis. Além disso, proporciona **redução no tempo de resposta** para os usuários finais.

O controle de concorrência atua como guardião, garantindo que o resultado da execução concorrente seja equivalente a alguma execução serial das transações. Este conceito fundamental é conhecido como **serializabilidade**.

Serializabilidade de Schedules

Schedule Serial

Executa transações uma após a outra, de forma sequencial. Garante correção mas limita a concorrência e o desempenho do sistema.

Schedule Concorrente

Intercala operações de diferentes transações. Permite melhor utilização de recursos e maior throughput do sistema.

Schedule Serializável

Seu efeito final é equivalente ao de algum schedule serial. Combina as vantagens da concorrência com a garantia de correção.

O objetivo do controle de concorrência é permitir que schedules concorrentes sejam executados, desde que sejam serializáveis. Dessa forma, conseguimos concorrência sem perder correção, maximizando o desempenho enquanto mantemos a consistência dos dados.

BLOQUEIOS

Protocolos Baseados em Bloqueio

O bloqueio é um dos mecanismos mais utilizados para controlar o acesso simultâneo a itens de dados. Funciona como um sistema de permissões que regula quais transações podem acessar quais dados e de que forma.

Modo Exclusivo (X)

Permite que o item de dados seja tanto **lido quanto escrito**. Solicitado através da instrução `lock-X`. Nenhuma outra transação pode acessar o item simultaneamente.

Modo Compartilhado (S)

Permite que o item de dados seja apenas **lido**. Solicitado através da instrução `lock-S`. Múltiplas transações podem manter bloqueios compartilhados simultaneamente.

> As solicitações de bloqueio são feitas ao gerenciador de controle de concorrência. A transação só pode prosseguir após a concessão da solicitação, garantindo acesso coordenado aos dados.

Matriz de Compatibilidade de Bloqueios

A compatibilidade entre bloqueios determina quando uma transação pode receber um bloqueio sobre um item já bloqueado por outra transação. Esta matriz define as regras fundamentais do sistema de bloqueio.

	S	X
S	true	false
X	false	false

A matriz mostra que bloqueios compartilhados (S) são compatíveis entre si, permitindo leituras simultâneas. Já bloqueios exclusivos (X) são incompatíveis com qualquer outro bloqueio.

- **Regra Fundamental**

  Qualquer quantidade de transações pode manter bloqueios compartilhados sobre um item simultaneamente.

- **Exclusividade**

  Se uma transação mantém um bloqueio exclusivo, nenhuma outra pode manter qualquer tipo de bloqueio sobre o mesmo item.

- **Espera por Concessão**

  Se um bloqueio não puder ser concedido imediatamente, a transação deve aguardar até que todos os bloqueios incompatíveis sejam liberados.

Exemplo de Transação com Bloqueio

> **Importante:** Simplesmente usar bloqueios não é suficiente para garantir a serializabilidade. É necessário um protocolo bem definido que especifique quando solicitar e quando liberar bloqueios.

Cenário Problemático

Considere uma transação que lê os valores de A e B para calcular sua soma. Se os bloqueios forem liberados prematuramente:

- A transação lê o valor de A
- Libera o bloqueio sobre A
- Outra transação atualiza A e B
- A primeira transação lê B
- A soma calculada estará incorreta

Necessidade de Protocolos

Um protocolo de bloqueio precisa estabelecer uma **política clara** para solicitar e liberar bloqueios. Estes protocolos restringem o conjunto de schedules possíveis, mas garantem a correção.

PROBLEMA CRÍTICO

Armadilha: Impasse (Deadlock)

T_3	T_4
lock-X(B)	
read(B)	
$B := B - 50$	
write(B)	
	lock-S(A)
	read(A)
	lock-S(B)
lock-X(A)	

O impasse ocorre quando duas ou mais transações ficam aguardando indefinidamente por bloqueios mantidos umas pelas outras, criando um **ciclo de espera** do qual nenhuma consegue sair.

> *"O impasse é uma consequência inevitável da maioria dos protocolos de bloqueio. Sistemas de bancos de dados precisam ter mecanismos para detectar e resolver estas situações."*

No exemplo ilustrado, temos uma situação clássica de impasse onde cada transação mantém um bloqueio que a outra necessita, criando uma dependência circular irresolvível sem intervenção externa.

Armadilha: Inanição (Starvation)

Além do impasse, outro problema crítico nos sistemas de controle de concorrência é a inanição, onde uma transação fica indefinidamente aguardando por recursos que nunca são concedidos.

Cenário de Bloqueios Compartilhados

Uma transação espera por um bloqueio exclusivo (X) sobre um item, enquanto uma **sequência contínua** de outras transações solicita e recebe bloqueios compartilhados (S) sobre o mesmo item. A transação original nunca consegue seu bloqueio exclusivo.

