Universidade Estadual da Paraíba

Banco de Dados

Serviços de um SGBD

Prof. Dr. Vladimir Costa Alencar

valencar@gmail.com

https://www.valencar.com/

@vladimiralencar (twitter)

Julho, 2017

INTRODUÇÃO

Subsistema de Controle de Concorrência

Subsistema de Recuperação à Falhas

Subsistema de Integridade

Subsistema de Segurança

Subsistema de Otimização de Consultas

INTRODUÇÃO

Até então assumimos que nossas aplicações executam 'sozinhas' no SGBD.

Porém, na realidade precisamos permitir **múltiplos** acessos num mesmo instante aos dados do BD, preservando a integridade dos dados.

Ex.: Internet Banking, Amazon.com, Saraiva.com.br

Para garantir a integridade do BD é necessário usarmos o conceito de **Transações 'serializáveis'**

BANCO DE DADOS

CONTROLE DE CONCORRÊNCIA

Prof. Dr. Vladimir Costa de Alencar

valencar.com

vladimir.uepb@gmail.com

CONTROLE DE CONCORRÊNCIA

Uma das propriedades fundamentais de uma transação é o **isolamento**

Quando **várias transações são executadas ao mesmo tempo**, o sistema precisa **controlar** a interação entre as transações simultâneas

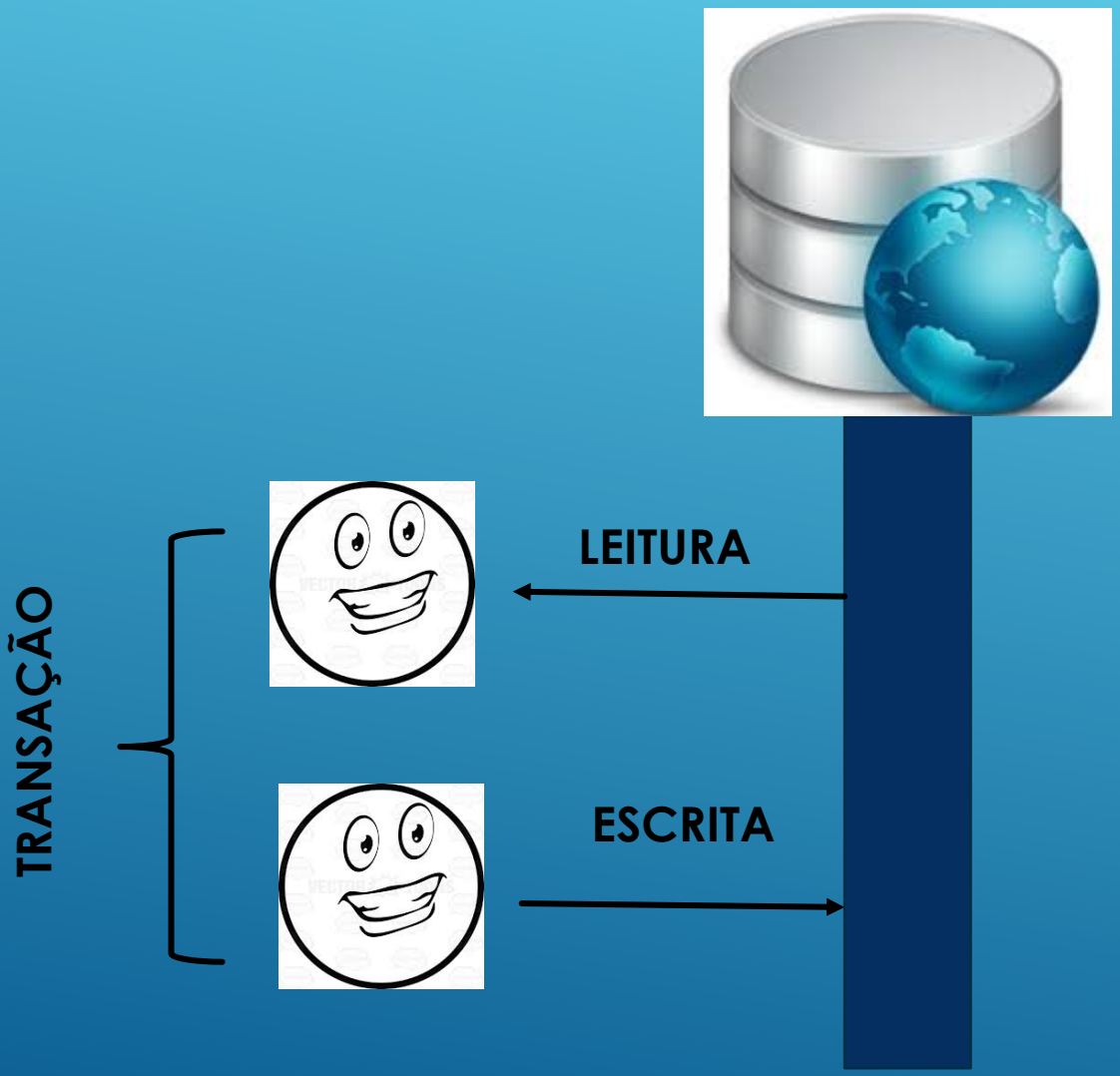
Esse controle é alcançado por meio de uma série de mecanismos chamados **esquemas de controle de concorrência**

