

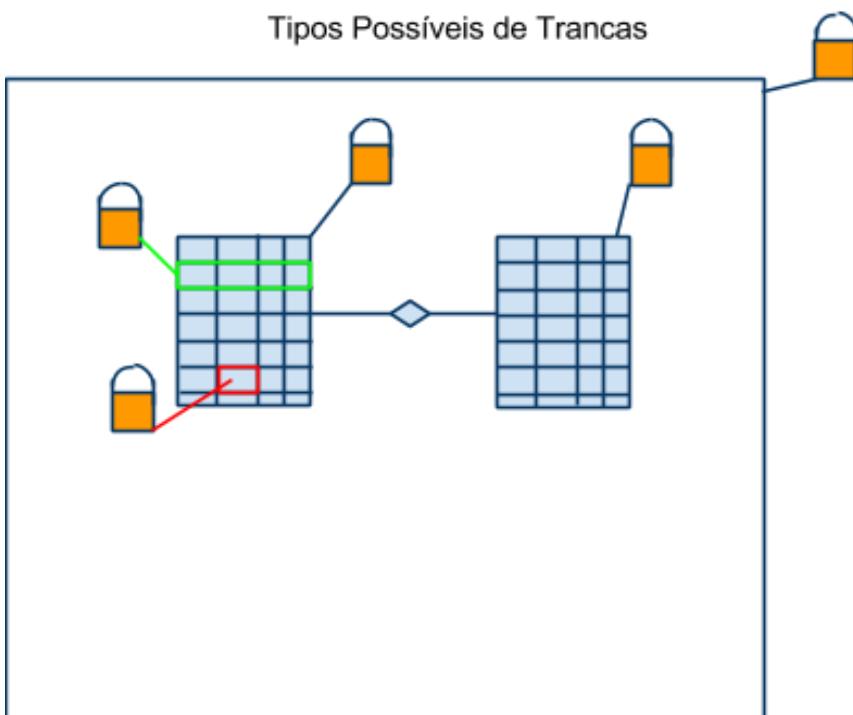
BD2



## Parte 2

### Controle de Transações Através de Trancas (Lock)

Tranca: Variável associada a um item de dado indicado se alguma transação está trabalhando com tal item.



Trancas impactam na performance do banco.

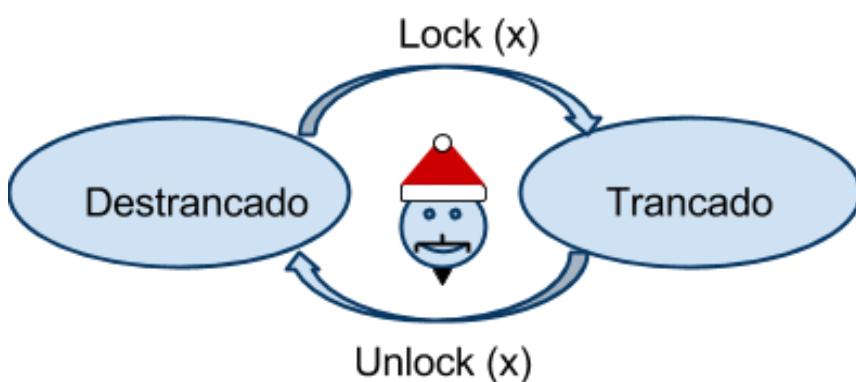
## Trancas Binárias

Dois Estados Possíveis:

- Trancado (Locked)
- Destrancado (Unlocked)

Duas Funções:

- Lock (x)
- Unlock (x)



suspensa até o 1 der unlock

∨

B1 B2 L1(x) L2(x) R1(x) L1(y) R1(y) W1(y) U1(y) U1(x) L2(x) R2 (x)  
/\

o unlock de x, L2 volta e pode dar lock

Trancas binárias não são utilizadas na prática, já que não permite leitura de mais de uma transação (no exemplo acima, T1 tranca X, apenas para lê-lo).

Deadlock:

com

$L1(x) \quad R1(x) \quad L2(y) \quad R2(y) \quad L1(y) \quad L2(x)$   
 $\wedge \quad \quad \quad \wedge \quad \quad \quad \wedge \quad \wedge$



As setas significam quem está esperando por quem.

### Como evitar deadlock

- não automatizado: obrigar o programador que utilize trancas sempre na mesma ordem X, Y, Z... em todas as transações.
- automatizado: monitorar o grafo de espera.
- automatizado: timeout, matar transações que demorarem demais.(Se uma transação estiver executando há muito tempo, pode significar que estão em deadlock; dá-se, portanto, um kill na transação)

