CEFET/RJ

BANCOS DE DADOS

Transações em Sistemas de Banco de Dados

Transações são fundamentais para garantir a integridade e consistência dos dados em sistemas de banco de dados modernos. Este material apresenta os conceitos essenciais sobre transações, suas propriedades e comportamento em ambientes concorrentes.

Eduardo Ogasawara

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

Conceito de Transação

Uma transação representa uma unidade lógica de trabalho que agrupa operações de leitura e escrita no banco de dados. É o mecanismo fundamental para manter a integridade dos dados em qualquer sistema gerenciador de banco de dados (SGBD).

Unidade de Execução

Uma transação é uma unidade da execução de programa que acessa e possivelmente atualiza vários itens de dados de forma coordenada e controlada.

Consistência Temporária

A transação precisa encontrar um banco de dados consistente no início. Durante sua execução, o banco pode ficar temporariamente inconsistente, mas ao final deve retornar a um estado consistente.

Persistência das Mudanças

Após a confirmação (commit) da transação, as mudanças que ela faz no banco de dados persistem permanentemente, mesmo diante de falhas de sistema.

Desafios Principais

- Gerenciamento de falhas de vários tipos, incluindo falhas de hardware e software
- Controle da execução simultânea de múltiplas transações concorrentes

Tipos de Falhas em Sistemas de Banco de Dados

Os sistemas de banco de dados estão sujeitos a diversos tipos de falhas que podem comprometer a integridade dos dados. O SGBD deve estar preparado para lidar com cada uma delas, garantindo que as propriedades ACID sejam mantidas mesmo em situações adversas.

Falhas de Transação

- **Erros lógicos:** problemas na lógica da aplicação que impedem a conclusão correta
- **Violação de restrições:** tentativas de violar regras de integridade do banco
- **Abort explícito:** cancelamento intencional por parte da aplicação

Falhas de Sistema

- **Queda de energia:** perda súbita de alimentação elétrica do servidor
- **Falha do sistema operacional:** crashes ou travamentos do SO que interrompem operações

Falhas de Mídia

- **Falha de disco:** problemas físicos no hardware de armazenamento
- **Perda permanente:** destruição irrecuperável de dados armazenados

Importante: O SGBD deve garantir as propriedades ACID (Atomicidade, Consistência, Isolamento e Durabilidade) mesmo na presença de qualquer um desses tipos de falhas.

Propriedades ACID

As propriedades ACID formam a base fundamental para o funcionamento correto de transações em bancos de dados. Cada letra representa uma garantia essencial que o sistema deve fornecer.

Cada propriedade ACID é garantida por componentes específicos do SGBD, trabalhando de forma integrada para manter a integridade do sistema. Compreender quem é responsável por cada propriedade ajuda a entender a arquitetura interna dos bancos de dados.

Atomicidade

Garantida pelo **mecanismo de recuperação**, que assegura que todas as operações de uma transação sejam completadas ou nenhuma seja aplicada.

Consistência

Responsabilidade **conjunta da aplicação e do SGBD**. A aplicação deve definir transações corretas, e o SGBD deve verificar restrições de integridade.

Isolamento

Garantido pelos **protocolos de controle de concorrência**, que coordenam o acesso simultâneo aos dados por múltiplas transações.

Durabilidade

Garantida pelo **armazenamento estável** e mecanismos de log, que preservam mudanças confirmadas mesmo após falhas de sistema.

Exemplo de Transferência de Fundos

Considere uma transação bancária que transfere US$ 50 da conta A para a conta B. Este exemplo clássico ilustra como as propriedades ACID se aplicam na prática.

Sequência de Operações:

1. read(A) - lê o saldo da conta A
2. A := A – 50 - subtrai o valor
3. write(A) - grava o novo saldo de A
4. read(B) - lê o saldo da conta B
5. B := B + 50 - adiciona o valor
6. write(B) - grava o novo saldo de B

Requisito de Atomicidade

Se a transação falhar após a etapa 3 e antes da etapa 6, o sistema deve garantir que suas atualizações não sejam refletidas no banco de dados. Caso contrário, o dinheiro seria retirado de A sem ser creditado em B, causando uma inconsistência grave.

