Unidade V

Controle de Concorrência

- Protocolos de Bloqueio
- Protocolo com base em Timestamps
- Protocolo de Validação
- Protocolos Multi-versão
- Controle de Deadlock
- Inserção e Remoção de Dados
- Concorrência em Índices

Controle de Concorrência

- É necessário controlar a concorrência para garantir o isolamento entre transações
- Protocolos usados para controlar a concorrência
  - Protocolos de bloqueio: controlam o acesso ao BD, bloqueando os dados que estão sendo usados
  - Protocolos com base em timestamps: ordenam as operações com base no tempo de início da transação
  - Protocolos de validação: efetuam testes de validação dos dados antes das operações de escrita
- Protocolos de controle garantem a serialização no processamento de transações

Protocolos de Bloqueio

- Impedem que um dado seja modificado enquanto uma transação o estiver acessando
- Modos de bloqueio
  - Compartilhado (S): permite somente leitura; obtido sempre que não houver nenhum bloqueio exclusivo
  - Exclusivo (X): permite leitura e escrita; obtido somente se não houver nenhum outro bloqueio
- Todas as transações devem:
  - Solicitar o bloqueio compartilhado (para leitura) ou exclusivo (para escrita) antes de acessar um dado
    - Autorização vai depender dos bloqueios existentes
  - Liberar o bloqueio quando não for mais necessário

Protocolos de Bloqueio

- Operações de bloqueio

Operação	Ação
lock-S(dado)	Solicita bloqueio compartilhado do dado
lock-X(dado)	Solicita bloqueio exclusivo do dado
unlock(dado)	Libera o bloqueio existente

- Resultado das operações

Operação	Estado: Livre	Estado: S	Estado: X
lock-S(dado)	S; n=1	S; n++	Espera
lock-X(dado)	X	Espera	Espera
unlock(dado)	Ignora	**Se n=1: Livre Se n≠1: S; n--**	Livre

- Transações em espera ganham direito de acesso quando o dado bloqueado for liberado

Concorrência

- Exemplo: suponha as transações T_1 e T_2
  - T_1: Read (Aplic);
    Aplic.Saldo = Aplic.Saldo – 500;
    Write (Aplic);
    Read (Conta);
    Conta.Saldo = Conta.Saldo + 500;
    Write (Conta);
  - T_2: Read (Conta);
    Read (Aplic);
    Print (Conta. Saldo + Aplic.Saldo);
  - Operações de bloqueio devem ser adicionadas ao código para garantir o isolamento entre as transações

Protocolos de Bloqueio

- Bloqueio só no acesso não garante isolamento

T_1	T_2
Lock-X (Aplic); Read (Aplic); Aplic.Saldo = Aplic.Saldo – 500; Write (Aplic); Unlock (Aplic);	
Bloqueada	Lock-S (Conta); Read (Conta); Unlock (Conta);
Lock-X (Conta); Read (Conta); Conta.Saldo = Conta.Saldo + 500; Write (Conta); Unlock (Conta);	Bloqueada
	Lock-S (Aplic); Read (Aplic); Unlock (Aplic); Print (Conta.Saldo + Aplic.Saldo);

Protocolos de Bloqueio

- Protocolo de bloqueio em duas fases
  - Primeira fase: Fase de expansão
    - Pode bloquear dados
    - Não pode liberar os bloqueios obtidos
  - Segunda fase: Fase de recolhimento
    - Pode liberar os dados bloqueados anteriormente
    - Não pode mais bloquear dados
  - Garante escala de execução serializável em conflito
  - Precedência é determinada em função do instante de obtenção do último bloqueio
  - Não evita deadlock e rollback em cascata (ver slide 8)

Protocolos de Bloqueio

- Exemplo de bloqueio em duas fases

T_1	T_2
Lock-X (Aplic); Read (Aplic); Aplic.Saldo = Aplic.Saldo – 500; Write (Aplic); Lock-X (Conta); Read (Conta);	
Bloqueada	Lock-S (Conta); // Bloqueio
Conta.Saldo = Conta.Saldo + 500; Write (Conta); Unlock (Conta); Unlock (Aplic);	Bloqueada
	Read (Conta); Lock-S (Aplic); Read (Aplic); Print (Conta.Saldo + Aplic.Saldo); Unlock (Conta); Unlock (Aplic);