### Trancas Compartilhadas

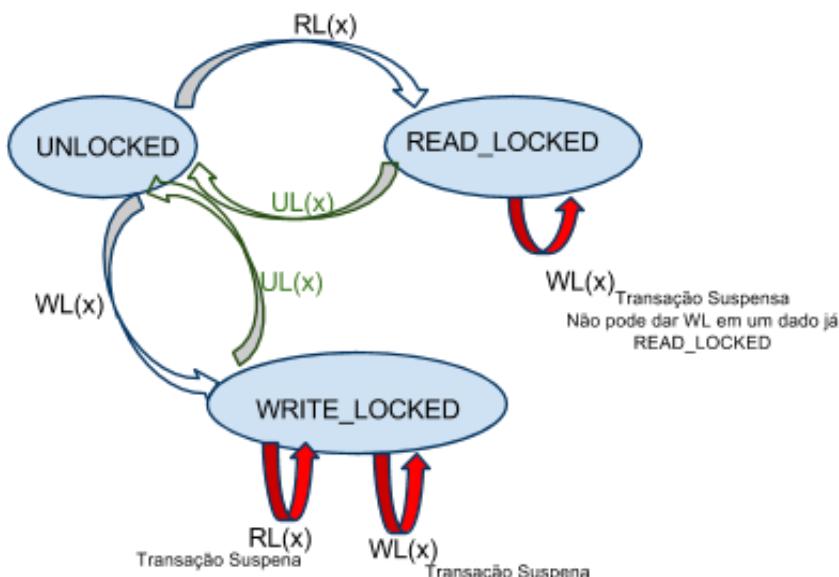
Três estados possíveis:

- destrancado (unlocked)
- trancado para leitura (read\_locked)
- trancado para escrita (write\_locked)

Três operações:

- read\_lock (RL(x))

- write\_lock (WL(x))
- unlock (UL(x))



Suspensa, pois faltou a T1 fazer

*unlock*RL1(x) RL2(x) R2(x) WL2(y) R2(y) W2(y) UL2(x) R1(x) **WL3(x)** RL1(y) R1(y) WL1(x) W1(x)UL1(x) **WL3(x)***Liberado*

Utilizando ReadyWriteLock

1:RL1(x), 2:RL2(x), 3:URL2(x), 4:WL2(x), 5:WL1(x)

A transação 2 ao executar o passo 4 fica suspensa a T1 está trancando com Read\_Locked, ja a transação 1 ao executar o passo 5 não fica porque a 1 já estava no Read\_Locked.

## Protocolo de Alocação de Trancas em Duas Fases

### Variação 1: 2PL Conservativo

A transação deve bloquear todos os itens antes que a transação inicie

Predeclaração de “readset” e “writeset”

Se algum dos itens não pode ser bloqueado, nenhum será. A transação espera por todos os itens

Vantagem: garante ausência de deadlocks

Desvantagem: difícil de implementar na prática

### Variação 2: 2PL Estrito

Nenhum writelock é liberado até que a transação execute um commit ou abort

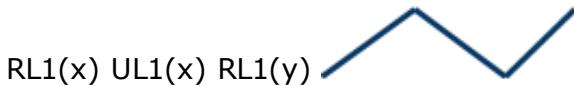
Nenhuma outra transação pode acessar um item até que T comprometa ou aborte.

Garante escalonamentos estritos, sem necessidade de rollback em cascata

Variação mais popular do 2PL

Não é livre de deadlocks

### O que não pode acontecer:



Funções de lock -&gt; precisam ser atômicas.

### LOCK\_ITEM(x)

```

BEGIN CRITICAL REGION
WHILE (LOCKED(x) = TRUE)
    WAIT (UNTIL LOCKED(x) = FALSE)
LOCK(x) <- TRUE
END CRITICAL REGION
END
  
```

### UNLOCK\_ITEM(x)

```

BEGIN CRITICAL REGION
LOCKED(x) <- FALSE
IF (há transações esperando liberação de x)
    NOTIFY(uma das transações esperando)
END CRITICAL REGION
END
  
```

### Controle de recuperabilidade:

R1(x) W1(x) R2(x) R1(y) W2(x) C2 A1

|  
leitura suja|  
fail, operação inconsistente

No caso acima, a T1 aborta depois do commit, por mais que o SGBD volte no log, ele não vai conseguir voltar para o X que era antes do W1 (já que o valor sujo de x foi gravado pelo C2).

**Escalonamento Recuperável:** Se nenhuma transação T é confirmada (committed) antes que toda transação T' que lê item de dado alterado por T seja confirmada.

S: r1(x); w1(x); r2(x); r1(y); w2(x); c2; a1

S é recuperável?

Não. Note que T2 termina ANTES de T1, mas DEPENDE do valor X que T1 escreveu. Portanto, T2 deveria ser desfeita após ter sido efetivada.

S: r1(x); r2(x); w1(x); r1(y); w2(x); c2; w1(y); c1

Note que S sofre do problema de atualização perdida....

S é recuperável?

Sim, pois T2 não depende de T1 e pode terminar antes que T2

### **Rollback em cascata:**

R1(x) W1(x) R2(x) R1(y) W2(x) C1 C2

|

|

leitura suja

vamos fingir que ocorreu um erro aqui e isso virou

um aborto A1

Pelo fato da T2 ter lido x sujo, ela terá que ser abortada também.

Caso houvesse esse A1 a transação ficaria o seguinte:

R1(x) W1(x) R2(x) R1(y) W2(x) A1 A2

**Escalonamento livre de rollback em cascata:** se nenhuma transação lê item alterado por outra transação não confirmada (not committed)

R1(x) W1(x) R2(x) R1(y) W2(x) W1(x) A1 <--- não é livre, teria que esperar todas as leituras do T2 para depois do C1, ex:

R1(x) W1(x) R1(y) W1(x) C1 R2(x) W2(x)  
É recuperável e livre de rollback em cascata

**Escalonamento Estrito:** se nenhuma transação nem lê nem escreve um item x até que a ultima transação que tenha escrito x seja confirmada ou abortada.

