

# Complementos de Base de Dados

---

Resumos  
2016/2017

João Alegria | 68661

# Capítulo 7

## Controlo Distribuído da Concorrência

---

- Problema de sincronização de transações simultâneas de tal forma que as 2 propriedades seguintes sejam atingidos:
  - A consistência da BD é mantida
  - O grau máximo de concorrência é atingido
- O controlo da concorrência lida com os propriedades de transações: isolamento, consistência
- A execução das transações em série atinge a consistência mas diminui a performance
- Objetivo: equilíbrio entre consistência e concorrência.

### Serialização

- Escalonamento: ordem cronológica pela qual as instruções de várias transações são executadas
- A serialização de escalonamento é usado para identificar quais escalonamentos estão corretos quando as execuções da transição tiverem intercalação das suas operações nos escalonamentos.
- A função primária de um controlador de concorrência é gerar um escalonamento serializável para a execução de transações pendentes.
- Se num escalonamento S as operações não estão intercaladas (ou seja, as operações de cada transação ocorrem consecutivamente) dizemos que o escalonamento é serial.
- A execução serial de um conjunto de transações mantém a consistência da base de dados, porém pode acarretar estados de inatividade da CPU, desperdiçando processamento.
- A execução concorrente de transações deve deixar a base de dados num estado que possa ser alcançado por uma execução sequencial em alguma ordem.
- Caso essa situação seja alcançada serão resolvidos problemas como os de atualizações perdidas.
- Assegurando o isolamento e não permitindo que os resultados incompletos sejam acedidos por outras transações.
- Um escalonamento S (ou Schedule, também chamado de Histórico) é definido sobre um conjunto de transações  $T=\{T_1, T_2, \dots, T_n\}$  e especifica uma ordem intercalada de execução dessas operações de transações.
- Um escalonamento pode ser especificado como uma ordem parcial sobre T.

#### • Tipos de Escalonamento em SGBDDs:

- Escalonamento local - Histórico da execução da transação em cada site
- Escalonamento global - Se a BD não é replicada e cada histórico local é serializável, a sua união (histórico global) também é se a ordem da serialização local é idêntica (união da BD não replicada e histórico local serializável)

**EXEMPLO**

$T_1:$	$Read(x)$	$T_2:$	$Read(x)$
	$x \leftarrow x + 5$		$x \leftarrow x * 10$
	$Write(x)$		$Write(x)$

#### Schedule:

- \* Site1:  $S_1 = \{R_1(x), W_1(x), R_2(x), W_2(x)\}$
- \* Site2:  $S_2 = \{R_2(x), W_2(x), R_1(x), W_1(x)\}$

- São serializáveis localmente.
- No entanto não são globalmente pois violam consistência tendo em conta que produzem valores diferentes.

## Taxonomia da Concorrência

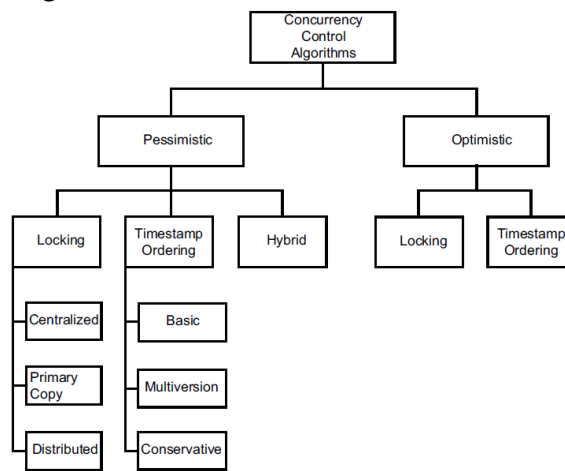
Modos de abordagem de controlo de concorrência:

- Distribuição de BD - total ou parcialmente replicada
- Topologia da rede - “star”, circular ou broadcasting
- Primitiva de sincronização - timestamp ordering ou locking-based

Podem ser usados em algoritmos com 2 pontos de vista:

- Pessimista: muitas transações entrarão em conflito, portanto a sincronização da execução concorrente ocorre mais cedo no seu ciclo de execução.
- Otimista: poucas transações em conflito portanto a sincronização de execução concorrente ocorre até ao fim

### Algoritmos de controlo de concorrência



### ➔ Locking-Based

- Controlo de concorrência baseado em bloqueio, que assegura que os dados compartilhados por operações conflitantes sejam acedidas por uma única operação de cada vez.
- É conseguido pela associação de um “bloqueio” a cada unidade de bloqueio
- Esse bloqueio é definido por uma transação antes de ser acedida e é redefinida no final do seu uso.
- Modos de Bloqueio: Bloqueio de Leitura e Bloqueio de Escrita

### ➔ Locking 2 fases (2PL)

- A regra de bloqueio de 2 fases estabelece que nenhuma transação deve solicitar um bloqueio após libertar um dos seus bloqueios.
- Como alternativa: uma transação não deve libertar um bloqueio até ter a certeza de que solicitará outro bloqueio

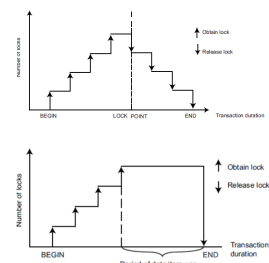
As 2 fases são:

- Fase de Crescimento: Obtém bloqueios e acede a itens de dados
- Fase de Contração: Liberta os bloqueios

Qualquer escalonamento / histórico gerado por um algoritmo que obedece à regra 2PL é serializável.