Transação

É uma unidade de execução do programa que acessa e possivelmente atualiza itens de dados.

Coleção de operações que formam uma única unidade lógica de trabalho.

Transação



Transação

É importante para proteger os dados

Evita inconsistências

Garante a recuperação de uma falha

Propriedades de uma Transação

ACID

Atomicidade – Totalidade, – ou tudo ou nada

Consistência – Não perder dados no BD

Isolamento - Transações independentes, uma não afeta a outra

Durabilidade – Não afetado por Falhas

Atomicidade

Uma transação precisa terminar com uma efetivação ou reversão

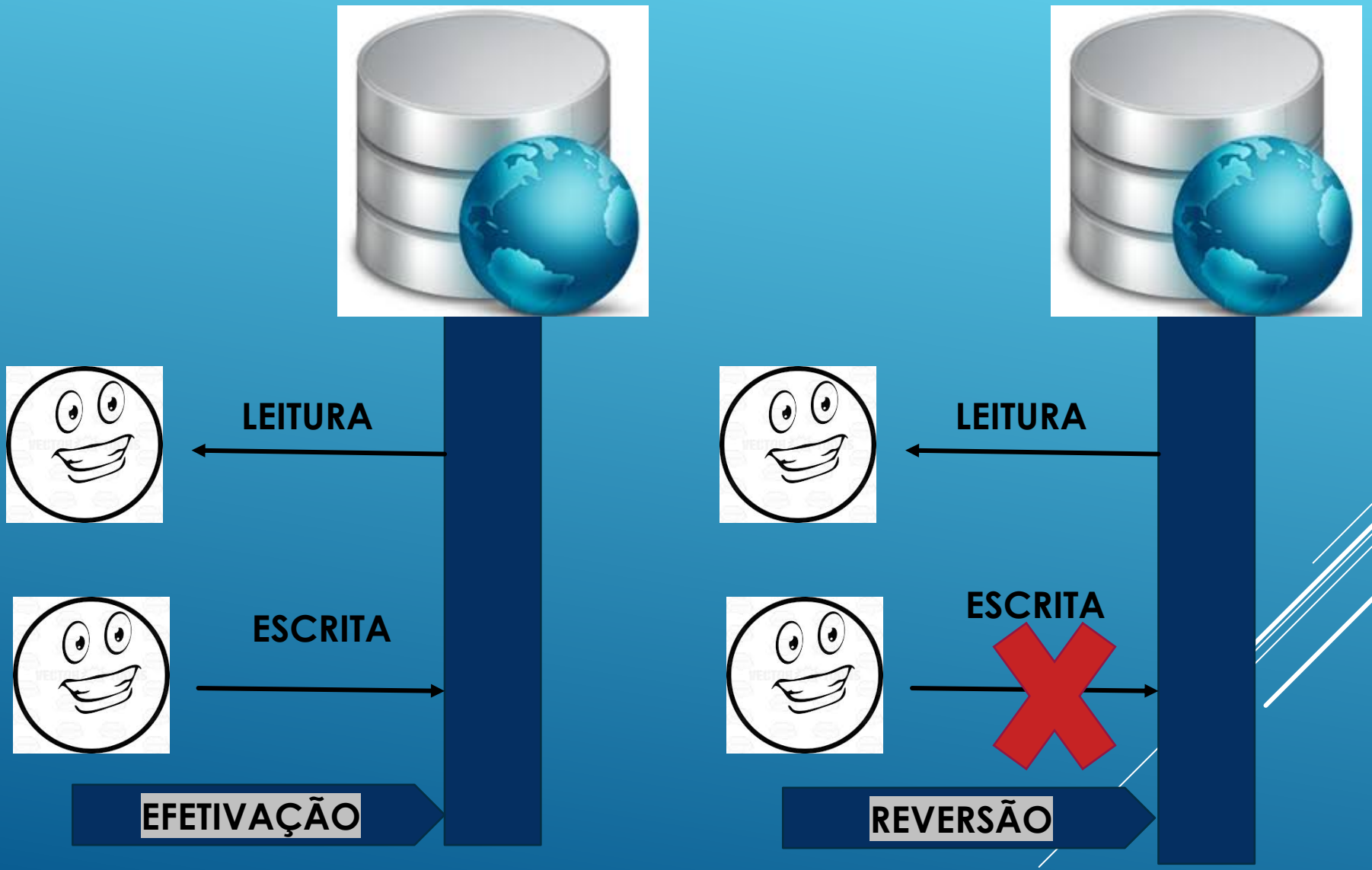
Mantém o BD livre de inconsistências

Todas as ações da transação são concluídas ou são canceladas.

Uma efetivação (COMMIT) **finaliza** uma transação

Uma reversão (ROLLBACK) **cancela** uma transação

Transação



Transação

Em alguns casos, uma efetivação ou reversão são efetuadas automaticamente.

Comandos SQL:

COMMIT – efetiva uma transação

ROLLBACK – reverte uma transação

Transação

READ (A) – transfere o dado A do banco de dados para a memória

WRITE (B) – transfere o dado X da memória para o banco de dados

T1: READ (A)
A = A – 50
