Controle de Concorrência

- Protocolos de Bloqueio
- Protocolo com base em *Timestamps*
- Protocolos Multi-versão
- Controle de *Deadlock*
- Inserção e Remoção de Dados
- Concorrência em BDs Distribuídos

Controle de Concorrência

- Controle de concorrência é usado para garantir a consistência e o isolamento de transações
- Protocolos de controle de concorrência garantem a serialização no processamento de transações
  - Protocolos de bloqueio: controlam o acesso ao BD, bloqueando os dados que estão sendo usados
  - Protocolos com base em *timestamps*: associam marcas de tempo às operações de leitura e escrita
  - Protocolos de validação: efetuam testes de validação dos dados antes das operações de escrita

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

Protocolos de Bloqueio

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

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

- Protocolo de Bloqueio em Duas Fases (2PL)
  - 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 *rollback* em cascata

Protocolos de Bloqueio

- Bloqueio em Duas Fases – Exemplo:

T_1	T_2

Lock-X (Aplic);
Read (Aplic);
Aplic.Saldo = Aplic.Saldo – 500;
Write (Aplic);
Lock-X (Conta);
Unlock (Aplic); // Inicia 2ª fase
Read (Conta);	

	Lock-S (Conta);
Conta.Saldo = Conta.Saldo + 500; Write (Conta); Unlock (Conta);	*Bloqueada*

	Read (Conta);
	Lock-S (Aplic);
	Unlock (Conta); // Inicia 2ª fase
	Read (Aplic);
	Print (Conta.Saldo + Aplic.Saldo);
	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;	
	Lock-S (Conta); Read (Conta); Lock-S (Aplic);
Write (Aplic); Lock-X (Conta);	*Bloqueada*
Bloqueada	*Bloqueada*
Não executa: Unlock (Aplic); Read (Conta); Conta.Saldo = Conta.Saldo + 500; Write (Conta); Unlock (Conta);	**Não executa:** Unlock (Conta); Read (Aplic); Print (Conta.Saldo + Aplic.Saldo); Unlock (Aplic);

Protocolos de Bloqueio

- Bloq. em 2 fases não evita *rollback* em cascata

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

Conta.Saldo = Conta.Saldo + 500;
Write (Conta); // ABORTA
Unlock(Conta);
	Bloqueada

	Read (Conta); // ABORTA
	Unlock (Conta);
	Print (Conta.Saldo + Aplic.Saldo);
	Unlock (Aplic);

Protocolos de Bloqueio

- Variantes do Bloqueio em Duas Fases
  - Evitam o *rollback* em cascata
  - Usados pela maioria dos SGBDs
  - 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 de Bloqueio

- Bloqueio em Duas Fases Severo – Exemplo:

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

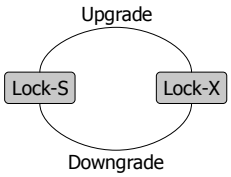
Protocolos de Bloqueio

- Bloqueio em Duas Fases Rigoroso – Exemplo:

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

Protocolos de Bloqueio

- Protocolos com Conversão de Bloqueios
  - Utilizam instruções de conversão de bloqueios para alternar entre modos de bloqueio diferentes
  - Upgrade(Q): bloqueio compartilhado → exclusivo
  - Downgrade(Q): bloqueio exclusivo → compartilhado



Protocolos de Bloqueio

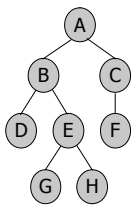
- Modos de Bloqueio Intencional
  - Propiciam maior paralelismo entre transações
  - Intenção de compartilhamento (IS): indica que um bloqueio compartilhado poderá ser solicitado
  - Intenção de exclusividade (IX): indica que um bloqueio exclusivo poderá ser solicitado
  - Compartilhado com intenção de exclusividade (SIX): indica que há bloqueio compartilhado ativo, mas que um bloqueio exclusivo pode vir a ser solicitado