Suponha x=9

W1(x,5) W2(x,8) A1 <- tá errado, mas não entendi direito -> A transação T1 seta 5 no x, a transação T2 depois seta 8, mas depois a T1 é abortada e o rollback vai fazer com que o x volte a ser 9, mas era pra ficar 8 porque a T2 é independente/não ocorreu erro nesse caso.

## **Banco de Dados Distribuído**

Estado (idEst, nome)

Cidade (idEst(FK), idCid)

Bairro (idEst(FK), idCid(FK), idBairro, nome)

Logr\_Bairro (idEst(FK), idCid(FK), idBairro(FK), idLogr(FK), CEP, nroInicial, nroFinal)

Logradouro(idLogr, nome, idTipoLogr(FK))

TipoLogr(idTipoLogr, nome, descr)

Agencia(idAgencia, nome, idEst(FK), idCid(FK), idBairro(FK), idLogr(FK), CEP(FK), numero, complem)

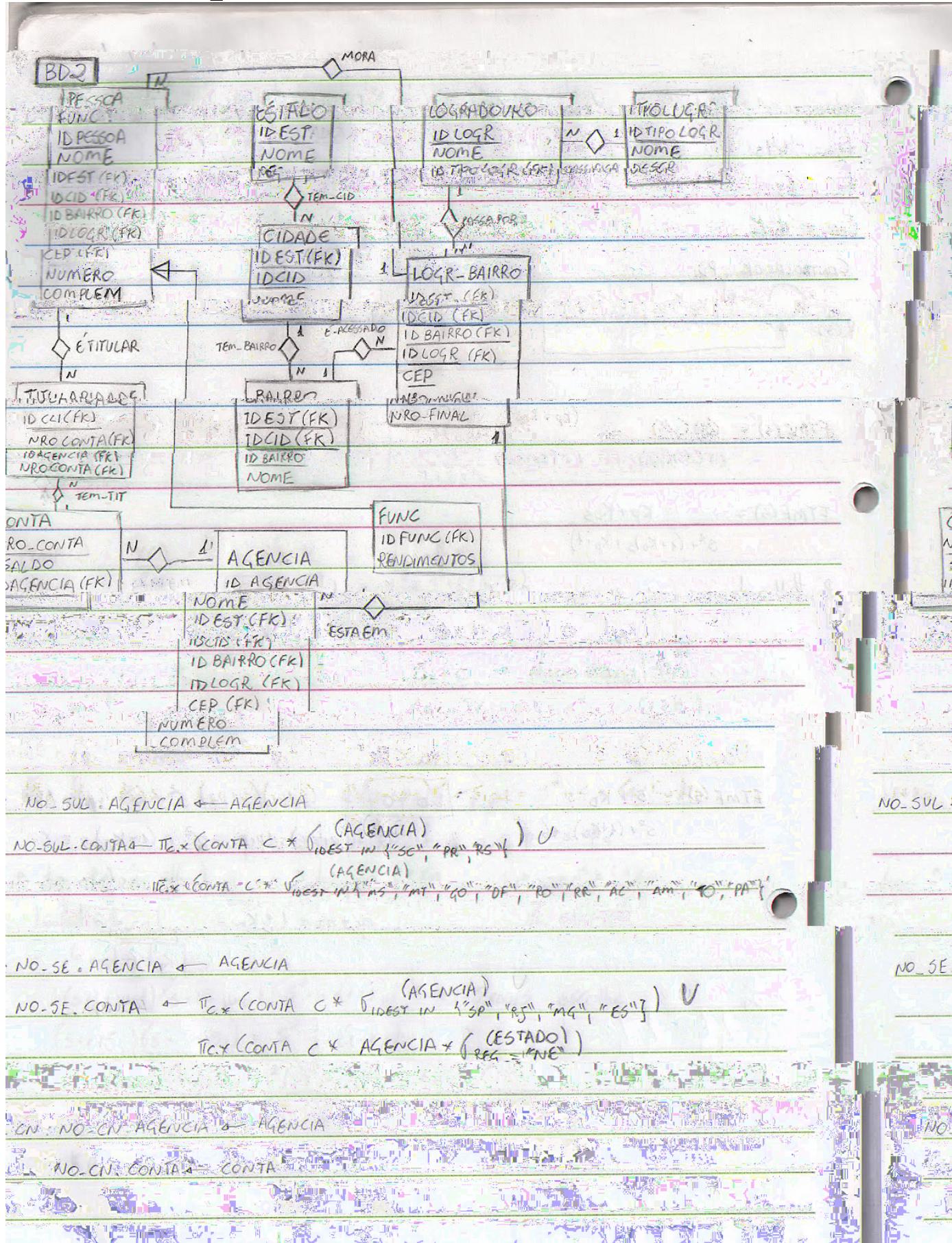
Conta (idAgencia(FK), nroConta, saldo)

Titularidade(idCli(FK), nroConta(FK))

Pessoa(idPessoa, func, nome, idEst(FK), idCid(FK), idBairro(FK), idLogr(FK), CEP(FK), numero, completo)

Horizontal (usando seleção)

NÓ\_SUL: NO\_SUL.Agencia &lt;-- Agencia

NO\_SUL.Conta <--  $\pi_{c.*}(Conta \ c * \sigma_{idEst \in \{PR, SC, RS\}}(Agencia))$ 

## Visão Geral dos Problemas de Controle de Transações em Bancos de Dados distribuídos

- Lidar com múltiplas cópias de itens de dados
- Falha de nodos
- Falha das linhas de comunicação

- Commit distribuído
- Deadlock distribuído

Atomicidade sincrona - ou todo mundo da commit ou todo mundo da rollback

Protocolo Two-Phase:

- No pode cair na fase:

**■ Preparacao:**

**■ Commit:** Se no cair nessa fase, quando ele acordar deverá perceber que esteve fora e perguntar para os outros o que aconteceu.