Requisito de Consistência

A soma de A e B deve permanecer **inalterada** pela execução da transação. O invariante "saldo_total = A + B" é preservado do início ao fim da operação.

Requisito de Isolamento

Se entre as etapas 3 e 6, outra transação receber permissão de acessar o banco de dados parcialmente atualizado, ela verá um banco de dados inconsistente - a soma A + B será menor do que deveria ser.

Isso pode ser trivialmente assegurado **executando transações serialmente**, ou seja, uma após outra. Entretanto, executar múltiplas transações simultaneamente oferece vantagens significativas em termos de desempenho e utilização de recursos, como veremos mais adiante.

Requisito de Durabilidade

Quando o usuário é notificado de que a transação está concluída (ou seja, a transferência dos US$ 50 ocorreu com sucesso), as atualizações no banco de dados pela transação precisam **persistir permanentemente**, apesar de quaisquer falhas subsequentes de hardware ou software.

Isso é especialmente crítico em sistemas bancários, onde a perda de dados de transações confirmadas seria inaceitável.

Estados da Transação

Durante seu ciclo de vida, uma transação passa por diversos estados bem definidos. Compreender esses estados é fundamental para entender como o SGBD gerencia transações e garante suas propriedades.

Ativa

O estado inicial de toda transação. A transação permanece nesse estado enquanto suas instruções estão sendo executadas normalmente.

Parcialmente Confirmada

Estado atingido depois que a instrução final foi executada. A transação completou suas operações mas ainda não teve suas mudanças tornadas permanentes.

Confirmada

Estado final de sucesso, atingido após o término bem-sucedido da transação. Todas as mudanças são tornadas permanentes no banco de dados.

Falha

Estado atingido após a descoberta de que a execução normal não pode mais prosseguir devido a algum erro ou violação.

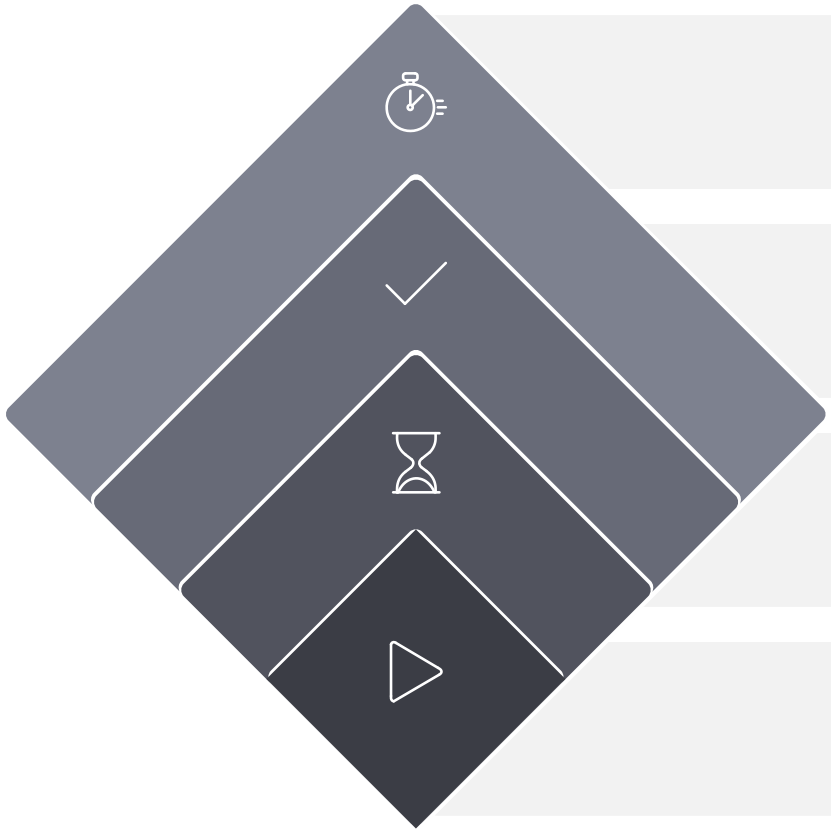
Abortada

Estado após a transação ter sido revertida e o banco de dados restaurado ao seu estado anterior. Duas opções estão disponíveis:

- **Reiniciar a transação:** possível apenas se não houver erro lógico interno
- **Excluir a transação:** descartar permanentemente a operação

Diagrama de Estados da Transação

O diagrama abaixo ilustra as transições possíveis entre os diferentes estados de uma transação, mostrando os caminhos de sucesso e falha.