WRITE (A)
READ (B)
B = B + 50
WRITE (B)
COMMIT (Implícito muitas vezes)

Consistência

Uma transação não deve gerar erros

Se o banco de dados estava consistente antes da transação, ele precisa continuar consistente depois que a transação se encerra

Ex. Na transação T1 as somas devem estar inalteradas após a execução da transação

Consistência

Quando transações são processadas simultaneamente, mas de uma pode acessar a mesmo dado

Transações deve ser capazes de acessar o mesmo **recurso** (ou dado, tabela, tupla) **simultaneamente** sem criar inconsistências

Consistência – acesso concorrente

Vários usuários atualizam os dados simultaneamente

Ex. Acesso concorrente para transferir fundos da conta A para B

T1

Leia (A)
A = A – 50
Grave(A)

T2

Leia (A)
A = A – 100
Grave(A)

Situação

T1 A = 500
T2 A = 500
T1 A = A – 50
T1 A = 450
T2 (A = 500)
T2 A = A – 100
T2 A = 400

Valor correto (A = 350)

Isolamento

Serialização:
É Quando duas ou mais transações simultâneas produzem os mesmos resultados quando processadas em momentos diferentes

Essa propriedade exige que as transações sejam serializáveis como proteção contra erros

É preciso ter controle sobre as transações.