Reversões Repetidas

Uma mesma transação é repetidamente revertida devido a impasses. O sistema sempre escolhe esta transação como vítima, impedindo que ela jamais seja completada com sucesso.

Solução: O gerenciador de controle de concorrência deve ser projetado cuidadosamente para prevenir inanição, implementando políticas justas de concessão de bloqueios e seleção de vítimas em situações de impasse.

PROTOCOLO FUNDAMENTAL

O Protocolo de Bloqueio em Duas Fases (2PL)

O protocolo de bloqueio em duas fases é um dos mecanismos mais importantes para garantir a serializabilidade por conflito. Ele divide a execução de cada transação em duas fases distintas e bem definidas.

Fase 1: Crescimento

- A transação **pode obter** novos bloqueios
- A transação **não pode liberar** bloqueios
- Acumula os recursos necessários

Fase 2: Encolhimento

- A transação **pode liberar** bloqueios
- A transação **não pode obter** novos bloqueios
- Libera gradualmente os recursos

Garantia de Serializabilidade

O protocolo 2PL garante que as transações podem ser serializadas na ordem de seus pontos de bloqueio - o momento em que cada transação adquire seu último bloqueio antes de começar a liberá-los.

Esta propriedade pode ser formalmente provada e é a base para garantir a correção em sistemas concorrentes.

Tipos de Serializabilidade

Existem diferentes formas de definir e verificar se um schedule é serializável. Cada abordagem tem suas características e aplicações práticas em sistemas de bancos de dados.

Serializabilidade por Conflito

Baseada na análise da **ordem de operações conflitantes** entre transações. Duas operações conflitam se:

- Pertencem a transações diferentes
- Acessam o mesmo item de dados
- Pelo menos uma é uma operação de escrita

Pode ser verificada através da construção de grafos de precedência.

Serializabilidade por Visão

Baseada nos **valores lidos e escritos** por cada transação. Considera:

- Qual transação escreveu o valor lido
- Qual transação realizou a escrita final
- Leituras de valores não modificados

Define um conjunto mais amplo de schedules corretos que a serializabilidade por conflito.

Na prática, a maioria dos protocolos de controle de concorrência implementados em sistemas comerciais garante **serializabilidade por conflito**, por ser mais eficiente de verificar e implementar.

Conversões de Bloqueio no Protocolo 2PL

Uma versão refinada do protocolo de duas fases permite a conversão de bloqueios, proporcionando maior flexibilidade e eficiência no gerenciamento de recursos.

Fase de Crescimento (Expansão)

- Pode adquirir um **bloqueio-S** (compartilhado) sobre qualquer item
- Pode adquirir um **bloqueio-X** (exclusivo) sobre qualquer item
- Pode converter um bloqueio-S para bloqueio-X (upgrade)

Fase de Encolhimento (Contração)

- Pode converter um bloqueio-X para bloqueio-S (downgrade)
- Pode liberar um **bloqueio-X** (exclusivo)
- Pode liberar um **bloqueio-S** (compartilhado)

Observação Importante: Este protocolo garante a serializabilidade. No entanto, como apresentado, ele demanda que o programador insira manualmente as diversas instruções de bloqueio, upgrade e downgrade no código das transações.

Variações do Protocolo de Duas Fases

Diferentes variações do protocolo 2PL foram desenvolvidas para atender necessidades específicas de sistemas de bancos de dados, cada uma com seus trade-offs entre concorrência, simplicidade e capacidade de recuperação.

2PL Básico

Características:

- Garante serializabilidade por conflito
- Permite deadlocks
- Maior concorrência possível

É a forma padrão do protocolo, sem restrições adicionais.

2PL Conservador

Características:

- Todos os bloqueios adquiridos antes da execução
- Evita completamente deadlocks
- Reduz a concorrência

Útil quando os deadlocks são muito custosos.

2PL Rigoroso

Características:

- Bloqueios exclusivos liberados apenas no commit
- Facilita muito a recuperação
- Evita leituras sujas

Amplamente usado em sistemas comerciais.

A escolha do protocolo afeta diretamente três aspectos críticos do sistema: o nível de **concorrência** alcançável, a complexidade dos mecanismos de **recuperação**, e a **complexidade geral** de implementação e manutenção do sistema.

Aquisição Automática de Bloqueios

Para facilitar o desenvolvimento de aplicações, os sistemas de bancos de dados implementam mecanismos automáticos de aquisição de bloqueios. Isso libera o programador da responsabilidade de gerenciar explicitamente os bloqueios.