Falha

Erro detectado

Confirmada

Sucesso final

Parcialmente Confirmada

Confirmação parcial

Ativa

Estado inicial

Caminho de Sucesso

Ativa → Parcialmente Confirmada → Confirmada

Caminho de Falha

Ativa/Parcialmente Confirmada → Falha → Abortada

Execuções Simultâneas

A execução concorrente de transações é um dos pilares fundamentais dos sistemas de banco de dados modernos. Permite que múltiplas transações sejam processadas em paralelo, maximizando a utilização dos recursos do sistema.

Vantagens da Concorrência

Melhor Utilização de Recursos

Uma transação pode estar usando a CPU enquanto outra está lendo ou escrevendo no disco, resultando em **melhor throughput** de transações e uso mais eficiente do hardware.

Tempo de Resposta Reduzido

As transações curtas não precisam esperar atrás das transações longas. Isso melhora significativamente a experiência do usuário em sistemas interativos.

Esquemas de Controle de Concorrência

São mecanismos essenciais para obter **isolamento** entre transações concorrentes. Eles controlam a interação entre as transações simultâneas a fim de evitar que elas destruam a consistência do banco de dados.

Escalonamento (Schedule)

Um escalonamento define a ordem de execução das operações de múltiplas transações concorrentes. É o conceito central para analisar a corretude da execução simultânea de transações.

Definição

Um escalonamento é uma sequência de instruções que especifica a **ordem cronológica** em que as instruções das transações simultâneas são executadas pelo sistema.

Requisitos Estruturais

- Precisa conter **todas as instruções** de cada transação no conjunto
- Deve preservar a ordem em que as instruções aparecem em cada transação individual

Instrução de Commit

Uma transação que completa com sucesso sua execução terá uma instrução de **commit** como a última instrução (pode ser omitida quando óbvia do contexto).

Instrução de Abort

Uma transação que não completa com sucesso sua execução terá instrução de **abort** como a última instrução (também pode ser omitida quando óbvia).

Modelo Abstrato de Operações

Para análise de corretude de escalonamentos, utilizamos um modelo simplificado que considera apenas operações essenciais de acesso aos dados. Esta abstração facilita o estudo teórico sem perder generalidade.

Operações Consideradas

read(X)

Operação de leitura do item de dado X do banco de dados para um buffer local da transação.

write(X)

Operação de escrita do item de dado X do buffer local da transação para o banco de dados.

Todos os cálculos internos da transação são feitos em **buffers locais** e não são considerados na análise.

Por Que Esse Modelo?

Apenas operações de leitura e escrita:

- Afetam diretamente o estado do banco de dados
- Podem gerar **conflitos** entre transações concorrentes
- São suficientes para estudar serializabilidade e recuperabilidade

Este modelo simplificado é suficiente para estudar os conceitos fundamentais de **serializabilidade** e **recuperabilidade** de escalonamentos.

Schedule 1: Execução Serial

Tempo	T_1	T_2
1	read(A)	
2	A := A - 50	
3	write(A)	
4	read(B)	
5	B := B + 50	
6	write(B)	
7		read(A)
8		A := A * 1.1
9		write(A)
10		read(B)
11		B := B * 1.1
12		write(B)

Este exemplo mostra um escalonamento serial simples, onde as transações T_1 e T_2 executam uma após a outra, sem sobreposição. Escalonamentos seriais são sempre corretos, pois evitam completamente problemas de concorrência.

No Schedule 1, T_1 executa completamente primeiro, seguida por T_2. Esta é a execução mais simples e sempre preserva a consistência, mas não aproveita os benefícios da concorrência.