O uso de travas (lock) é o método mais comum

Isolamento - travas

Trava **compartilhada** é usada quando se **leem** os dados

Trava **exclusiva** é usada quando se **gravam** dados

Isolamento - travas

Trava Compartilhada em uso

- Usuário pode aplicar uma trava **compartilhada** em outras transações

- Usuário **não** pode aplicar uma trava **exclusiva**

Isolamento - travas

Trava exclusiva em uso

- Usuário pode aplicar uma trava **compartilhada** em outras transações

- Usuário **não** pode aplicar nem uma trava **compartilhada** nem uma trava **exclusiva**

Isolamento - travas

Matriz de Coexistência entre os tipos de trava

	Trava Compartilhada	Trava Exclusiva
Trava Compartilhada	SIM	NÃO
Trava Exclusiva	NÃO	NÃO

Trava Compartilhada – lock-S (A)

Trava Exclusiva - lock-X (A)

SEJAM: A = 100, B = 200

T1	T2	Gerencidador de controle de concorrência
LOCK-X (B) - Exclusiva		Grant-x (B, T1)
READ (B)		
B = B - 50		
WRITE (B)		
UNLOCK (B)		
	LOCK-S (A) - Compartilhada	GRANT-S (A, T2)
	READ(A)	
	UNLOCK (A)	
	LOCK-S(B)	GRANT-S(B,T2)
	READ (B)	
	UNLOCK (B)	
	DISPLAY (A+B)	(A+B = 250)
LOCK-X (A)		GRANT-X (A, T2)
READ (A)		
A = A + 50		
WRITE (A)		
UNLOCK (A)		
	DISPLAY (A+B)	(A+B = 300)

RESULTADO: A = 150, B = 150

Travas

Para que as transações sejam serializáveis, Precisamos obedecer certas regras para aplicar em cada transação

Uma das regras é o **travamento em duas fases**

Fase 1: Aplicar travas
Fase 2: Remover travas

Transação

T1

TRAVA (A)
TRAVA(B)
LER (A)
LER (B)
ESCREVE (A)
ESCREVE(B)
DESTRAVA(A)
DESTRAVA (B)

T2

TRAVA (A)
LER (A)
ESCREVE (A)
DESTRAVA(A)

TRAVA (B)
LER (B)
ESCREVE(B)
DESTRAVA (B)

As travas foram removidas antes que as escritas tenham sido feitas

Granularidade de Travas

Existem diversos recursos que podem ser travados (bloqueados)

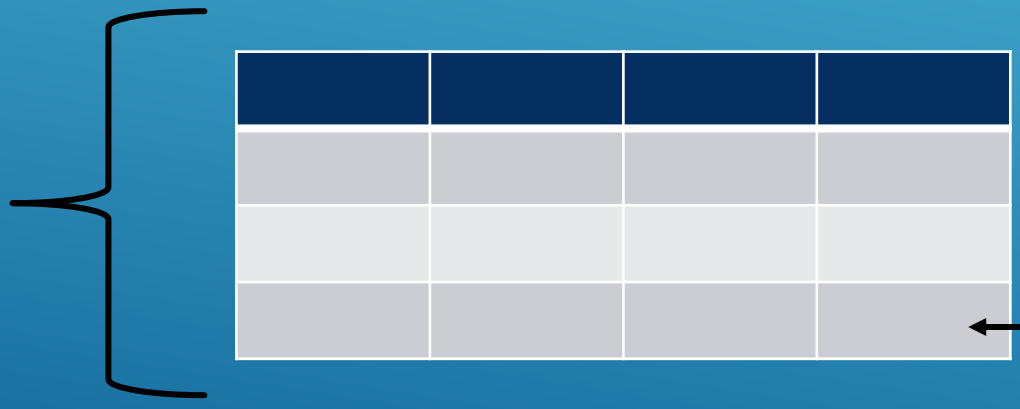
Pode-se travar tabelas, linhas como unidade