Protocolos de Bloqueio

- Bloqueio em duas fases pode causar deadlock

T_1	T_2
Lock-X (Aplic); Read (Aplic); Aplic.Saldo = Aplic.Saldo – 500; Write (Aplic);	
Bloqueada	Lock-S (Conta); Read (Conta); Lock-S (Aplic);
Lock-X (Conta);	Bloqueada
Bloqueada	Bloqueada
Não executa: Read (Conta); Conta.Saldo = Conta.Saldo + 500; Write (Conta); Unlock (Aplic); Unlock (Conta);	**Não executa:** Read (Aplic); Print (Conta.Saldo + Aplic.Saldo); Unlock (Conta); Unlock (Aplic);

Protocolos de Bloqueio

- Variantes do bloqueio em duas fases
  - Evitam o rollback em cascata
  - Usados na maioria dos SBDs
  - Protocolo de bloqueio em duas fases severo
    - Obriga que os bloqueios exclusivos sejam mantidos até a efetivação da transação
  - Protocolo de bloqueio em duas fases rigoroso
    - Obriga que todos os bloqueios (compartilhados e exclusivos) sejam mantidos até o commit
- Protocolos com conversão de bloqueios
  - Upgrade(Q): bloqueio compartilhado → exclusivo
  - Downgrade(Q): bloqueio exclusivo → compartilhado

Protocolos de Bloqueio

- Bloqueio pode causar inanição (starvation)

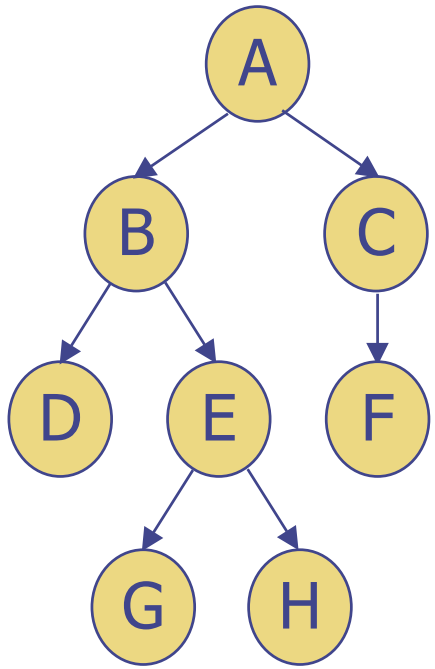
T_1	T_2	T_3	T_4	T_5
lock-S(Q)				
	lock-X(Q)			
	bloqueada	lock-S(Q)		
	bloqueada		lock-S(Q)	
	bloqueada			lock-S(Q)

  - T_2 nunca recebe o direiro de acesso!
  - Pode ser evitado fazendo que o direito de acesso compartilhado não seja concedido se houver uma transação esperando por bloqueio exclusivo

Protocolos de Bloqueio

- Protocolos de bloqueio baseados em gráficos
  - Determinam uma ordenação parcial no acesso aos dados usando um gráfico de precedência
  - Supondo que exista a relação de precedência A→B no gráfico, qualquer transação que acesse A e B, deve acessar primeiro A e depois B
  - A ordenação pode ser baseada na organização física ou lógica dos dados, ou pode ser imposta de modo aleatório somente para controlar a concorrência
  - Existem vários protocolos baseados em gráficos; um dos mais simples é o protocolo de gráfico em árvore