Processamento da Operação Read(D)

```
if Ti tem um bloqueio sobre D then
   read(D)
else
begin
   se necessário, espera até que nenhuma outra
   transação tenha um bloqueio-X sobre D

   concede a Ti um bloqueio-S sobre D;
   read(D)
end
```

O sistema verifica automaticamente se a transação Ti já possui algum bloqueio sobre o item D. Se não possuir, o gerenciador de bloqueios garante que nenhuma transação mantenha um bloqueio exclusivo sobre D antes de conceder um bloqueio compartilhado e realizar a leitura.

Este processo é **transparente para o programador**, que simplesmente emite instruções padrão de leitura e escrita sem chamadas explícitas de bloqueio.

Processamento Automático de Write

O processamento de operações de escrita requer lógica mais complexa, pois pode envolver upgrade de bloqueios compartilhados para exclusivos.

Processamento da Operação Write(D)

```
if Ti tem um bloqueio-X sobre D
   then
      write(D)
   else
   begin
      se for preciso, espera até que nenhuma outra
      transação tenha um bloqueio sobre D,

      if Ti tem um bloqueio-S sobre D
         then
            upgrade do bloqueio sobre D para bloqueio-X
         else
            concede a Ti um bloqueio-X sobre D

      write(D)
   end
```

Verificação Inicial

Sistema verifica se a transação já possui bloqueio exclusivo

Espera por Compatibilidade

Aguarda até que não haja bloqueios incompatíveis

Upgrade ou Concessão

Realiza upgrade de S para X, ou concede novo bloqueio-X

Execução

Realiza a operação de escrita com segurança

Liberação de Bloqueios: Todos os bloqueios são automaticamente liberados após o **commit** ou **abort** da transação, garantindo o protocolo de duas fases.

Implementação do Gerenciador de Bloqueios

O gerenciador de bloqueios é tipicamente implementado como um componente especializado do SGBD, responsável por coordenar todas as solicitações de bloqueio e garantir a correção da execução concorrente.

Arquitetura do Sistema

O gerenciador pode ser implementado como um **processo separado** ao qual as transações enviam:

- Solicitações de bloqueio
- Solicitações de desbloqueio
- Notificações de commit/abort

As respostas incluem:

- Mensagem de concessão de bloqueio
- Mensagem de rollback em caso de deadlock

Espera Sincronizada

A transação solicitante aguarda até que sua solicitação seja respondida pelo gerenciador

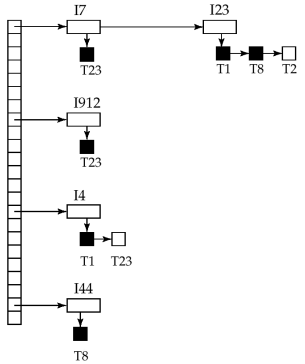
Tabela de Bloqueios

Estrutura de dados que registra bloqueios concedidos e solicitações pendentes

Implementação Eficiente

Tipicamente usa tabela hash em memória indexada pelo nome do item de dados

Estrutura da Tabela de Bloqueios



Legenda: Retângulos pretos indicam bloqueios concedidos, retângulos brancos indicam solicitações aguardando.

Operações na Tabela de Bloqueios

Registro de Tipo

A tabela registra o **tipo de bloqueio** concedido ou solicitado (S ou X) para cada entrada

Fila de Solicitações

Novas solicitações são **acrescentadas ao final da fila** para o item de dados específico

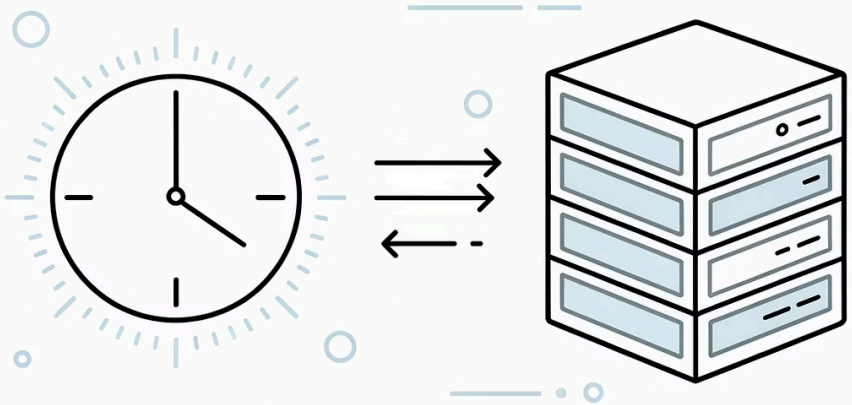
Concessão por Compatibilidade

Uma solicitação é concedida se for compatível com todos os bloqueios anteriores na fila

Processamento de Unlock

Solicitações de desbloqueio resultam na **exclusão da entrada**, e solicitações posteriores são verificadas

Otimização: Se uma transação abortar, todas as suas solicitações (aguardando ou concedidas) são excluídas. O gerenciador mantém uma lista de bloqueios por transação para implementar isso eficientemente.