Propriedade Chave: Escalonamentos seriais garantem consistência por definição, assumindo que cada transação preserva individualmente a consistência do banco de dados.

Schedule 2: Execução Concorrente

Tempo	T_1	T_2
1	read(A)	
2	A := A - 50	
3	write(A)	
4		read(A)
5		A := A * 1.1
6		write(A)
7	read(B)	
8	B := B + 50	
9	write(B)	
10		read(B)
11		B := B * 1.1
12		write(B)

Este escalonamento ilustra a execução concorrente de T_1 e T_2, onde as operações das duas transações se intercalam. O objetivo é verificar se esse entrelaçamento preserva a consistência.

Neste escalonamento concorrente, as operações se intercalam de forma controlada. A questão crucial é: **este escalonamento é equivalente a algum escalonamento serial?**

Schedule 3: Concorrência Problemática

Este exemplo demonstra um escalonamento concorrente que pode levar a resultados incorretos. É fundamental identificar quando a intercalação de operações viola a consistência do banco de dados.

Tempo	T_1	T_2
1	read(A)	
2	A := A - 50	
3		read(A)
4		A := A * 1.1
5	write(A)	
6		write(A)
7	read(B)	
8	B := B + 50	
9		read(B)
10		B := B * 1.1
11	write(B)	
12		write(B)

Problema Identificado

Neste escalonamento, T_2 lê o valor de A **antes** de T_1 escrever, mas escreve **depois** de T_1. Isso causa uma perda de atualização.

O resultado final pode ser diferente de qualquer execução serial possível, violando a consistência do banco de dados.

Este é um exemplo clássico de **anomalia de concorrência** que os protocolos de controle devem prevenir.

Serializabilidade

Serializabilidade é o critério fundamental para determinar se um escalonamento concorrente é correto. Um escalonamento é considerado correto se produz o mesmo resultado que alguma execução serial das mesmas transações.

Suposição Básica

Cada transação individual preserva a consistência do banco de dados quando executada isoladamente. Portanto, a **execução serial** de um conjunto de transações sempre preserva a consistência.

Definição de Serializabilidade

Um escalonamento (possivelmente simultâneo) é **seriável** se for equivalente a algum escalonamento serial das mesmas transações.

Tipos de Equivalência

Diferentes formas de equivalência de escalonamento geram diferentes noções de serializabilidade:

1

Seriação de Conflito

Baseada na ordem de operações conflitantes entre transações. É o tipo mais comum e prático de serializabilidade.

2

Seriação de Visão (View)

Baseada nos valores lidos e escritos pelas transações. Mais abrangente teoricamente, mas menos prática de verificar.

Para análise de serializabilidade, ignoramos operações exceto `read` e `write`, e consideramos que as transações podem realizar cálculos arbitrários sobre dados em buffers locais.

Instruções Conflitantes

O conceito de conflito entre instruções é fundamental para entender serializabilidade de conflito. Duas instruções conflitam quando sua ordem de execução pode afetar o resultado final.

Definição de Conflito

Duas instruções de transações diferentes **conflitam** se e somente se:

Acessam o Mesmo Item

Ambas as instruções operam sobre o mesmo item de dado X no banco de dados.

Pelo Menos Uma é Write

Ao menos uma das instruções é uma operação de escrita (write). Isso garante que a ordem importa para o resultado.

Tipos de Conflito

- **Read-Write:** Uma transação lê enquanto outra escreve o mesmo dado
- **Write-Read:** Uma transação escreve enquanto outra lê o mesmo dado
- **Write-Write:** Duas transações escrevem no mesmo dado

Não-Conflito

- **Read-Read:** Duas leituras do mesmo dado nunca conflitam
- **Dados diferentes:** Operações em dados distintos nunca conflitam

Operações que **não conflitam** podem ter sua ordem trocada sem afetar o resultado do escalonamento.

Serializabilidade de Conflito

A serializabilidade de conflito é o critério mais utilizado na prática para verificar a corretude de escalonamentos concorrentes. Ela se baseia na possibilidade de reordenar operações não-conflitantes.