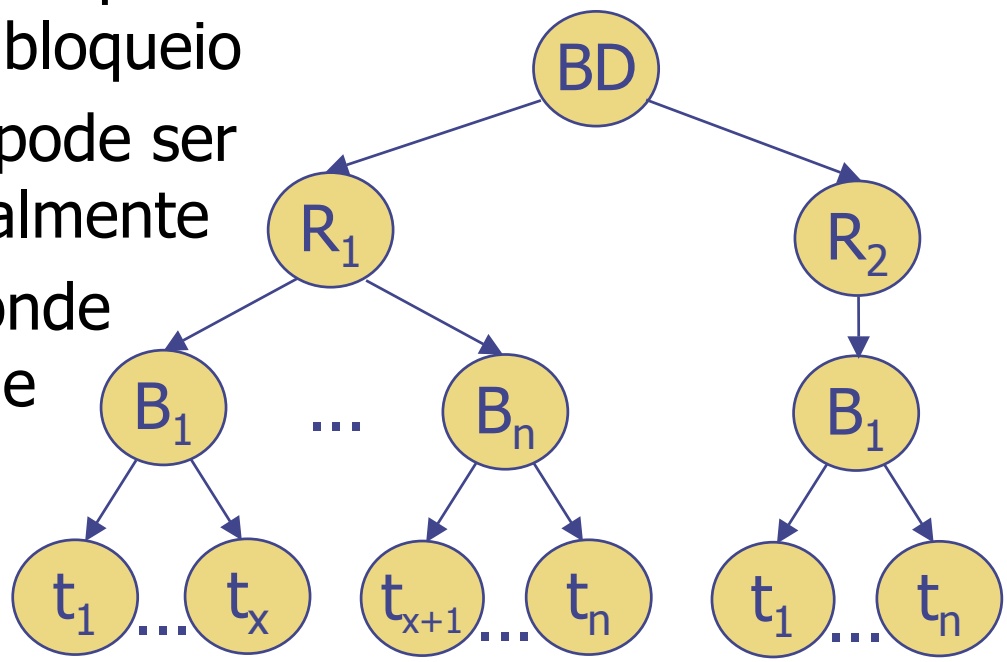
Protocolos de Bloqueio



- Protocolo de gráfico em árvore
  - Bloqueios são todos exclusivos
  - Primeiro bloqueio pode ser em qualquer dado
  - Os bloqueios seguintes podem ocorrer somente se o dado precedente estiver bloqueado
  - Dados podem ser desbloqueados a qualquer instante
  - Bloqueio mais curto que no protocolo de 2 fases
  - Garante serialização de conflito e evita deadlock
  - Pode bloquear mais dados do que reamente precisa
  - Escalas serializáveis por este protocolo podem não o ser com bloqueio em 2 fases, e vice-versa

Protocolos de Bloqueio

- Granularidade Múltipla
  - Ao invés de bloquear um item de dados, podemos bloquear tuplas, tabelas, blocos de disco ou BDs
  - Podemos usar árvores para escolher a granularidade do bloqueio
  - Cada nó da árvore pode ser bloqueado individualmente
  - Cada nível corresponde a uma granularidade de bloqueio



Protocolos de Bloqueio

- Modos de bloqueio intencional em árvores propiciam maior paralelismo entre transações
  - Intenção de compartilhamento (IS): indica bloqueio compartilhado sendo feito mais abaixo na árvore
  - Intenção de exclusividade (IX): indica bloqueio exclusivo sendo feito mais abaixo na árvore
  - Compartilhado com intenção de exclusividade (SIX): indica que há bloqueio compartilhado do ramo, com bloqueio exclusivo sendo feito mais abaixo na árvore

	IS	IX	S	SIX	X
IS	true	true	true	true	false
IX	true	true	false	false	false
S	true	false	true	false	false
SIX	true	false	false	false	false
X	false	false	false	false	false

Protocolo com base em Timestamps

- Associam timestamps (marcas de tempo) às transações e aos dados para ordenar operações de leitura e escrita
  - Cada transação T ganha um timestamp TS ao iniciar, com base no relógio do sistema ou em um contador
  - Dois timestamps são associados a cada dado
    - W-TS: indica a transação de maior timestamp que alterou o valor do dado
    - R-TS: indica a transação e maior timestamp que leu o valor do dado

Protocolo com base em Timestamps

- Ordenação de operações por timestamps
  - No caso da transação T querer ler um dado Q
    - Se TS(T) < W-TS(Q): T é abortada e desfeita (T quer ler um dado que já foi sobrescrito)
    - Se TS(T) ≥ W-TS(Q): a operação é executada; se R-TS(Q) < TS(T), então R-TS(Q) = TS(T)
  - No caso da transação T querer alterar um dado Q
    - Se TS(T) < R-TS(Q): T é abortada e desfeita (T não pode alterar um valor que já foi lido)
    - Se TS(T) < W-TS(Q): T é abortada e desfeita (T está tentando escrever um valor obsoleto)
    - Senão, a operação é executada; W-TS(Q) = TS(T)