ALTERNATIVA AO BLOQUEIO

Protocolos Baseados em Estampa de Tempo

Uma abordagem alternativa aos protocolos baseados em bloqueio para controle de concorrência utiliza **estampas de tempo** (timestamps) para ordenar as transações. Cada transação recebe um identificador único baseado no momento de seu início.

Este método evita completamente deadlocks, pois não usa espera: em vez disso, transações que violam a ordem são abortadas e reiniciadas. A serializabilidade é garantida pela ordem das estampas de tempo.

Funcionamento dos Protocolos de Timestamp

Os protocolos baseados em estampa de tempo utilizam um sistema de ordenação temporal para garantir a execução serializável sem necessidade de bloqueios explícitos.

1 Atribuição de TS

Cada transação Ti recebe uma estampa de tempo única TS(Ti) no momento de seu início

2 Ordem de Serialização

A ordem das estampas determina a ordem de serialização equivalente

3 Verificação em Cada Acesso

Sistema verifica se a operação respeita a ordem temporal

4 Abortar se Necessário

Transações que violam a ordem são abortadas e reiniciadas

Estampas nos Itens de Dados

Cada item de dados X mantém duas estampas:

- **W-timestamp(X):** maior estampa de transação que escreveu X com sucesso
- **R-timestamp(X):** maior estampa de transação que leu X com sucesso

O protocolo garante que qualquer ordem de execução conflitante seja equivalente à ordem das estampas de tempo, proporcionando serializabilidade sem deadlocks.

Regras do Protocolo de Timestamp

O protocolo define regras específicas para operações de leitura e escrita, garantindo que a ordem temporal seja respeitada em todas as situações.

Read(X) por Ti

Se TS(Ti) < W-timestamp(X):

Ti precisa ler um valor de X que já foi sobrescrito. A operação é rejeitada e Ti é **abortada**.

Se TS(Ti) ≥ W-timestamp(X):

A operação é executada com sucesso e R-timestamp(X) é atualizado para max(R-timestamp(X), TS(Ti)).

Write(X) por Ti

Se TS(Ti) < R-timestamp(X):

O valor de X que Ti está produzindo foi necessário por uma transação com estampa maior. A operação é rejeitada e Ti é **abortada**.

Se TS(Ti) < W-timestamp(X):

Ti está tentando escrever um valor obsoleto de X. A operação é rejeitada e Ti é **abortada**.

Caso contrário:

A operação é executada e W-timestamp(X) é atualizado para TS(Ti).

Vantagem Principal: Este protocolo é livre de deadlocks, pois nenhuma transação espera por outra. A desvantagem é que pode haver mais aborts e rollbacks do que em protocolos baseados em bloqueio.

Exemplo de Uso do Protocolo de Timestamp

Vamos analisar um schedule parcial com quatro transações concorrentes que possuem estampas de tempo 1, 2, 3 e 4, demonstrando como o protocolo funciona na prática.

T_1	T_2	T_3	T_4
			read(x)
	read(y)		
read(y)			
		write(y)	
		write(z)	**read(z)**
	write(x)		
	abort		
read(x)			
		write(z)	
		abort	
			write(y)
			write(z)

Análise do Exemplo

Neste cenário, todas as operações são executadas com sucesso porque respeitam a ordem temporal estabelecida pelas estampas.

As leituras e escritas ocorrem em ordem crescente de timestamps, garantindo **serializabilidade**.

Cenário de Conflito

Se T1 tentasse escrever em A após T2 já ter escrito (TS(T1)=1 < W-timestamp(A)=2), a operação seria rejeitada e T1 seria abortada e reiniciada com uma nova estampa maior.

Quando a transação **T_3** reverte, a transação **T_4** acaba sendo revertida em cascata, pelo fato da leitura de z ter sido influenciada por **T_3**.

> O protocolo de timestamp oferece uma alternativa interessante aos bloqueios, especialmente em ambientes onde deadlocks são problemáticos ou onde há poucas contenções por recursos.

CONTROLE DE CONCORRÊNCIA

Garantias do Protocolo de Estampa de Tempo

Garantia de Seriação

O protocolo garante a seriação porque todos os arcos no grafo de precedência seguem uma direção consistente: sempre da transação com menor estampa de tempo para a transação com maior estampa de tempo. Esta propriedade fundamental assegura que não haverá ciclos no grafo de precedência.

Sem ciclos no grafo, o sistema mantém uma ordem total consistente entre as transações concorrentes, preservando a equivalência com alguma execução serial das mesmas transações.