No coordenador pode cair: Nesse caso terá sempre um outro no backup, onde ele será eleito através da confirmação, por broadcast de todos os nós, de que o chefe caiu (com mais de 50%)  
obs: essa eleição ocorre tanto para transações quanto para cópias (ascenção de cópias primárias/secundárias)

- Escalonamentos seriais:
  - Exclusão mutua por bloqueio (lock)
  - Métodos Otimistas

Relógios Lógicos (analogia): Todo mundo que realizar uma operação decrementa o contador global...

Suponha um BDD com 4 nós ( $N_1, N_2, N_3$  e um nó coordenador de transações  $C_i$ ) e os seguintes Logs:

$N_1$	$N_2$	$N_3$	$C_i$
<pre>&lt;start T1&gt; &lt;write T1,x,1,2&gt; &lt;start T2&gt; &lt;ready T1&gt; &lt;write T2,q,5,7&gt; &lt;ready T2&gt;</pre>	<pre>&lt;start T3&gt; &lt;write T3,g,0,2&gt; &lt;start T2&gt; &lt;ready T2&gt; &lt;write T2,r,5,7&gt; &lt;ready T2&gt;</pre>	<pre>&lt;start T2&gt; &lt;write T2,w,1,8&gt; &lt;start T3&gt; &lt;write T3,b,1,5&gt; &lt;ready T2&gt; &lt;ready T3&gt;</pre>	<pre>&lt;start T1&gt; &lt;start T2&gt; &lt;start T3&gt; &lt;write T1,y,8,0&gt; &lt;write T1,z,1,4&gt; &lt;write T3,a,5,3&gt; &lt;write T2,f,6,9&gt; &lt;prepare T1&gt; &lt;prepare T2&gt; &lt;commit T1&gt; &lt;prepare T3&gt;</pre>

- suponha que  $N_2$  falha. Que ações  $C_i$  irá tomar com relação a  $T_2$  e  $T_3$ ?
- suponha que  $N_1$  voltou a ativa após uma falha. Que ações ele irá tomar?
- suponha que o BDD se recupera por completo de uma falha de sistema. Que ações  $C_i$  irá tomar?
- suponha que  $C_i$  falha e  $N_3$  assume a coordenação de  $T_2$  e  $T_3$ . Que ações  $N_3$  irá tomar?
- suponha que  $C_i$  falha e  $N_1$  assume a coordenação de  $T_1$  e  $T_2$ . Que ações  $N_1$  irá tomar?

- Commit  $T_2$ , Abort/Rollback  $T_3$
- "O  $C_i$ " Perguntar sobre  $T_2$ , e verá que tem que commitar.
- Continua as suas ações, commit  $T_2$  e wait for <ready  $T_3$ > do  $N_2$
- esperar o ready de  $T_3$  pelo  $N_2$ , mandar commit  $T_2$ . Caso  $N_2$  de abort ou ocorra time-out da  $T_3$ ,  $T_3$  fará rollback (primeira fase do 2PC).
- Commit  $T_2$ . Se for uma falha mais séria,  $N_1$  e os demais nos teriam que eleger um novo backup (se nesta abordagem for usado o protocolo com site de backup).

Quando um nó volta, quais suas ações?

Tá meia boca, se alguém puder melhorar...

Primeiro se da conta de que ficou "fora", olha seu log e conclui transações pendentes e pergunta para o coordenador se perdeu algo, isto é necessário para que ele volte a um estado consistente.

A relação Func está no nodo1 e Filiais está fragmentada por cidade, estando as filiais de Fpolis no nodo2 (FFp) ( $n_{FFp} = 20$ ) e as filiais de Blumenau no nodo3 (FBlu) ( $n_{FBlu} = 10$ ) Dada a seguinte consulta no nodo2:

$\Pi_{\text{nome}, \text{codF}, \text{Func.nroFil}} (\sigma_{\text{Func.nroFil} = \text{Filiais.nroFil}} (\text{Func} \times \text{Filiais})) \wedge (\text{Filiais.cidade} = 'Fpolis' \vee \text{Filiais.cidade} = 'Blumenau')$

pode-se considerar algumas alternativas, como:

A1) filtrar os funcionários desejados (de *Fpolis* e de *Blumenau*) no nodo1 e enviar o resultado para o nodo 2  
A2) trazer os atributos desejados de funcionários e os códigos das filiais de Blumenau para o nodo2 e processar a consulta no nodo2

- a) Qual alternativa tem o menor custo de transmissão?
- b) O que pode ser processado em paralelo em A1 e A2?

a) A que possui o menor custo de transmissão a priori é a A1. Levando em consideração que existem várias filiais o custo de pegar os funcionários seria muito grande.

b) A1 -> As seleções das filiais por Fpolis e Blumenau ocorrem paralelamente no Nod2 e no Nod3

A media que forem chegando ao Nod1 também pode se iniciar a consulta.