Protocolo com base em Timestamps

- Exemplo de escala ordenada por timestamps

T_1	T_2	Timestamps
Read (Aplic); Aplic.Saldo –= 500 Write (Aplic);		$TS(T_1) = 1$ R-TS(Aplic) = 1 W-TS(Aplic) = 1
Bloqueada	Read (Conta); Read (Aplic); Print (Conta.Saldo + Aplic.Saldo);	$TS(T_2) = 2$ R-TS(Conta) = 2 R-TS(Aplic) = 2
Read (Conta); Conta.Saldo += 500; Write (Conta);		R-TS(Conta) = 2 $TS(T_1) <$ R-TS(Conta) $\rightarrow T_1$ deve ser refeita
Read (Aplic); Aplic.Saldo –= 500 Write (Aplic); Read (Conta); Conta.Saldo += 500; Write (Conta);		Reinicia com $TS(T_1) = 3$ R-TS(Aplic) = 3 W-TS(Aplic) = 3 R-TS(Conta) = 3 W-TS(Conta) = 3

Protocolo com base em Timestamps

- Características da ordenação por timestamps
  - Evita deadlocks e garante a serialização de conflito na ordem dos timestamps
  - Não garante recuperação e nem impede cascata
- Regra de escrita de Thomas
  - Se a transação T tentar alterar um dado Q, e TS(T) for menor que W-TS(Q), não é preciso abortar a transação; basta ignorar a operação de escrita
  - Garante a serialização de visão
  - Aumenta a concorrência entre transações

Protocolo de Validação

- Garante serialização e evita rollback em cascata
- Fases de execução
  1. Fase de leitura e execução: a transação lê os dados e escreve apenas em variáveis locais
  2. Fase de validação: a transação verifica se as variáveis locais podem ser escritas sem violar a serialização
  3. Fase de escrita: se a transação for validada, ela é efetivada; caso contrário ela é recuperada
- Cada transação T possui três timestamps:
  - Start(T): início da execução
  - Validation(T): início da fase de validação
  - Finish(T): final da transação

Protocolo de Validação

- Regras de validação
  - Supondo duas transações T_i e T_j, sendo que Validation(T_i) < Validation(T_j)
  - Se Finish(T_i) < Start(T_j), as transações não são concorrentes, e a serialização certamente ocorre
  - Se as transações forem concorrentes, a serialização ocorre na ordem dos tempos de validação caso:
    - Dados escritos por T_i não tenham sido lidos por T_j (a escrita de T_i não invalida o valor lido por T_j)
    - Tenhamos Finish(T_i) < Validation(T_j) (T_j não pode invalidar os dados lidos por T_i)
  - Caso contrário, a transação é abortada e desfeita

Protocolo de Validação

- Exemplo de execução com validação:

T_1	T_2	Timestamps
Read (Aplic); Aplic.Saldo –= 500 Read (Conta); Conta.Saldo += 500;		Start(T_1) = 1
Bloqueada	Read (Conta); Read (Aplic); **Validate**; Print (Conta.Saldo + Aplic.Saldo);	Start(T_2) = 2 Validation(T_2) = 3 Finish(T_2) = 4 <Validation(T_1) → OK!
Validate; Write (Aplic); Write (Conta);		Validation(T_1) = 5 Finish(T_1) = 6

Protocolo de Validação

- Exemplo de execução com validação:

T_1	T_2	Timestamps
Read (Aplic); Aplic.Saldo –= 500 Read (Conta); Conta.Saldo += 500; **Validate**; Write (Aplic);		Start(T_1) = 1 Validation(T_1) = 2
Bloqueada	Read (Conta); Read (Aplic); **Validate**;	Start(T_2) = 3 Validation(T_2) = 4 T_1 escreveu Aplic→abortar
Write (Conta);	Bloqueada	Finish(T_1) = 5
	Read (Conta); Read (Aplic); **Validate**; Print (Conta.Saldo + Aplic.Saldo);	Start(T_2) = 6 > Finish(T_1) → OK! Validation(T_2) = 7 Finish(T_2) = 8