Liberdade de Impasse

Uma característica crucial do protocolo é a garantia de liberdade de impasse (deadlock freedom). Diferentemente de protocolos baseados em bloqueio, nenhuma transação precisa esperar por recursos mantidos por outras transações.

As transações são simplesmente abortadas e reiniciadas quando detectados conflitos, eliminando completamente a possibilidade de impasses, onde duas ou mais transações ficam eternamente esperando umas pelas outras.

Transação com Menor Estampa de Tempo

Transações iniciadas anteriormente recebem prioridade no sistema. Sua estampa de tempo menor indica que foram requisitadas primeiro cronologicamente.

Fluxo de Precedência

O grafo de precedência sempre direciona da transação mais antiga para a mais recente, estabelecendo uma ordem clara e inequívoca.

Transação com Maior Estampa de Tempo

Transações iniciadas posteriormente têm menor prioridade. Devem aguardar ou serem abortadas em caso de conflito com transações mais antigas.

Esta ordenação temporal é a base para a consistência do protocolo, criando uma hierarquia natural entre transações concorrentes que previne tanto ciclos quanto impasses no sistema.

Regra do Write de Thomas

A Regra do Write de Thomas (Thomas Write Rule) é uma otimização importante do protocolo básico de ordenação por estampa de tempo. Esta regra permite ignorar certas operações de escrita obsoletas, aumentando significativamente a concorrência do sistema sem comprometer a corretude.

Conceito Fundamental

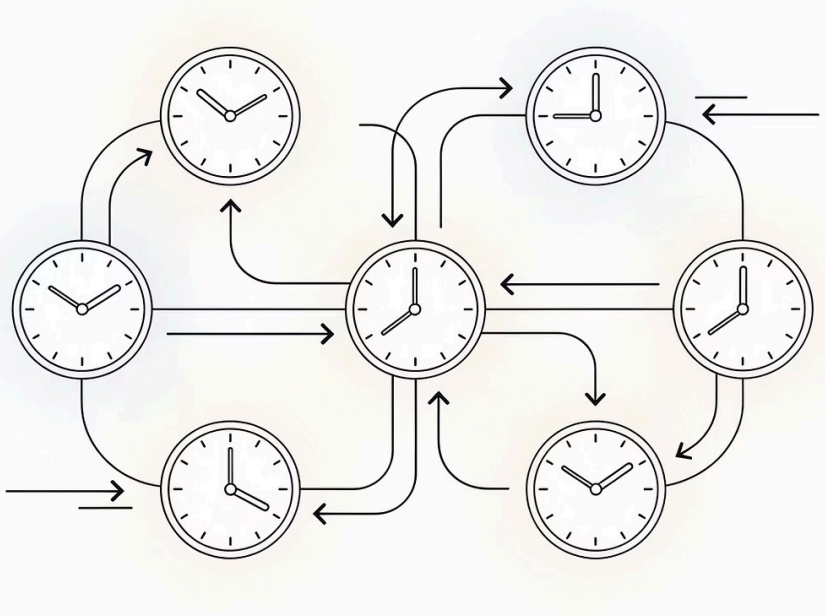
Permite que escritas obsoletas sejam simplesmente ignoradas em vez de causarem abort da transação, quando uma escrita mais recente já foi realizada no mesmo item de dados.

Condição de Aplicação

Se uma transação Ti tenta escrever em um item Q, mas outra transação Tj com estampa maior já escreveu em Q, a escrita de Ti é simplesmente ignorada pois está obsoleta.

Benefício de Desempenho

Reduz o número de aborts desnecessários, permitindo maior grau de concorrência e melhor desempenho geral do sistema de gerenciamento de transações.



Protocolos de Validação

Os protocolos de validação representam uma estratégia alternativa e otimista para o controle de concorrência em sistemas de bancos de dados. Ao invés de impor bloqueios ou ordenar transações durante sua execução, estes protocolos permitem que as transações operem livremente, validando suas ações apenas no momento da confirmação. Este método visa maximizar a concorrência e o desempenho, verificando a consistência dos dados apenas quando uma transação está prestes a ser finalizada.

VALIDAÇÃO

Protocolo Baseado em Validação

O protocolo de validação, também conhecido como certificação otimista, assume que conflitos entre transações são raros. Sob essa premissa, as transações executam sem restrições e são validadas apenas no momento do commit, aumentando significativamente a concorrência.

Fase de Leitura

A transação executa livremente, lendo valores do banco de dados e realizando computações. Todas as escritas são feitas apenas em cópias locais privadas.

1

Fase de Validação

Ao final da execução, o sistema verifica se há conflitos com outras transações. Testa se a execução é equivalente a alguma ordem serial.