A extensão do travamento é conhecida como **granularidade**

Granularidade de Travas

Granularidade grossa – quando muitos recursos são travados ao mesmo tempo

Granularidade fina – poucos recursos travados

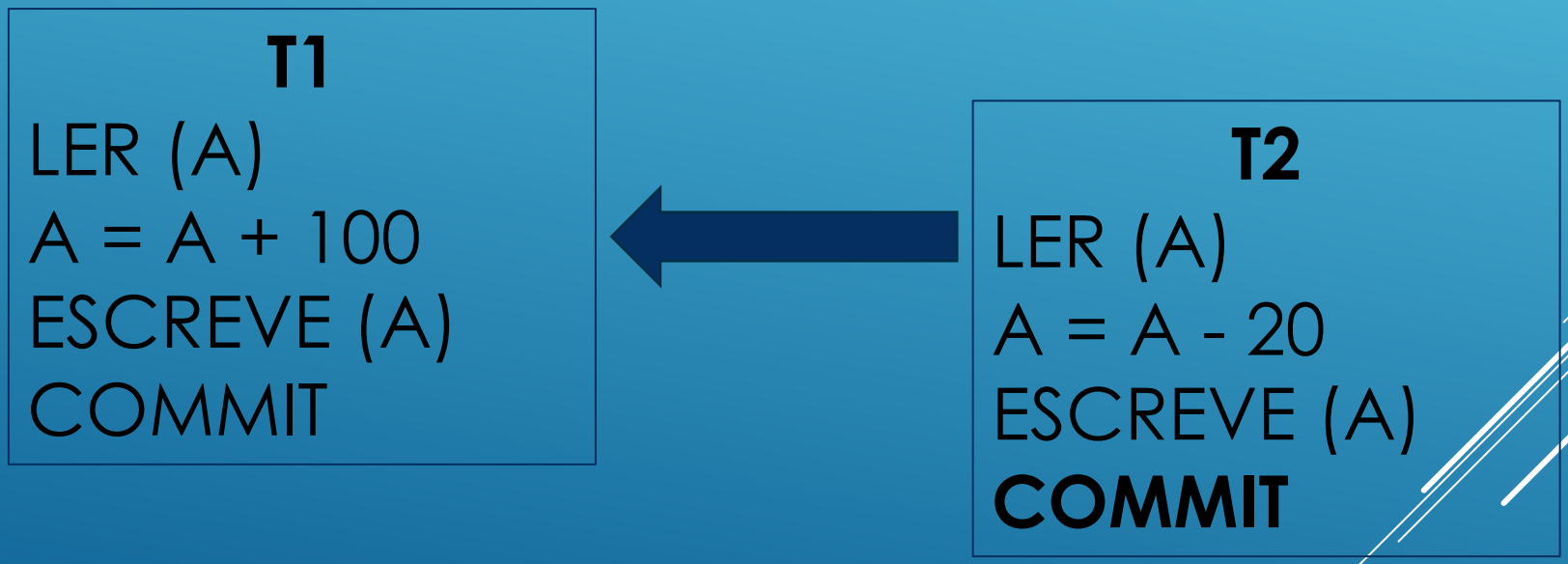


Granularidade grossa (travar tabela)

Granularidade fina (travar linha, tupla)

LEITURA SUJA

O problema advém do fato de **T2 ter lido um valor de T1** que ainda não havia sido confirmado (commit)