Transformação

Trocar operações não-conflitantes de posição no escalonamento

Equivalência

Chegar a um escalonamento serial através dessas trocas

Serializável

O escalonamento original é seriável por conflito

Definição Formal

Um escalonamento S é **seriável por conflito** se puder ser transformado em um escalonamento serial através de uma série de trocas de instruções **não-conflitantes** consecutivas.

Equivalência por Conflito

Dois escalonamentos são **equivalentes por conflito** se:

- Envolvem as mesmas transações
- Todo par de instruções conflitantes aparece na mesma ordem em ambos

Serializabilidade de Conflito: Exemplo Prático

Escalonamento Original S

Passo	T_1	T_2	T_3
1	read(X)		
2		read(Y)	
3			read(X)
4	write(X)		
5		write(Y)	
6			write(X)

Vamos analisar um exemplo concreto para verificar se um escalonamento é seriável por conflito, aplicando o método de transformação por meio de trocas de operações não-conflitantes.

01

Identificar Conflitos

T_1.read(X) conflita com T_3.write(X), e T_1.write(X) conflita com T_3.read(X) e T_3.write(X).

02

Buscar Trocas Válidas

Operações de T_2 não conflitam com T_1 e T_3 pois operam em dado diferente (Y).

03

Transformar em Serial

Mover operações não-conflitantes para obter ordem serial: $T_1 \rightarrow T_3 \rightarrow T_2$ ou similar.

Serializabilidade de Conflito: Verificação

A verificação de serializabilidade de conflito pode ser realizada através da construção de um **grafo de precedência** (ou grafo de seriação). Este método gráfico facilita a identificação de ciclos que impedem a serializabilidade.

Construção do Grafo de Precedência

- ### Vértices

  Cada transação Ti no escalonamento corresponde a um vértice no grafo.

- ### Arestas

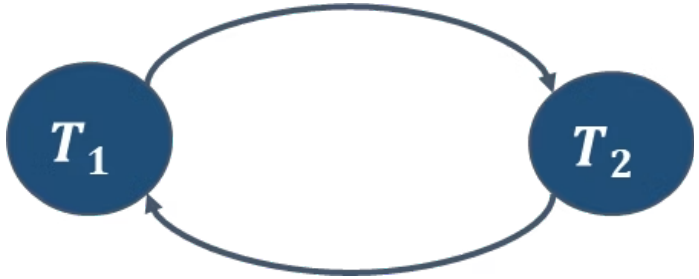
  Desenhe uma aresta Ti → Tj se existir uma operação conflitante onde Ti vem antes de Tj no escalonamento.

- ### Teste de Ciclos

  Se o grafo contiver um ciclo, o escalonamento **NÃO é seriável** por conflito.

- ### Ordem Serial

  Se o grafo for acíclico, qualquer **ordenação topológica** do grafo produz uma ordem serial equivalente.



Complexidade: A verificação de serializabilidade pode ser realizada em tempo polinomial usando algoritmos de detecção de ciclos.

Importância Prática

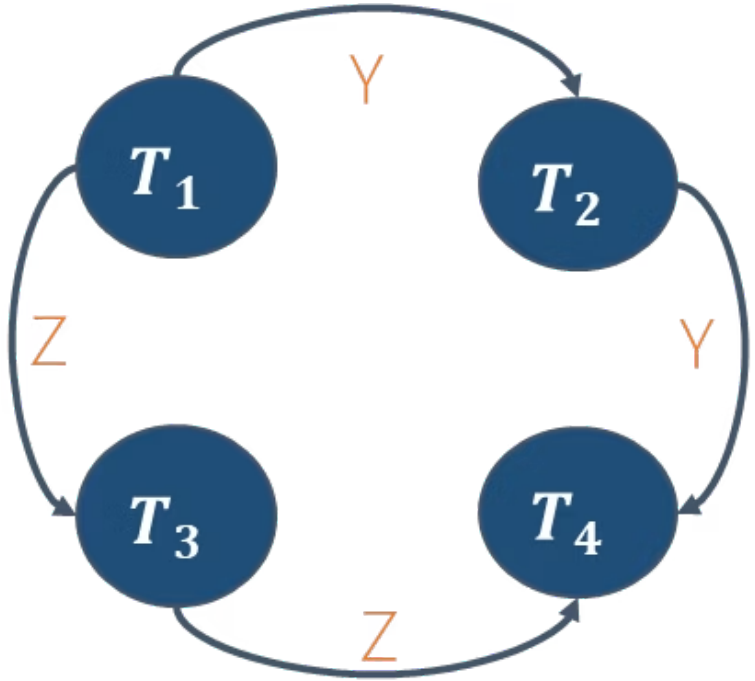
Os SGBDs modernos utilizam protocolos de controle de concorrência (como bloqueios de duas fases) que **garantem** serializabilidade de conflito sem precisar verificar o grafo explicitamente. Isso permite operação eficiente mesmo com milhares de transações concorrentes.