Protocolos Multi-Versão

- Permite que um dado tenha vários valores
  - Operações de escrita criam novas versões do dado
  - Quando é feita a leitura, o SBD escolhe a versão do dado que será lida de modo a garantir a serialização
  - Operações de leitura não precisam aguardar, pois sempre há uma versão do dado pronta para ser lida
  - Protocolo determina como as versões são usadas
- Protocolos com base em múltiplas versões
  - Multi-versão com ordenação por timestamps
  - Multi-versão com bloqueio em duas fases
- Garantem a criação de escalas serializáveis

Protocolos Multi-Versão

- Multi-versão com ordenação por timestamps
  - Cada versão do dado possui timestamps de leitura (R-TS) e escrita (W-TS), além do valor do dado
  - Uma transação T sempre acessa a versão Q_k do dado com o maior W-TS que seja menor ou igual a TS(T)
  - A transação T sempre lê a versão Q_k
  - Ao escrever, T é desfeita se TS(T) < R-TS(Q_k); senão, se TS(T) = W-TS(Q_k), o valor de Q_k é alterado; caso contrário, uma nova versão de Q é criada
  - Versões antigas, com W-TS(Q_k) < TS(T), sendo T a última transação executada, podem ser removidas

Protocolos Multi-Versão

- Exemplo de escala multi-versão com timestamps

T_1	T_2	Timestamps
Read (Aplic); Aplic.Saldo –= 500 Write (Aplic);		$TS(T_1) = 1$ $R\text{-}TS(Aplic_1) = 1$ $W\text{-}TS(Aplic_2) = 1$
Bloqueada	Read (Conta); Read (Aplic); Print (Conta.Saldo + Aplic.Saldo);	$TS(T_2) = 2$ $R\text{-}TS(Conta_1) = 2$ $R\text{-}TS(Aplic_2) = 2$
Read (Conta); Conta.Saldo += 500; Write (Conta);		$R\text{-}TS(Conta_1) = 2$ $R\text{-}TS(Conta_1) > TS(T_1)$ → T_1 e T_2 refeitas

Protocolos Multi-Versão

- Multi-versão com bloqueio em duas fases
  - Usa timestamps e contador de commits (ts-counter)
  - Distingue transações de atualização e somente-leitura
  - Transação de atualização (update)
    - Faz o bloqueio em duas fases
    - Cada **write** cria uma nova versão Q_k do dado quando fizer o commit com $TS(Q_k)$ = ts-counter
  - Transação somente-leitura (read-only)
    - TS(T) = ts-counter no momento que ela se inicia
    - Lê a versão do dado com o maior $TS(Q_k) \leq TS(T)$

Protocolos Multi-Versão

- Exemplo de multi-versão c/ bloqueio em 2 fases

T_1	T_2	Timestamps
Lock-X (Aplic); Read (Aplic); Aplic.Saldo –= 500 Write (Aplic);		ts-counter = 1 → Lê $Aplic_1$ = 1000 $TS(Aplic_2)$ = 2
Bloqueada	Lock-S (Conta); Read (Conta); Unlock (Conta);	$TS(T_2)$ = ts-counter = 1 → Lê $Conta_1$ = 1000
Lock-X (Conta); Read (Conta); Conta.Saldo += 500; Write (Conta); Unlock (Conta); Unlock (Aplic);	Bloqueada	→ Lê $Conta_1$ = 1000 $TS(Conta_2)$ = 2 ts-counter = 2
	Lock-S (Aplic); Read (Aplic); Print (Conta+Aplic); Unlock (Aplic);	→ Lê $Aplic_1$ = 1000

Controle de Deadlock

- Deadlock ocorre quando temos um conjunto de transações no qual todas estão em espera, uma aguardando o término da outra para prosseguir
- Técnicas para controle de deadlock
  - Prevenção de deadlock
    - Preferível se a probabilidade de ocorrerem deadlocks for muito alta
  - Detecção e recuperação de deadlock
    - Mais eficiente se ocorrerem poucos deadlocks
- Rollback pode ser necessário em todos os casos