A2 -> As seleções de funcionário do Nod1 pro Nod2 e do Nod3, filiais de Blumenau, para o Nod2.

A media que forem chegando ao Nod2 também pode se iniciar a consulta.

## 1. Defina os termos fragmentação e replicação de dados.

Fragmentação define a granularidade dos dados em cada partição/nó.

Em sala foram discutidos três tipos:

Horizontal ( subconjunto de tuplas );

Vertical ( relação apenas com alguns atributos, lembrando que a chave deve ser replicada em todas as relações para fazer join); e

Mista ( Horizontal + Vertical ).

Replicação diz respeito a cópia dos dados em diferentes partições/nós, pode ser parcial ou total. Aumenta a disponibilidade e paralelismo.

## 2. Qual a diferença entre replicação síncrona e assíncrona?

"Atualizações síncrona só realiza a transação **se garantir** atualizações em todos as partições."  
"Atualizações assíncrona, cópias atualizadas periodicamente, **réplicas podem estar inconsistente**"

## 3. O que é independência de localização em um sistema de banco de dados distribuído?

Os usuários do sistema não devem saber o local onde estão localizados os dados: devem se comportar como se os dados estivessem armazenados localmente.

4. Sobre o protocolo 2PL, quais as diferenças entre 2PL Conservativo, 2PL Estrito e 2PL Commit ?

Conservativo - 'Adquire em primeiro caso todas as trancas' Libera as trancas aos poucos

Estrito - Libera todas as trancas de uma vez (ou algo assim) :P

Commit - Commit em duas fases, uma de preparação e outro de finalização.

Que tipo de 2PL é este:

RL1(x), R1(x), WL1(x), W(x), UL1(x)  
(creio q estrito)

Pode ser dito que é estrito, porque liberou as trancas somente no final.  
Não é conservativo porque tem um  $R1(x)$  antes do  $WL1(x)$ , segundo o protocolo 2PL para ser conservativo deve primeiramente adquirir todas as trancas necessárias antes de realizar qualquer outra ação.

5. Como identificar em um escalonamento se ele é 2PL Conservativo ou Estrito?

Conservativo - Não pode ter um UL's no meio de L.

Estrito - Não pode ter L no meio de UL's

**6. Marque V ou F:**

- (v) - 2PL conservativo evita deadLock
  - (v) - 2PL estrito evita rollback em cascata e garante recuperabilidade.
  - (v) - Todo escalonamento livre de Rollback em cascata é recuperável.

7. Qual a diferença entre escalonamento estrito e 2PL estrito?

8. O principal objetivo do escalonamento estrito é?

Evitar leitura suja.

9. Considere: <-- ## Questão valendo um milhão de dólares ##>

BOOKS (Book#, Primary\_author, Topic, Total\_stock, \$price)

BOOKSTORE (Store#, City, State, Zip, Inventory value)

Total\_stock is the total number of books in stock, and inventory\_value is the total inventory value for the store in dollars.

a. Give an example of two simple predicates that would be meaningful for the BOOKSTORE relation for horizontal partitioning.

Talvez por cidade, e estado. heheheheh

b. How would a derived horizontal partitioning of STOCK be defined based on the partitioning of BOOKSTORE?

Bah, essa é foda! Se a fragmentação for por cidade/estado, talvez pegar o total de livros de cada cidade mesmo. Mas o legal seria se tivesse o total total sabe.

c. Show predicates by which BOOKS may be horizontally partitioned by topic.

d. Show how the STOCK may be further partitioned from the partition h

(log com base maior, menos entradas de indice -> menos blocos de disco, SGBD vai optar por esse quando possivel, senao busca binaria e por ultimo busca exaustiva). log(n)

#### 4- Busca via Indice Secundario:

Desempenho inferior apesar de ser de complexidade (log n + K).

#### Obs.:

1. "árvores sao arvores B+, com ponteiros para trás e para frente"
2. "banco de dados cria índices para colunas...se nao busca exaustiva"

#### 5 - Busca via Indice de Agrupamento (clustering): O(logN + K)

6 - Busca via Hashing O(1) Só se aplica a consultas por um único valor, intervalo, ou conhece a distribuição de dados mais uniformemente.

---

`select * from empregado where cod_e = '1234'`

ð cod\_e = '1234' (Empregado) -> **busca por indice primario**

`select * from funcionario where salario > 500`

ð salario > 500 (funcionario) -> **Indice Secundario/Agrupamento** - O(logN + K) k=numero de funcionários (pode ser alto)

`select * from funcionario where cod_dep(FK) = 5` - **se é chave estrangeira -> indice secundario**

`select * from funcionário where salary > 10.000 and sexo='F'` - tunning: só vale a pena ter índice para atributo se o existe mais que 100 valores para ele. - verificar por qual procurar primeiro: "menos salários maior que 10000 ou sexo feminino?"

- Se a seletividade for muito dispare [1% (para salários > 10000 e 50% (número de mulheres)] mais vale por **força bruta**.

"Comparar ponteiro com ponteiro dos arquivos de indice, sem precisar verificar isso nos blocos".

`select * from funcionario where salary > 10000 OR sexo = 'F'`

`select *from func where nome like "%fileto"` -> indice para mascaras completes efficient.