CONCEITOS FUNDAMENTAIS

Exemplo de Escalonamento

T_1	T_2	T_3	T_4	T_5
	read(X)			
read(Y) read(Z)				
				read(V) read(W) read(W)
	read(Y) write(Y)			
		write(Z)		
read(U)				
			read(Y) write(Y) read(Z) write(Z)	
read(U) write(U)				

Grafo de precedência



O escalonamento correto garante que, mesmo com execução concorrente, o resultado seja equivalente a alguma execução serial das transações. Este exemplo ilustra a importância de analisar cuidadosamente a ordem das operações concorrentes.

Teste para Seriação de Conflito

A seriação de conflito é uma das formas mais importantes de verificar se um escalonamento de transações é correto. Um escalonamento é serializável por conflito se puder ser transformado em um escalonamento serial através de uma série de trocas de operações não conflitantes.

Conflitos de Leitura-Escrita

Ocorrem quando uma transação lê um dado que outra transação está modificando. A ordem dessas operações é crítica para manter a consistência.

Conflitos de Escrita-Escrita

Acontecem quando duas transações tentam escrever no mesmo dado. O resultado final depende de qual escrita é executada por último.

Grafo de Precedência

Ferramenta fundamental para detectar conflitos. Se o grafo contém ciclos, o escalonamento não é serializável por conflito.

VISÃO AMPLIADA

Seriação de Visão

A seriação de visão oferece uma abordagem mais flexível que a seriação de conflito. Um escalonamento é serializável por visão se produz o mesmo resultado que algum escalonamento serial, considerando três aspectos fundamentais.

Primeiro, se uma transação lê o valor inicial de um dado no escalonamento original, ela deve ler o mesmo valor inicial no escalonamento serial equivalente. Segundo, se uma transação lê um valor escrito por outra transação, essa relação de leitura deve ser preservada.

Finalmente, a transação que realiza a última escrita em cada dado deve ser a mesma em ambos os escalonamentos. Esta abordagem permite maior concorrência que a seriação de conflito.

01

Análise de Leituras Iniciais

Verificar se cada transação que lê o valor inicial de um item no escalonamento S também lê o valor inicial desse item em algum escalonamento serial S'.

02

Preservação de Dependências de Leitura

Garantir que se uma transação T lê um valor produzido por uma transação U em S, então T também deve ler o valor produzido por U em S'.

03

Consistência de Escritas Finais

Assegurar que a transação que realiza a escrita final de cada item de dado seja a mesma tanto em S quanto em S'.

Esses três critérios trabalham em conjunto para determinar se um escalonamento mantém a equivalência de visão com alguma execução serial, permitindo maior flexibilidade na ordenação de transações concorrentes.

Teste para Seriação de Visão

Complexidade Computacional

O problema de verificar se um schedule é serial de visão pertence à classe dos problemas não procedurais completos (NP-hard). Isso significa que não existe algoritmo conhecido que possa resolver o problema de forma eficiente para todos os casos possíveis.

A complexidade NP-hard implica que, para escalonamentos grandes, o tempo necessário para verificar a seriação de visão pode crescer exponencialmente com o número de transações.

Abordagens Práticas

Embora a existência de um algoritmo eficiente seja improvável, os algoritmos práticos que simplesmente verificam algumas condições suficientes para a seriação de visão ainda podem ser usados em sistemas reais.