Controle de Deadlock

- Prevenção de deadlock usando esperar-morrer
  - Uma transação T_i somente espera um dado mantido por T_j se esta for mais nova; caso contrário T_i aborta
- Prevenção de deadlock usando ferir-esperar
  - Uma transação T_i somente espera um dado mantido por T_j se esta for mais antiga; caso contrário ela obriga T_j a abortar e a liberar o dado (T_i fere T_j)
- Prevenção de deadlock usando timeout
  - Pedidos de bloqueio possuem um tempo máximo de espera; se o dado não for liberado neste tempo, a transação é abortada e reiniciada

Controle de Deadlock

- Detecção de Deadlock
  - Algoritmo verifica estado do sistema periodicamente para determinar se transações estão em deadlock
  - O sistema precisa manter informações sobre a alocação dos dados e as solicitações pendentes
  - Deadlocks podem ser detectados usando gráficos de espera, nos quais um ciclo indica um deadlock

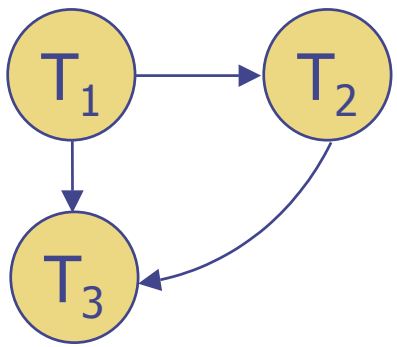


Gráfico sem ciclo

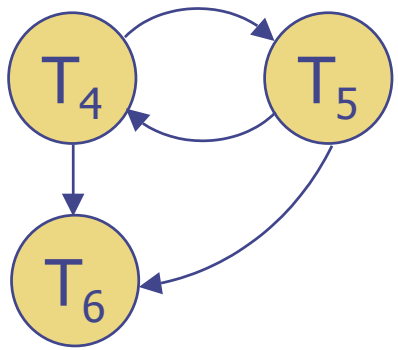


Gráfico com ciclo

Controle de Deadlock

- Recuperação de deadlock
  - Após detectar o deadlock, precisamos abortar uma transação para quebrar o ciclo de espera
  - Devemos escolher a transação em função do custo do rollback, determinado em função do:
    - Tempo que a transação está em processamento
    - Tempo necessário para conclusão da transação
    - Número de acessos a dados já efetuados
    - Número de acessos a dados que faltam p/ concluir
    - Número de transações a recuperar em cascata
    - Número de abortos sofridos (para evitar inanição)
    - etc.

Inserção e Remoção de Dados

- Operações de inserção e remoção devem ser consideradas no controle de concorrência, pois são conflitantes com qualquer outra operação
- Protocolo de bloqueio em duas fases
  - Devemos bloquear o dado em modo exclusivo para inserir ou remover
- Protocolo com base em timestamps
  - Devemos inicializar W-TS(Q) e R-TS(Q) com o timestamp da transação que insere o dado Q
  - Devemos fazer o teste para operações que alteram o valor do dado no momento da remoção

Inserção e Remoção de Dados

- Tuplas fantasma
  - As transações abaixo não são conflitantes se fizermos controle de concorrência por tupla, mas entram em conflito se forem executadas concorrentemente

```
select sum(saldo)                    insert into contas
from contas                          values(123,'João',50.00)
```

  - Soluções possíveis
    - Bloquear toda a relação → pouca concorrência
    - Bloquear um campo especial que indica que uma transação está inserindo ou removendo dados
    - Bloquear o índice da relação

Concorrência em Índices

- A concorrência também precisa ser controlada ao acessar e modificar índices
  - No acesso aos dados, os blocos dos índices consultados devem ser bloqueados em modo compartilhado
  - Para modificar os dados, a transação deve procurar todos os blocos dos índices que apontam para a tupla e obter bloqueio exclusivo dos blocos
  - Transações não podem inserir ou remover tuplas sem atualizar todos os índices da relação
  - Regras de bloqueio em 2 fases devem ser seguidas