	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*

Protocolos de Bloqueio

- Protocolos de Bloqueio baseados em Grafos
  - Determinam uma ordenação parcial no acesso aos dados usando um grafo de precedência
  - Supondo que exista a relação de precedência A→B no grafo, 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 grafos; um dos mais simples é o protocolo de grafo em árvore

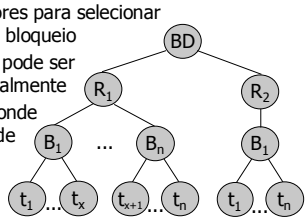
Protocolos de Bloqueio



- Protocolo de Grafo 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 selecionar a granularidade do bloqueio
  - Cada nó da árvore pode ser bloqueado individualmente
  - Cada nível corresponde a uma granularidade de bloqueio



Protocolos com base em *Timestamps*

- Usam *timestamps* (marcas de tempo) associados às transações e aos dados para ordenar as 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

Protocolos 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)

Protocolos 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
	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) → T_1 e T_2 abortadas

Protocolos com base em *Timestamps*

- Características da ordenação por *timestamps*
  - Evita *deadlocks* e garante a serialização de conflito na ordem dos *timestamps*
  - Não impede a cascata nem garante a recuperação
- 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

Protocolos Multi-Versão

- Permitem que um dado tenha vários valores
  - Operações de escrita criam novas versões do dado
  - Quando é feita a leitura, o SGBD 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);		$TS(T_1) = 1$ $R\text{-}TS(Aplic_1) = 1$ $W\text{-}TS(Aplic_2) = 1$
	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 abortadas

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 com $TS(Q_k)$ = *ts-counter* quando faz o *commit*
  - Transação somente-leitura (*read-only*)
    - Não precisa manter bloqueios até o final
    - 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);		$TS(T_1)$ = ts-counter = 1 → Lê $Aplic_1$ = 1000 → Cria $Aplic_2$ = 500
	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);		→ Lê $Conta_1$ = 1000 → Cria $Conta_2$ = 1500 $TS(Conta_2) = TS(Aplic_2)$ = 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

T_1	T_2
Lock-X (A); Read (A);	
	Lock-S (B); Read (B); Lock-S (A);
Write (A); Lock-X (B);	*Bloqueada*
Bloqueada	*Bloqueada*
Deadlock!	

Controle de *Deadlock*

- Técnicas para Controle de *Deadlock*
  - Prevenção de *deadlock*
    - Evita os deadlocks antes que estes ocorram
    - Preferível se a probabilidade de ocorrerem *deadlocks* for muito alta
  - Detecção e recuperação de *deadlock*
    - Não evita os deadlocks, mas os detecta e impede o bloqueio indefinido das transações envolvidas
    - Mais eficiente se ocorrerem poucos *deadlocks*
  - *Rollback* pode ser necessário independentemente da técnica utilizada

Controle de *Deadlock*

- Prevenção de *Deadlock*
  - Usuário deve sempre acessar dados na mesma ordem
  - Estratégias de prevenção usadas por SGBDs
    - Esperar-morrer: T_i só espera um dado mantido por T_j se esta for mais nova; caso contrário T_i aborta
    - Ferir-esperar: T_i somente espera um dado mantido por T_j se esta for mais antiga; caso contrário ela obriga T_j a abortar e liberar o dado (T_i fere T_j)
    - *Timeout*: pedido de bloqueio possui 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 grafos de espera, nos quais um ciclo indica um *deadlock*

Grafo sem ciclo *Grafo 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 BDs Distribuídos

- Controle de concorrência é essencial em BDDs
  - É preciso garantir a consistência e o isolamento, apesar da fragmentação e da replicação dos dados
  - Devemos evitar *deadlocks* distribuídos, que são ainda mais difíceis de detectar e recuperar
- Os protocolos de controle de concorrência são modificados para trabalhar em BDs distribuídos
- Bloqueio Único (Centralizado)
  - Um *site* funciona como gerenciador de *locks*, que administra todos os pedidos de bloqueio de dados
  - É simples, mas sujeito a falhas, e pouco escalável