Essas abordagens práticas sacrificam completude por eficiência, rejeitando alguns escalonamentos que poderiam ser serializáveis, mas garantindo que todos os escalonamentos aceitos sejam de fato serializáveis.

Outras Noções de Seriação

Além da seriação de conflito e da seriação de visão, existem outras noções de correção em sistemas de bancos de dados concorrentes. Cada abordagem oferece diferentes garantias e trade-offs entre desempenho e consistência.

A seriação estrita adiciona requisitos temporais, garantindo que as leituras reflitam apenas valores de transações já confirmadas. A seriação de snapshot permite que transações operem em versões consistentes do banco de dados, isolando-as de mudanças concorrentes.

Seriação Estrita

Impõe ordem temporal nas operações, evitando leituras sujas e escritas sujas através de restrições de bloqueio mais rigorosas.

Seriação de Snapshot

Permite que cada transação trabalhe com uma visão consistente do banco de dados, capturada no momento de seu início.

Seriação por Timestamp

Utiliza marcas temporais para ordenar transações, garantindo execução consistente baseada em ordem cronológica.

Facilidade de Recuperação

A recuperabilidade é uma propriedade fundamental dos escalonamentos que garante a capacidade de desfazer transações em caso de falhas. Um escalonamento é recuperável se nenhuma transação T confirma suas mudanças até que todas as transações que escreveram valores lidos por T tenham sido confirmadas.

Leitura de Dados

Transação T lê valor escrito por U

Commit de U

U deve confirmar antes de T

Commit de T

T pode confirmar com segurança

Esta propriedade evita situações onde uma transação confirmada precisa ser desfeita porque leu dados de uma transação que foi posteriormente abortada. Escalonamentos não recuperáveis podem levar a estados inconsistentes do banco de dados que são impossíveis de corrigir.

Facilidade de Recuperação: Conceitos Avançados

Existem diferentes níveis de recuperabilidade que oferecem garantias progressivamente mais fortes sobre a capacidade de recuperação do sistema.

Um escalonamento sem cascata (cascadeless) evita rollbacks em cascata, garantindo que transações só leiam valores de transações já confirmadas. Esta propriedade é mais forte que a simples recuperabilidade.

Escalonamentos estritos vão ainda mais longe, exigindo que transações não leiam nem escrevam dados modificados por transações não confirmadas. Isso simplifica significativamente o processo de recuperação.