- **Indice invertido.**

DATABASE SYSTEMS - livro

## Algoritmos para junção

### 1. laços aninhados (nested loops)

para cada r ∈ R	O( R . S )
para cada s ∈ S	se  R ~ S ~N
se r.K = s.K	O(N.N)=O(N <sup>2</sup> )
gere REG r ° s	

### 2. usando índice para buscar registros correspondentes (HASH)

para cada r ∈ R	
aux = busca (r.K,S) O(log S )	
para todo s ∈ aux	
gere reg r ° s	

O(|R|.log|S|)      O(N.logN)

### 3. algoritmo do ziper (Sort Merge Join)

$O(|R| + |S|)$

BD2

HEURÍSTICAS PARA OTIMIZAÇÃO DE ÁRVORES DE CONSULTA:

REGRAS DE TRANSFORMAÇÃO DE CONSULTAS:

1.  $\sigma_{C_1}^{(R)} \text{ AND } C_2 \text{ AND } \dots \text{ AND } C_n = \sigma_{C_1}(\sigma_{C_2}(\dots(\sigma_{C_n}(R)), \dots))$
2.  $\sigma_{C_1}(\sigma_{C_2}(R)) = \sigma_{C_2}(\sigma_{C_1}(R))$
3.  $\pi_{\text{LISTA}_1}(\pi_{\text{LISTA}_2}(\dots(\pi_{\text{LISTA}_n}(R), \dots))) = \pi_{\text{LISTA}_1}(R)$
4.  $\pi_{A_1, A_2, \dots, A_n}(\sigma_C(R)) = \sigma_C(\pi_{A_1, A_2, \dots, A_n}(R))$
5.  $R \setminus (R \setminus S) \equiv S$
6.  $\sigma_C(R \setminus S) \equiv \sigma_C(R) \setminus \sigma_C(S)$
7.  $\sigma_C(R \setminus S) \equiv \pi_{A_1, \dots, A_n}(R) \setminus \pi_{B_1, \dots, B_n}(S)$
8.  $R \cup S \equiv S \cup R$   
 $R \cap S \equiv S \cap R$   
 $R - S \neq S - R$
9. PARA  $\Theta = \{X_1, X_2, \dots, X_n\}$ :  
 $(R \Theta S) = (\sigma_{C_1}(R)) \Theta (\sigma_{C_1}(S))$

$$11. \pi_c(R \cup S) = \pi_c(R) \cup \pi_c(S)$$

$$12. \text{NOT}(C_1 \text{ AND } C_2) \equiv \text{NOT}(C_1) \text{ OR } \text{NOT}(C_2)$$

$$\text{NOT}(C_1 \text{ OR } C_2) \equiv \text{NOT}(C_1) \text{ AND } \text{NOT}(C_2)$$

### ALGORITMO DE OTIMIZAÇÃO HEURÍSTICA DE CONSULTAS.

1. USANDO A REGRAS 1.

QUEBRE AS CONSULTAS CONJUNTIVAS EM CASCATAS DE SELEÇÕES.

2. USANDO AS REGRAS 2, 4, 6 E 10

MIXE AS SELEÇÕES PARA TÃO PRÓXIMO DAS FOLHAS DA ÁRVORE DE CONSULTA QUANTO POSSÍVEL

3. USANDO A REGRAS 9.

REARRANGE AS FOLHAS DA ÁRVORE DE CONS. DE MODO QUE AS OPERAÇÕES MAIS RESTRITIVAS SEJAM EXECUTADAS PRIMEIRO

4. COMBINE OPERAÇÕES DE PRODUTO CARTESIANO COM OPERAÇÕES SUBSEQUENTES DE SELEÇÃO DE MODO A OBTER OPERAÇÕES DE JUNÇÃO

5. USANDO AS REGRAS 3, 4, 7 E 11

QUEBRE E MOVA LISTAS DE ATRIBUTOS A SEREM PROJETADOS PARA O MAIS PRÓXIMO POSSÍVEL DAS FOLHAS DA ÁRVORE

6. IDENTIFIQUE SUBÁRVORES DE CONSULTAS QUE PODEM SER REALIZADAS POR UM ÚNICO ALGORITMO

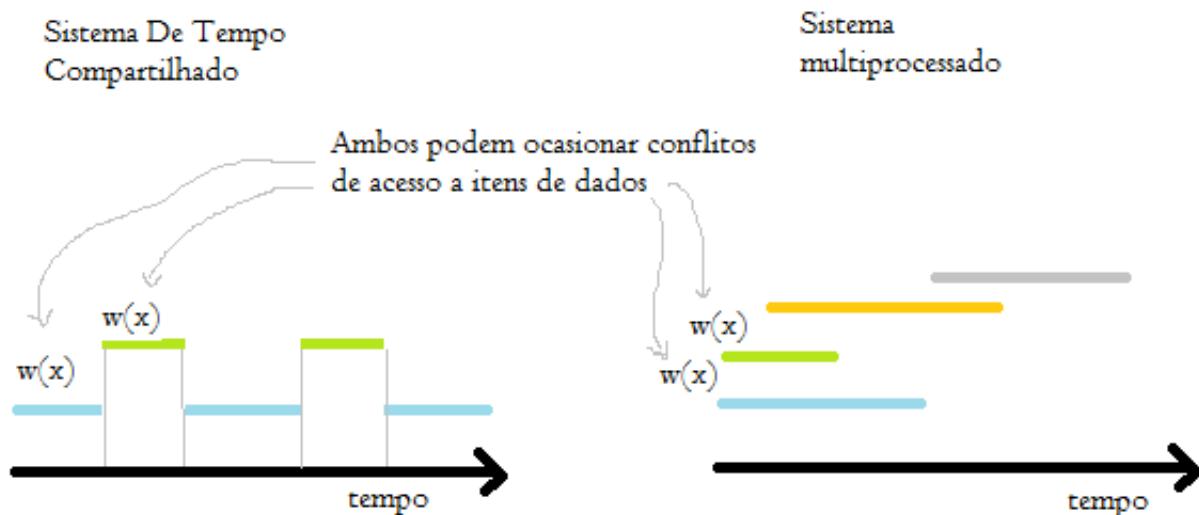
EXERCÍCIO: APLIQUE AS HEURÍSTICAS APRESENTADAS PARA GERAR UMA ESTRATÉGIA DE EXECUÇÃO OTIMIZADA PARA A CONSULTA A SEGUIR. APRESENTE A ÁRVORE DE CONSULTA ANTES E DEPOIS DO PROCESSO DE