2

Fase de Escrita

Se a validação for bem-sucedida, as atualizações locais são aplicadas permanentemente ao banco de dados. Caso contrário, a transação é abortada e reiniciada.

3

Estrutura de Fases do Protocolo de Validação

Fase 1: Leitura

- Início até o momento antes da validação
- Leitura de itens do banco de dados
- Escritas mantidas em variáveis locais temporárias
- Sem interferência de outras transações

Fase 2: Validação

- Teste de seriação é executado
- Verifica conflitos com transações concorrentes
- Decisão de commit ou abort
- Período crítico do protocolo

Fase 3: Escrita

- Ocorre apenas se validação bem-sucedida
- Atualização do banco de dados
- Alterações se tornam permanentes
- Liberação de recursos

Cada transação recebe três estampas de tempo: Start(Ti) marca o início da fase de leitura, Validation(Ti) marca o início da validação, e Finish(Ti) marca a conclusão da fase de escrita. Estas estampas são fundamentais para os testes de validação.

Teste de Validação para a Transação Tj

O teste de validação determina se uma transação pode realizar commit com segurança. Para que a transação Tj seja validada com sucesso, pelo menos uma das seguintes condições deve ser verdadeira para cada transação Ti que completou sua escrita:

Condição Temporal de Não-Sobreposição

Finish(Ti) < Start(Tj): A transação Ti completou sua fase de escrita antes que Tj iniciasse sua fase de leitura. Portanto, não há sobreposição temporal entre as transações, garantindo ausência de conflitos.

Condição de Conjunto Disjunto de Escrita

Start(Tj) < Finish(Ti) < Validation(Tj), e o conjunto de itens escritos por Ti não intercepta o conjunto de itens lidos por Tj. Mesmo havendo sobreposição temporal, os dados acessados são distintos.

Condição de Isolamento de Leitura-Escrita

Start(Tj) < Finish(Ti) < Finish(Tj), e o conjunto de itens escritos por Ti não intercepta nem o conjunto lido nem o conjunto escrito por Tj. Garante que Ti não influencia as operações de Tj.

Nota Importante: Se nenhuma dessas condições for satisfeita para alguma transação Ti, então Tj falha na validação e deve ser abortada e reiniciada para garantir a seriação.

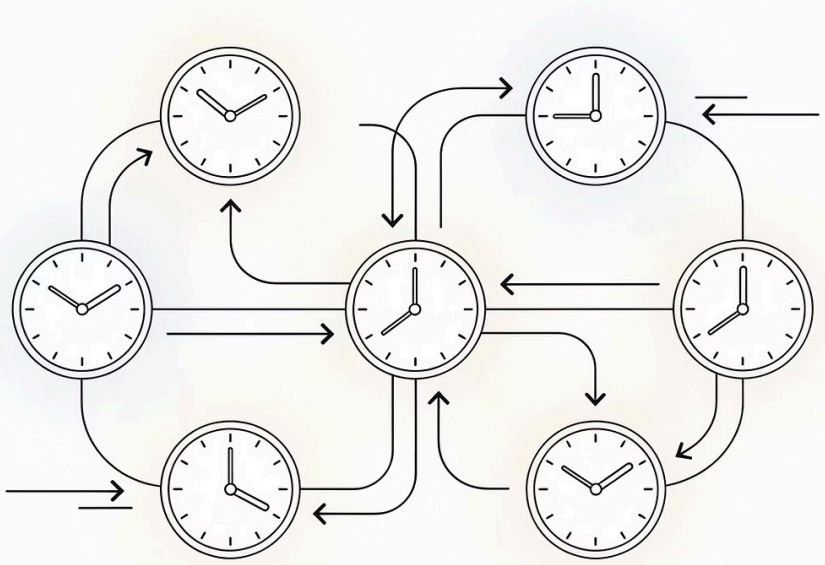
EXEMPLO PRÁTICO

Schedule Produzido por Validação

Considere um exemplo concreto de como o protocolo de validação produz um schedule válido. Analisaremos a execução de duas transações concorrentes T14 e T15 que acessam os mesmos itens de dados:

T_14	T_15
read(b)	
	read(b)
	b := b – 50
	read(a)
	a := a + 50
read(a) **(validate)**	
display(a+b)	
	(validate)
	write(b)
	write(a)

Neste exemplo, as transações T14 e T15 conseguem avançar sem problemas, pois a escrita em b é deixada para fase de validação.



