## Aspectos Operacionais

Centro de Informática UFPE

**Controle de Concorrência**

## Controle de Concorrência

- Definição

  Concorrência é a propriedade de uma transação poder ser executada em paralelo com outras transações



## Controle de Concorrência

- Justificativa de uso

  Com a execução de várias transações ao mesmo tempo, o processador pode ser compartilhado entre essas transações, melhorando a eficiência global do computador dado que uma maior quantidade de trabalho é executada em menos tempo



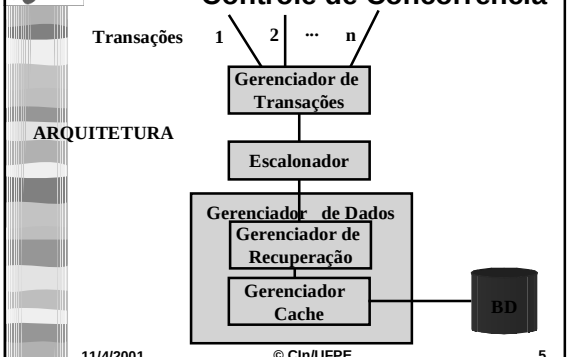
## Controle de Concorrência

- Justificativa de um controle

  É necessário que o sistema monitorize a interação entre transações concorrentes, de modo a evitar que elas destruam a consistência do BD



## Controle de Concorrência






## Controle de Concorrência

- Técnicas de Bloqueio (Locks)
- LOCK
  - ⇨ Variável associada a um item de dados no BD que descreve o status desse item com respeito a possíveis operações a serem aplicadas a ele
- LOCK BINÁRIO
  - ⇨Dois estados ou valores: locked (1) ou unlocked (0)

## Controle de Concorrência

- LOCK BINÁRIO (Cont.)
  - Um lock distinto é associado a cada item do BD - referenciado como lock (x) para o item x
  - Operações incluídas nas transações : lock_item e unlock_item, implementadas como operações indivisíveis
  - Um gerenciador é mantido pelo SGBD para registrar e controlar o acesso a locks



## Controle de Concorrência

- LOCK BINÁRIO (continuação)
  - Registro de lock: <nome-do-item, LOCK>
  - Tabela de locks: mantém esses registros
  - Regras:
    - Uma transação T tem que executar uma operação lock_item(x) antes de qualquer read_item ou write_item executada por T



## Controle de Concorrência

- Regras (Cont.)
  - Uma transação T tem que executar a operação unlock_item(x) após todo read/write completados em T
  - Uma transação não pode executar outra operação lock_item(x) se já tem um lock sobre x



## Controle de Concorrência

- Regras (Cont.)
  - Uma transação T não pode executar um unlock_item(x) a menos que tenha um lock sobre x
  - Apenas uma transação pode ter um lock num dado item



## Controle de Concorrência

- LOCK DE MODO MÚLTIPLO
  - Três operações (indivisíveis)
    - *read_lock(x)* - lock compartilhado
    - *write_lock(x)* - lock exclusivo
    - *unlock(x)*
  - Registro de Lock

    <nome-do-item, LOCK, número-de-leituras>
  - Regras:
    - Uma transação T tem que executar uma operação *read_lock(x)* ou *write_lock(x)* antes de qualquer *read-item(x)* em T



## Controle de Concorrência

- Regras (Cont.):
  - Uma transação T tem que executar uma operação *write_lock(x)* antes de qualquer *write-item(x)* em T
  - Uma transação tem que executar uma operação *unlock(x)* após todas as operações *read_item(x)* e *write_item(x)* completadas em T
  - Uma transação T não executará um *read_lock(x)* se já tem um lock (read) compartilhado ou um lock (write) exclusivo em **x** (pode ser relaxado)

## Controle de Concorrência

- Regras (Cont.)
  - Uma transação T não executará outro *write_lock(x)* se já tem um lock (read) compartilhado ou um lock (write) exclusivo em **x** (pode ser relaxado)
  - Uma transação T não executará um *unlock(x)* a menos que já tenha um lock compartilhado ou exclusivo em **x**.



## Controle de Concorrência

- INCREMENTO DE LOCK

  Uma transação após ter um read_lock(x) e sendo a única que detém x, pode posteriormente executar um write_lock(x) sobre x
- DECREMENTO DE LOCK

  Após ter um lock exclusivo sobre um item x, uma transação T pode decrementar o lock, executando um read_lock(x)



## Controle de Concorrência

- Locks binários ou múltiplos não garantem a serializabilidade de escalonamentos nos quais as transações participam. É necessário um protocolo adicional que contempla o posicionamento de locks e unlocks em cada transação.