1

Recuperável

Nível básico: transações confirmam na ordem correta de dependência

2

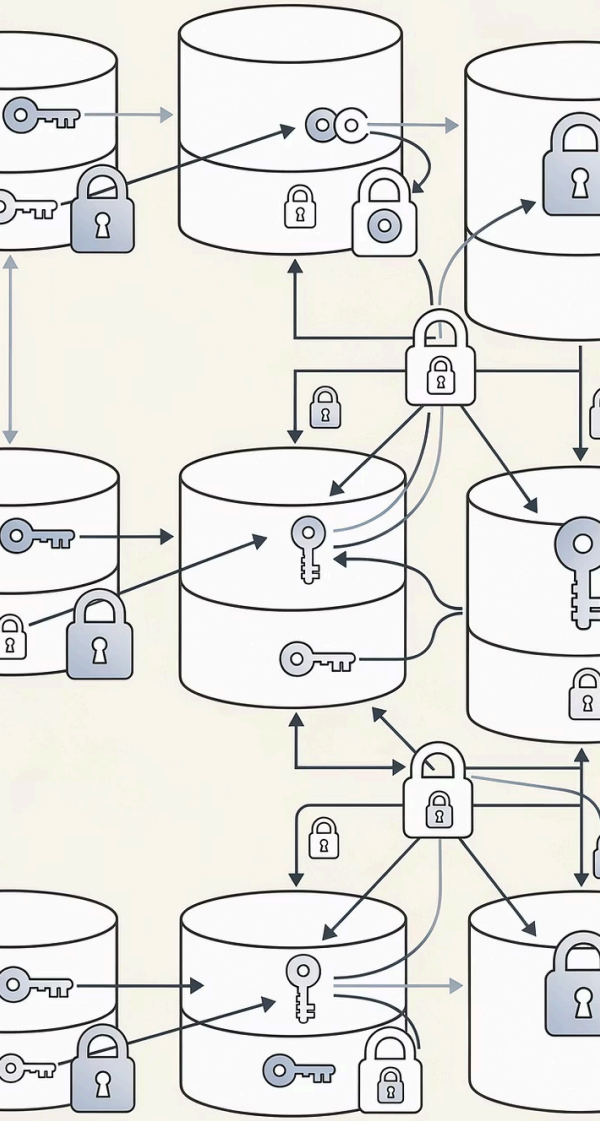
Sem Cascata

Nível intermediário: elimina necessidade de rollbacks em cascata

3

Estrito

Nível máximo: previne leituras e escritas de dados não confirmadas



CONTROLE DE CONCORRÊNCIA

Testes de Controle e de Seriação

> **Desafio Fundamental:** Testar a seriação de um schedule após ele ter sido executado é um pouco tarde demais...

Objetivo Principal

Desenvolver protocolos de controle de concorrência que garantam a capacidade de seriação durante a execução, não após.

Protocolos Preventivos

Os protocolos normalmente não examinam o grafo de precedência enquanto está sendo criado; em vez disso, impõem regras que evitam schedules não serializáveis.

Validação Teórica

Os testes para seriação ajudam a entender por que um protocolo de controle de concorrência está correto e fornecem base teórica sólida.

Definição de Transação na SQL

A linguagem de manipulação de dados precisa incluir uma construção para especificar o conjunto de ações que compõem uma transação. Na SQL, uma transação começa implicitamente quando a primeira instrução é executada.

O término de uma transação é explícito e pode ocorrer de duas formas principais:

- **Commit work:** confirma a transação atual, tornando todas as mudanças permanentes, e inicia uma nova transação
- **Rollback work:** faz com que a transação atual seja abortada, descartando todas as mudanças realizadas

1

Serializable

Nível padrão, garante isolamento completo

2

Repeatable Read

Impede leituras não repetíveis

3

Read Committed

Permite apenas dados confirmados

4

Read Uncommitted

Nível mais permissivo

Níveis de Consistência na SQL-92

Serializable (Padrão)

O nível mais alto de isolamento. Garante que a execução concorrente de transações produza resultados idênticos a alguma execução serial. Oferece proteção completa contra todos os fenômenos de inconsistência.

Repeatable Read

Apenas registros confirmados podem ser lidos, e reads repetidos do mesmo registro precisam retornar o mesmo valor. Entretanto, uma transação pode não ser serializável - ela pode encontrar alguns registros inseridos por uma transação mas não encontrar outros (phantom reads).

Read Committed

Apenas registros confirmados podem ser lidos, mas reads sucessivos do mesmo registro podem retornar valores diferentes (mas sempre confirmados). Previne leituras sujas, mas permite leituras não repetíveis e phantom reads.

Read Uncommitted

O nível mais permissivo - mesmo registros não confirmados podem ser lidos. Graus de consistência mais baixos são úteis para coletar informações aproximadas sobre o banco de dados, como estatísticas para otimização de consultas.

CONSIDERAÇÕES PRÁTICAS

Observações sobre os Níveis de Isolamento SQL

Serializable: Máxima Segurança

Apenas o nível Serializable garante serializabilidade por conflito, oferecendo a mais forte proteção contra anomalias de concorrência.

Níveis Mais Baixos: Maior Concorrência

Níveis inferiores permitem maior concorrência e melhor desempenho, mas não garantem serializabilidade completa.

Implementações Reais

Implementações de SGBDs podem não seguir exatamente a especificação SQL-92, com variações entre fornecedores.

Trade-off Fundamental

A escolha do nível de isolamento envolve um compromisso crítico entre correção e desempenho. Aplicações que exigem consistência absoluta devem usar Serializable, enquanto aplicações analíticas podem se beneficiar de níveis mais baixos.

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*.