Concorrência em BDs Distribuídos

- Bloqueio Múltiplo (Distribuído)
  - Há gerenciadores de bloqueio em diferentes *sites*
  - Cada gerenciador administra os bloqueios de um conjunto de dados
  - Evita o 'gargalo' do protocolo centralizado
  - Impede o acesso ao dado se o gerenciador falhar
- Bloqueio com Cópia Primária
  - Cada dado possui uma cópia primária em um *site*
  - O bloqueio é solicitado ao *site* com a cópia primária daquele dado, que administra todos os bloqueios
  - É tão simples e escalável quanto o protocolo anterior
  - Impede o acesso às réplicas se o primário falhar

Concorrência em BDs Distribuídos

- Bloqueio pela Maioria
  - Cada *site* possui um gerenciador que administra o bloqueio dos dados locais
  - O bloqueio de dados com réplicas é concedido se a maioria dos *sites* permitir
  - Mais complexo para implementar que os anteriores
- Bloqueio Parcial
  - Assim como no bloqueio por maioria, cada *site* possui um gerenciador de bloqueio dos dados locais
  - Bloqueios compartilhados são obtidos contactando o gerenciador de somente uma réplica do dado
  - Bloqueios exclusivos devem ser solicitados em todos os gerenciadores de bloqueio de todas as réplicas

Concorrência em BDs Distribuídos

- *Timestamp* Único (Centralizado)
  - Um único site define as marcas de tempo usando um contador lógico ou o seu relógio local
  - Sofre de problemas de confiabilidade e escalabilidade
- *Timestamp* Global (Distribuído)
  - Usa o princípio de relógios lógicos (Lamport, 1978)
  - Um *timestamp* é formado por uma leitura do relógio concatenada com o identificador do site que o gerou
  - Sites atualizam os relógios com base nos *timestamps* de sub-transções que recebem para executar
  - Mais robusto e escalável que o esquema centralizado

Concorrência em BDs Distribuídos

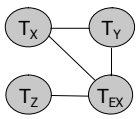
- *Deadlocks* e Replicação
  - É preciso evitar *deadlocks* ao acessar réplicas
  - Se duas transações forem acessar um dado duplicado e cada uma delas bloquear uma réplica do dado, nenhuma das duas conseguirá prosseguir
  - Solução: obrigar que as transações bloqueiem as réplicas na mesma ordem
- *Deadlocks* e Fragmentação
  - Situação semelhante à anterior pode ocorrer quando duas transações bloqueiam fragmentos de dados
  - Solução: bloquear fragmentos seguindo uma ordem

Concorrência em BDs Distribuídos

- Detecção de *Deadlock*
  - O algoritmo de detecção de *deadlock* visto anteriormente pode ser usado em BDs distribuídos se usarmos o protocolo de bloqueio centralizado
  - Para os demais protocolos, precisamos modificar o algoritmo de detecção de *deadlock*
    - Para montar o grafo, um gerente central precisa obter informações sobre os *locks* mantidos por todos os gerenciadores de bloqueio
    - Podem aparecer ciclos falsos no grafo devido ao atraso na comunicação → abortos desnecessários
  - Outra opção é fazer a detecção de modo distribuído

Concorrência em BDs Distribuídos

- Detecção de *Deadlock* (cont.)
  - Na detecção distribuída, cada *site* monta um grafo de espera com os bloqueios mantidos localmente
  - Um nó T_{EX} é adicionado ao grafo para representar as esperas externas (dados bloqueados por outros *sites*)
  - Um ciclo envolvendo apenas nós locais indica *deadlock*
  - Um ciclo passando por T_{EX} indica um possível *deadlock*



  - Temos que confirmar a possibilidade de *deadlock* contactando os gerenciadores dos *sites* envolvidos