## Controle de Concorrência

- BLOQUEIOS EXCLUSIVOS

  Se uma transação T contiver um bloqueio exclusivo em algum objeto, então nenhuma transação distinta T' pode fazer bloqueio daquele objeto até que T libere o seu bloqueio



## Controle de Concorrência

- Protocolo

  Qualquer transação que pretenda atualizar um registro R, primeiro terá que acessar aquele registro e bloqueá-lo exclusivamente. Se o bloqueio não puder ser feito, a transação entra em um estado de espera e apenas continuará o processamento quando o registro se tornar disponível e o bloqueio puder ser concedido



## Controle de Concorrência

- Protocolo (Cont.)
  - O sistema terá que garantir que uma transação forçada a esperar por causa desse protocolo, eventualmente sairá do estado de espera (não sofre livelock)
  - Para evitar a possibilidade de perda de atualizações, nenhuma transação poderá ter acesso aos dados de uma atualização não efetuada

## Controle de Concorrência

- BLOQUEIOS COMPARTILHADOS

  Se uma transação T mantiver um bloqueio compartilhado em algum objeto, então uma transação T' distinta também pode fazer um bloqueio compartilhado sobre aquele objeto. Mas, nenhuma transação T' distinta pode fazer um bloqueio exclusivo sobre aquele objeto até que todos os bloqueios compartilhados do objeto tenham sido liberados



## Controle de Concorrência

| T' \ T | Excl | Comp | – |
|---|---|---|---|
| Excl | N | N | C |
| Comp | N | C | C |
| – | C | C | C |



## Controle de Concorrência

- BLOQUEIOS DE DUAS FASES

  Se todas as transações obedecerem às seguintes regras:

  ①Antes de operar sobre qualquer objeto, a transação primeiro adquire um bloqueio sobre aquele objeto;

  ②Após liberar o bloqueio, a transação não adquire mais bloqueios.



## Controle de Concorrência

- Todas as execuções intercaladas dessas transações são serializáveis.
- Uma transação que obedeça as regras 1 e 2 acima é dito que satisfaz o bloqueio de duas fases.
- Fases:
  - ⇨ Crescente:quando os bloqueios são feitos
  - ⇨ Encolhimento: quando os bloqueios são liberados



## Controle de Concorrência

- Se o decréscimo de lock for permitido, ele tem que ser feito na fase de encolhimento.
- Pode limitar o acesso concorrente

  Uma transação tem que manter um lock sobre **x** mesmo que não precise dele, se tiver que exercer um lock sobre outro objeto **y**. Por outro lado, para poder liberar **x** mais cedo, a transação terá que bloquear **y** mais cedo.



## Controle de Concorrência

- BLOQUEIOS DE DUAS FASES (cont.)
  - ⇨Variação do protocolo:
    - ➳ BÁSICO (visto anteriormente)
    - ➳CONSERVADOR OU ESTÁTICO: bloqueia todos os itens que acessará, antes de iniciar a execução da transação
    - ➳ ESTRITO (mais popular): não libera os locks até o commit ou abort
  - ⇨ Podem Causar Deadlock ou Livelock

## Controle de Concorrência

- DEADLOCK
  É uma situação em que duas ou mais transações estão em estado simultâneo de espera, cada uma aguardando que uma das demais libere um bloqueio para ela poder prosseguir.
- Principais métodos para solucionar o impasse (podem resultar na repetição da transação):



## Controle de Concorrência

- Principais Métodos (Cont.)
  - Protocolo de Prevenção de Deadlock
    - Conservador: se algum dos itens não pode ser bloqueado, nenhum será bloqueado
    - Ordenado: tentar por uma ordenação em todos os itens e os locks só podem ocorrer segundo esta ordem.



## Controle de Concorrência

- DEADLOCK (cont.)
  - Técnicas Baseadas em Timestamping
    - Identificador único assinalado a cada transação
    - Ordenados segundo a ordem em que as transações começaram
    - Principal vantagem: não usa bloqueios, logo deadlock é impossível



## Controle de Concorrência

- Protocolos
  Quando uma transação solicita um bloqueio de um registro que já está bloqueado por outra transação, então um dos dois procedimentos é seguido:
  - Wait-die - se a transação que solicitou o bloqueio é a mais antiga, pode aguardar. Se for a mais nova, sofre rollback e recomeça mais tarde com mesmo timestamping.



## Controle de Concorrência

- Protocolos (Cont.)
  - Wound-wait - se for a transação mais nova, pode aguardar. Se for a mais antiga, interrompe a mais nova, a qual sofre rollback. Recomeça mais tarde com mesmo timestamping.