credeal



## Controle de Transações

Transação: Sequência de operações de acesso ao BD, que levam o BD de um estado consistente para outro estado consistente.



## Operações de Leitura e Escrita

$R(x) \equiv \text{Read\_item}(x)$  Lê o item de dados  $x$

Passos:

- 1-encontre o endereço do bloco de disco contendo  $x$
- 2- copie o valor de  $x$  do bloco de disco para um buffer em memória ram
- 3-copie  $x$  do buffer para uma variável de programa

$W(x) \equiv \text{Write\_item}(x)$  Escreve o valor de um item de dado  $x$

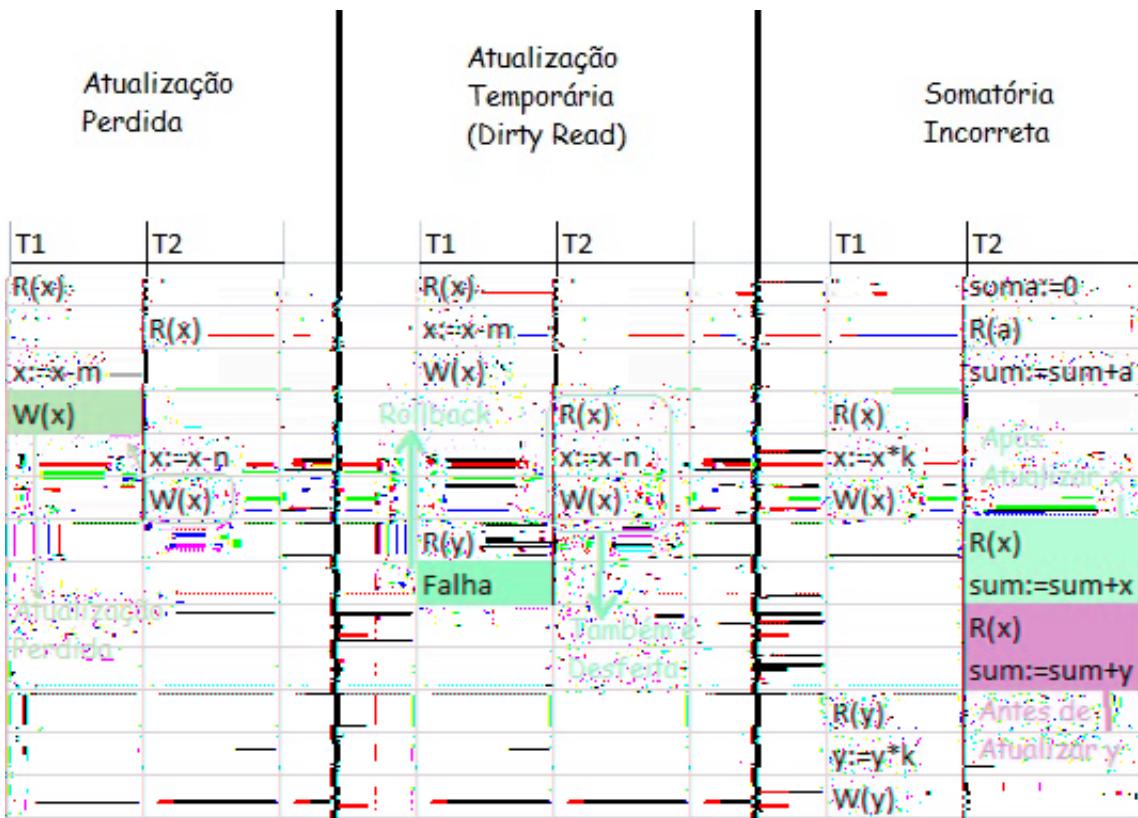
Passos:

- 2 primeiros passos de  $\text{Read\_item}(x)$

3-copie o valor da variável de programa para o buffer em memória correspondente ao bloco de disco que contém  $x$

4-grave o buffer no reg. de disco correspondente

## Anomalias por falta de controle de concorrência



## Propriedades Desejáveis das Transações

- **Atomicidade:** Realiza todas as operações de cada transação ou nenhuma delas.
- **Consistência:** a execução correta de uma transação leva o BD de um estado consistente para outro estado consistente.
- **Independencia:** As atualizações realizadas pela execução de operações de uma transação não podem afetar outras transações enquanto a primeira não fizer "commit".
- **Durabilidade:** Os resultados de uma transação confirmada não podem ser perdidos

**ACID**, de tras pra frente, fica a **DICA!!** fiKdiK

## Escalonamentos de transações.

Sequências de operações de transações realizadas pelo SGBD e registradas no Log.