Controle de Concorrência Multiversão

Os esquemas de controle de concorrência multiversão (MVCC) representam uma abordagem avançada para gerenciar transações em bancos de dados. Em vez de permitir apenas uma versão de um item de dado, o MVCC mantém múltiplas versões de cada item de dado, permitindo que transações de leitura acessem versões mais antigas do dado sem serem bloqueadas por transações de escrita. Este método otimista melhora significativamente a concorrência, reduzindo o número de bloqueios e conflitos, sendo particularmente eficaz em sistemas com alta carga de leitura. Ele serve como uma alternativa poderosa aos protocolos baseados em bloqueio ou validação, visando maximizar o desempenho e a capacidade de resposta do sistema.

Esquemas Multiversão

Os esquemas multiversão representam uma abordagem sofisticada para controle de concorrência que mantém múltiplas versões históricas de cada item de dados. Esta técnica permite aumentar significativamente o grau de concorrência no sistema sem sacrificar a corretude.

Princípio Fundamental

Em vez de sobrescrever dados, cada operação de escrita bem-sucedida cria uma nova versão do item. Versões antigas são preservadas com suas respectivas estampas de tempo.

Benefício para Leituras

Operações de leitura nunca precisam esperar, pois sempre existe uma versão apropriada disponível. O sistema seleciona automaticamente a versão correta baseada na estampa de tempo da transação leitora.

Variantes Principais

Dois protocolos importantes: ordenação de estampa de tempo multiversão e bloqueio em duas fases multiversão. Ambos exploram múltiplas versões para aumentar concorrência.

Seleção de Versão

Quando uma transação Ti emite read(Q), o sistema seleciona a versão de Q cuja estampa de tempo de escrita é a maior que é menor ou igual à estampa de tempo de Ti. Esta seleção garante consistência temporal.

MULTIVERSÃO

Ordenação de Estampa de Tempo Multiversão

Este protocolo estende o conceito básico de ordenação por estampa de tempo mantendo múltiplas versões de cada item de dados. Cada versão possui metadados específicos que permitem ao sistema determinar qual versão deve ser acessada por cada transação.

Estrutura de Versões

Cada item Q tem uma sequência de versões Q1, Q2, ..., Qm. Cada versão Qk contém três campos: Content (o valor), W-timestamp (estampa da transação que criou), e R-timestamp (maior estampa das transações que leram).

Operação de Leitura

Quando Ti executa read(Q), o sistema seleciona a versão Qk onde W-timestamp(Qk) é o maior menor ou igual a TS(Ti). A leitura sempre tem sucesso, e R-timestamp(Qk) é atualizado para max(R-timestamp(Qk), TS(Ti)).

Operação de Escrita

Quando Ti executa write(Q), o sistema localiza Qk onde W-timestamp(Qk) é a maior menor ou igual a TS(Ti). Se TS(Ti) < R-timestamp(Qk), a escrita é rejeitada e Ti é abortada. Caso contrário, uma nova versão é criada.

A principal vantagem deste protocolo é que operações de leitura nunca são rejeitadas e nunca precisam esperar, pois sempre existe uma versão apropriada disponível. Apenas escritas podem resultar em rollback, quando tentam modificar dados já lidos por transações futuras.

Comportamento do Protocolo Multiversão

Mecanismo de Decisão para Escritas

Quando uma transação Ti tenta escrever em Q:

1. Sistema localiza a versão Qk mais recente onde W-timestamp(Qk) ≤ TS(Ti)
2. Verifica se TS(Ti) < R-timestamp(Qk)
3. Se sim: escrita seria obsoleta, Ti é abortada
4. Se não: nova versão Qi é criada com W-timestamp(Qi) = TS(Ti)

Esta verificação garante que nenhuma transação futura que já leu uma versão mais recente seja afetada por uma escrita retroativa.

Gerenciamento de Versões Antigas

O sistema precisa eventualmente reciclar versões antigas para evitar crescimento ilimitado do armazenamento:

- Uma versão Qk pode ser deletada se existe uma versão Qj com W-timestamp maior
- E não há transação ativa Ti tal que W-timestamp(Qk) ≤ TS(Ti) < W-timestamp(Qj)
- Políticas de garbage collection removem versões que nunca mais serão acessadas

Trade-off Importante: Embora o protocolo multiversão aumente significativamente a concorrência, ele requer espaço adicional para armazenar múltiplas versões e processamento para gerenciar e reciclar versões obsoletas.

Tratamento de Impasse

Impasses (deadlocks) ocorrem quando duas ou mais transações ficam eternamente esperando umas pelas outras em um ciclo de dependências. Este é um problema clássico em sistemas concorrentes que requer detecção e resolução cuidadosas.

Considere duas transações

T1	T2
write(x)	
	write(y)
	write(x)
write(y)	

Schedule com impasse

T1	T2
lock-X on X	
write(x)	
	lock-X on Y
	Write(Y)
	wait for lock-X on X
wait for lock-X on Y	

Consequências do Impasse

Sem intervenção do sistema, as transações permaneceriam bloqueadas permanentemente. O sistema deve detectar esta situação e tomar medidas corretivas abortando uma das transações.