Existem 2 tipos de 2PL:

- Cascading Aborts: Se uma T aborta depois de libertar alguns bloqueios pode causar outra T a abortar
- Strict 2PL: Todos os bloqueios são libertados simultaneamente para evitar abortos em cascata



## ➔ 2PL Centralizado

- Um único site é responsável pelo coordenador de bloqueios (LM). Os bloqueios são solicitados pelo TM ao LM.

Vantagens	Desvantagens
- Fácil de implementar	- Provoca “engarrafamento” - Pouca Fiabilidade

## ➔ 2PL Distribuído

- O coordenador de bloqueio é distribuído pelos sites todos. O LM de cada site é responsável pelo bloqueio dos seus dados.
- As mensagens são enviadas para todos os LMs participantes

Desvantagens	
- Difícil gestão de <i>deadlocks</i> - Custos de comunicação elevados	Quando 2 ou mais operações ficam bloqueadas, esperando uma pela outra

## ➔ 2PL Cópia-Primária

- Extensão direta do 2PL centralizado
- Distribui LM em vários sites e cada um irá administrar um dado conjunto de unidades de bloqueio

## ➔ Timestamp Ordering (TO)

- Os algoritmos de controlo de concorrência do tipo timestamp definem uma ordem de serialização para cada transação e executam-nas de acordo com ela.
- Para estabelecer essa ordem, o TM atribui a cada transação um timestamp
- Transações são individualmente serializados e executados
- O TM atribui um TS único a cada T
- TS são únicos localmente e globalmente e crescem monotonamente

### • TO regra:

Dadas 2 operações O1 e O2 respetivas às T1 e T2:

- T1 é executada antes de T2 apenas se o timestamp de T1 for menor que o timestamp de T2  
 $ts(T1) < ts(T2)$
- Neste caso diz-se que T1 é a transação mais velha e a T2 a transação mais nova.

### • 2 Tipos de Timestamp:

- read (rts)
- write (wts)

## ➔ TO Básico

- O TM atribui um timestamp a cada T
- T são executadas imediatamente
- Scheduler compara a TS do item de dados com o TS de operação
- Se o TS do item entrar em conflito com o TS de operação então T é abortada
- Não existe bloqueios - T são abortados e reiniciados em caso de conflito (degradação da performance)
- Como as transações nunca esperam enquanto mantêm direitos de acessos aos dados o algoritmo básico de TO nunca provoca impasses. No entanto, o preço para se livrar de impasses é a reiniciação potencial de uma transação várias vezes.

## ➔ TO Conservativo

- Usado para minimizar o número de reiniciações de TS
- S atrasa cada operação até não existir uma operação com o TS menor (mais velhas) (S atrasa (O) até não existirem (O) mais velhas)
- Se isto for assegurado, não se rejeita mais TS

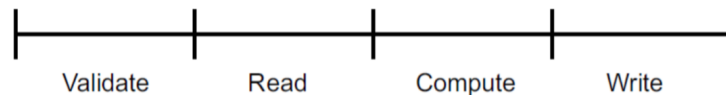
Cada S:

- Tem uma fila para cada TM
- Armazena informação de ordem ascendente (e executa)
- Mesmo assim a reiniciação pode ocorrer

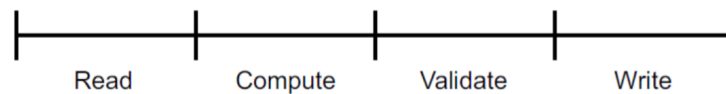
## ➔ TO Multiversão

- Os writes de operações não modificam os itens, apenas criam uma nova versão de item de dados
- Os reads são sempre bem sucedidos no seu TS ( $T_i$ )
- Writes são rejeitados se uma T que tem um TS maior já leu o item de dados
- Pode ser: Otimista ou Pessimista

**Pessimista:**



**Otimista:**

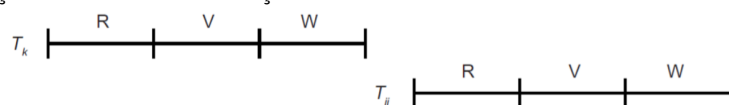


## • Políticas de Timestamp:

- TS na transação e não nos itens de dados (no rts e wts)
- Quando a primeira subtransação atinge a fase de validação
  - O mesmo TS é definida a todas as subtransações

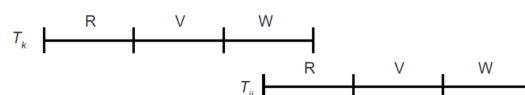
### Regra 1

Se toda a  $T_k$  onde  $TS(T_k) < TS(T_{ij})$  completou a fase *write* antes de  $T_{ij}$  começar a fase *read* a validação é bem sucedida



### Regra 2

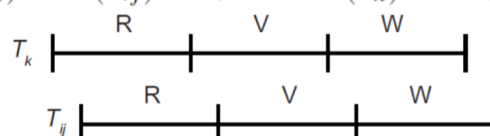
Se uma  $T_k$  em que  $TS(T_k) < TS(T_{ij})$  e  $T_k$  completa a sua fase de *write* enquanto de  $T_{ij}$  está na fase de *read*, a validação é bem sucedida se:  
 $WS(T_k) \cap RS(T_{ij}) = \emptyset$ .



### Regra 3

Se existe um  $T_k$  que  $TS(T_k) < TS(T_{ij})$  e que  $T_k$  completa a sua fase de *read* antes de  $T_{ij}$  completar a sua fase de *read*, a validação é bem sucedida se:

$$WS(T_k) \cap RS(T_{ij}) = \emptyset, \text{ and } WS(T_k) \cap WS(T_{ij}) = \emptyset.$$



Atualizações de  $T_k$  não afetam a fase de *read* ou *write* de  $T_{ij}$