Exemplos:

SA S1 S2 R1(X) R2(X) W1(X) R1(Y) W2(X) C2 W1(Y) C1

SB R1(X) W1(X) R2(X) W2(X) C2 R1(Y) A1

## Serialidade de Escalonamentos

**Escalonamento serial:** Executa as operações de cada transação em bloco, sem intercalar a execução de operações de transações distintas no tempo. **Não tem problemas de serialidade.**

**Escalonamento não serial:** tem intercalação de operações de transações distintas no tempo.  
**Pode ou não apresentar problemas de serialidade**

- Escalonamento Seriável: Todos os conflitos de execução de operações estão no mesmo sentido. É equivalente a algum(ns) escalonamento(s) serial(ais). Não tem problema.
- Escalonamento não Seriável: Pode causar problema de serialidade.

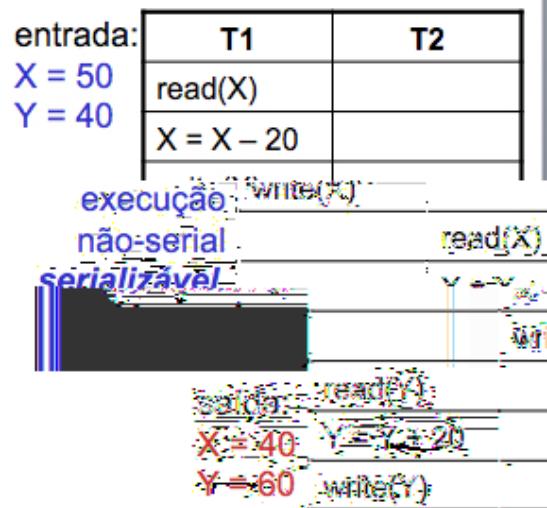
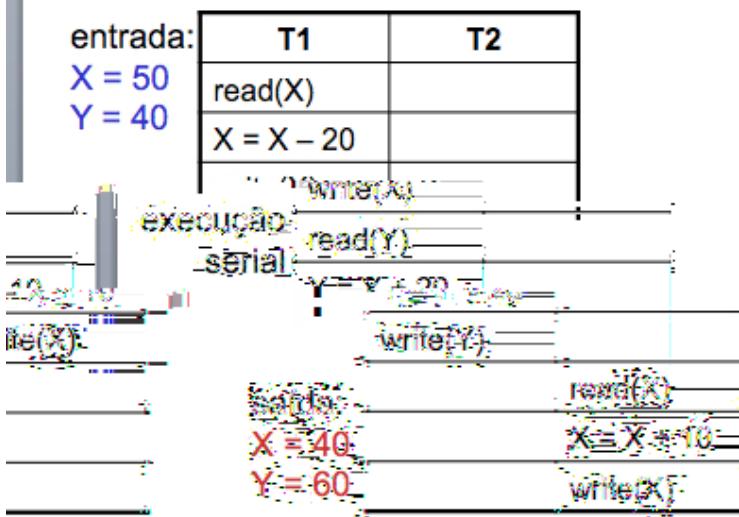
(Pares de) Operações conflitantes em um escalonamento:

- Realizadas por transações distintas
- Ao menos uma das duas operações do par é um write
- Atuam sobre um mesmo item de dado

as 3 condições tem que ser satisfeitas ao mesmo tempo

## Teoria da Serializabilidade

- Garantia de escalonamentos não-serials válidos
- Premissa
  - “um escalonamento não-serial de um conjunto de transações deve produzir resultado equivalente a alguma execução serial destas transações”

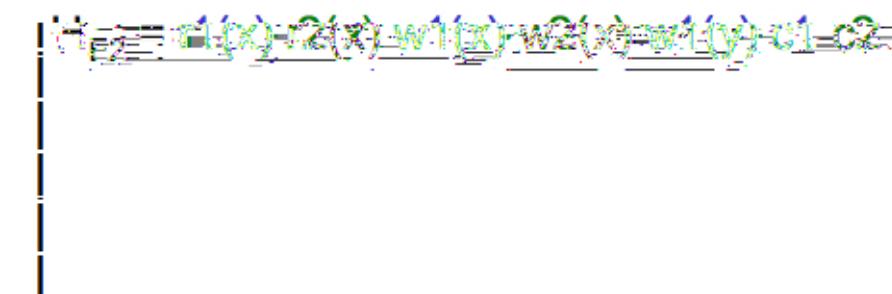


# História

- Representação seqüencial da execução entrelaçada de um conjunto de transações concorrentes
  - operações consideradas
    - *read (r)*, *write (w)*, *commit (c)*, *abort (a)*
- Exemplo

escalonamento não-serializável E2

T1	T2
read(X)	
X = X - 20	
	read(X)
	X = X + 10
write(X)	
	read(Y)
	Y = Y + 20
	write(Y)
	commit( )
	commit( )



Published by [Google Drive](#) – [Report Abuse](#) – Updated automatically every 5 minutes