## Controle de Concorrência

- Protocolo de Detecção e Recuperação
  - Verificação da existência de deadlock:
    - sempre que uma solicitação de bloqueio cause “espera”
      - permite detecção imediata do impasse
      - acarreta um overhead
    - em alguma base periódica
      - reduz o overhead
      - alguns deadlocks serão detectados tardiamente

## Controle de Concorrência

- DEADLOCK (cont.)
  - Condições ideais
    - Pequena interferência entre as transações
    - Transações curtas
    - Locks de poucos itens
    - Carga da transação leve



## Controle de Concorrência

- Quebra de deadlock consiste na escolha de uma das transações para forçá-la a um rollback:
  - a que foi iniciada mais recentemente
  - a que tiver feito o menor número de bloqueios
  - a que tiver feito o menor número de atualizações
- Situação oposta:
  - usar um esquema de prevenção de deadlock.



## Controle de Concorrência

- LIVELOCK

  A transação não pode prosseguir por um período indefinido de tempo enquanto outras transações continuam normalmente (esquema injusto de espera por lock)
  - associar first-come-first-serve
  - incremento de prioridade com o tempo



## Controle de Concorrência

- STARVATION

  Problema similar ao livelock, ocorre se o algoritmo seleciona a mesma transação como vítima repetidamente, causando abort repetidos e nunca acabando a execução
  - os protocolos wait-die e wound-wait apresentados, evitam o starvation



## Controle de Concorrência

- O ambiente de BD difere de outros ambientes no que concerne os problemas de bloqueio devido a:
  - O conjunto de objetos bloqueáveis não só é muito grande como também muda muito rápido
  - Objetos bloqueáveis são tipicamente endereçados pelo conteúdo
  - O escopo preciso do bloqueio para determinada transação normalmente é determinado com muita rapidez.



## Controle de Concorrência

Outros Métodos

- Técnicas de controle de concorrência com multi-versão
  - Várias versões X1, X2,..., Xn de cada item de dado X são guardadas pelo sistema
    - Baseada na ordem dos timestamping
    - Bloqueio de duas fases

## Controle de Concorrência

- Técnica de Validação (otimista)
  - ➪As atualizações efetuadas não são efetivadas no BD até o final da transação
    - ➵Fase de leitura: leitura mas atualizações não efetivadas
    - ➵Fase de validação: verifica se tudo correu em ordem
    - ➵Fase de escrita: se validação OK, atualiza o BD



## Controle de Concorrência

***Granularidade de Bloqueio no BD***

**< concorrência** ↓ **campo específico registro arquivo BD** ↑ **> número de bloqueios**

**Quanto maior número de *locks* a ser gerenciado:**
- **maior *overhead***
- **mais espaço para a tabela de *locks***

**OBS:**
1. **os bloqueios em todo o BD são liberados no final do programa e não no *commit***
2. **os bloqueios de atualização só fazem sentido na granularidade mais fina**



## Controle de Concorrência

Controle de concorrência sem bloqueios

- TIMESTAMPS

  Criados pelo SGBD para identificar uma transação:
  - ➪São assinalados na ordem em que as transações são submetidas ao sistema [TS(T)]
  - ➪ Geração: contador ou clock do sistema



## Controle de Concorrência

- Ordenação de transações baseadas em Timestamp:
  - ➪ executam livremente
  - ➪ asseguram que para cada item acessado por mais de uma transação, a ordem dos acessos não viola a serializabilidade do escalonamento.



## Controle de Concorrência

Ordenação de transações baseadas em Timestamp (cont.):
- ➪Para tanto são associados dois timestamps para cada item do BD
  - ➵read_TS(x) - timestamp de leitura de x

    O maior timestamp de todas as transações que leram x com sucesso.
  - ➵write_TS(x) - timestamp de escrita de x

    O maior timestamp de todas as transações que atualizaram x



## Controle de Concorrência

ORDENAÇÃO (Cont.)
- ➪Para uma transação disparar um read ou write em x, o timestamp de T é comparado com o read_TS ou write_TS de x para garantir a ordenação da execução.

## Controle de Concorrência

- Algoritmo para write

a . se read_TS(x) > TS(T)

abortar T e dar rollback em T

/* transação mais nova leu antes desta atualizar */

b. se write_TS(x) > TS(T)

não executar a operação mas prosseguir o processamento /* para não perder a última atualização feita */

c . Não ocorrendo a nem b

write_item(x) e write_TS(x) = TS(T)



## Controle de Concorrência

- Algoritmo para read

a . se write_TS(x) > TS(T)

abortar T e dar rollback em T

/* transação mais nova escreveu o item antes desta ler */

b. se write_TS(x) < = TS(T)

read_item(x) e

read_TS(x) = maior (TS(T), atual read_TS(x) )



## Controle de Concorrência

- OBS: Livelock pode ocorrer se T é continuamente abortada e recomeçada (Recomeço Cíclico)