T1: lock-X(y) → aguarda T2

T2: lock-X(x) → aguarda T1

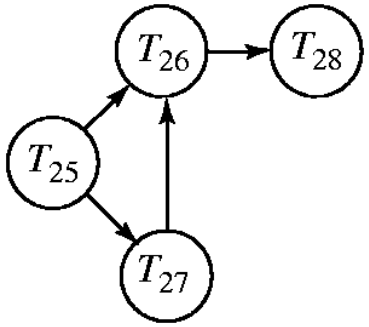
Resultado: Ciclo de espera

DETECÇÃO

Detecção de Impasse

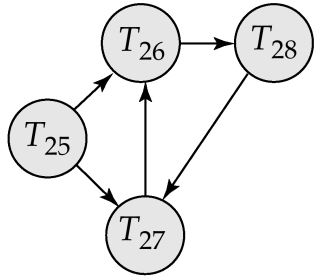
A detecção de impasses é realizada através da construção e análise de um grafo de espera (wait-for graph), onde os nós representam transações e as arestas representam relações de espera. Um ciclo neste grafo indica a presença de um impasse no sistema.

Grafo Sem Ciclo



Nesta configuração, as transações formam uma estrutura acíclica. Não há ciclos no grafo de espera, portanto não existe impasse. O sistema pode eventualmente completar todas as transações seguindo a ordem de dependências.

Grafo Com Ciclo



Aqui observamos um ciclo no grafo de espera, indicando claramente a presença de um impasse. As transações no ciclo estão mutuamente bloqueadas e não podem progredir sem intervenção externa do sistema.

Construção do Grafo

Para cada transação Ti aguardando por um recurso mantido por Tj, adiciona-se uma aresta Ti → Tj no grafo de espera. O grafo é continuamente atualizado conforme bloqueios são adquiridos e liberados.

Algoritmo de Detecção

O sistema periodicamente executa algoritmos de detecção de ciclos no grafo de espera. Quando um ciclo é detectado, identifica-se que as transações no ciclo estão em impasse e medidas de recuperação devem ser tomadas.

Frequência de Verificação

Trade-off entre overhead de detecção e tempo de resposta: verificações muito frequentes consomem recursos, mas verificações raras aumentam o tempo que transações ficam bloqueadas em impasse.

Recuperação de Impasse

Quando um impasse é detectado, o sistema deve intervir para quebrá-lo. A recuperação envolve decisões cuidadosas sobre qual transação abortar e até onde reverter sua execução para minimizar o custo total e evitar problemas como inanição.

Seleção da Vítima

Alguma transação deve ser escolhida para ser revertida (a vítima) e assim romper o ciclo de impasse. A seleção considera fatores de custo: tempo de computação investido, dados modificados, trabalho restante, número de transações afetadas em cascata, e número de rollbacks anteriores desta transação.

Estratégias de Rollback

Rollback Total: A forma mais simples é abortar completamente a transação e reiniciá-la do zero. Simples de implementar mas pode desperdiçar muito trabalho já realizado.

Rollback Parcial: Mais eficiente é reverter a transação apenas até o ponto necessário para romper o impasse, preservando parte do trabalho já executado e reduzindo o custo de recuperação.

Prevenção de Inanição

Inanição (starvation) ocorre quando a mesma transação é repetidamente escolhida como vítima, nunca conseguindo completar. Para evitar isto, o número de rollbacks sofridos deve ser incluído como fator no cálculo de custo, dando menor prioridade a transações que já foram vítimas múltiplas vezes.

Consideração Prática: A escolha da vítima deve buscar minimizar o custo total, mas também distribuir o impacto entre diferentes transações para garantir fairness e progresso contínuo de todas as transações no sistema.

Referências

Elmasri & Navathe

Fundamentals of Database Systems

Pearson, 2016

Referência abrangente sobre fundamentos de sistemas de bancos de dados, cobrindo aspectos teóricos e práticos.

Korth, Sudarshan & Silberschatz

Database System Concepts

McGraw-Hill, 2019

Texto fundamental que serviu como base para a maioria dos exemplos apresentados nesta apresentação.

Özsu & Valduriez

Principles of Distributed Database Systems

Springer Nature, 2019

Obra especializada em sistemas de bancos de dados distribuídos, essencial para compreensão avançada.

Nota: Os conceitos e exemplos apresentados baseiam-se principalmente na literatura clássica de sistemas de bancos de dados, em especial *Database System Concepts* e *Fundamentals of Database Systems*.