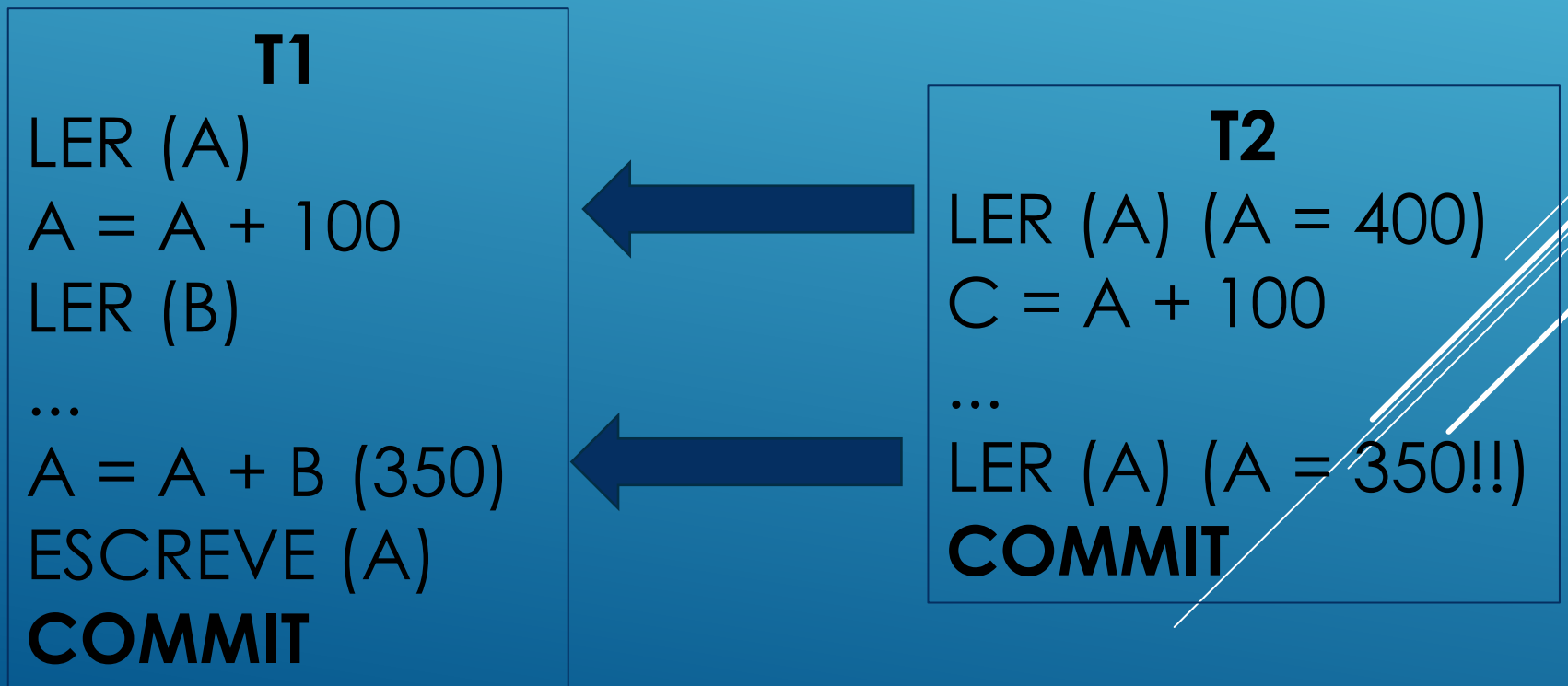


O valor de A ainda não foi confirmado em T1 !
Pode gerar inconsistência de valores

LEITURA NÃO REPRODUZÍVEL (REPETITIVA)

Uma transação lê os mesmos valores duas vezes
Encontra um **valor diferente** da **segunda vez**

A = 300, B = 50



LEITURA FANTASMA

Uma transação reexecuta uma consulta

A transação descobre que o conjunto de linhas que satisfazem a **condição mudou**
Ex. conta(num,saldo)

t20
Select sum (saldo)
from Conta
Where
nome_Agencia =
“campina”

t21
Inserto into conta
Values (233, 1.000)

T20 e t21 não acessam qualquer tupla em comum
Mas entram em conflito um com o outro!!!

NÍVEIS DE ISOLAMENTO

Pode-se configurar os o nível em que transações serão processada simultaneamente

Na linguagem SQL, usamos o comando
SET TRANSACTION (Configurar transação)

O comando SET TRANSACTION

Usado para colocar o nível de isolamento e operações permitidas (R/W).

Sintaxe:

SET TRANSACTION <acesso>, <isolamento>

Onde: <acesso> : READ ONLY | READ WRITE

<isolação>: READ UNCOMMITED
READ COMMITED
REPEATABLE READ
SERIALIZABLE

O comando SET TRANSACTION

Dependendo do nível de isolamento, os seguintes problemas podem ocorrer:

	Leitura suja	Leitura não reproduzível	Leitura fantasma
READ UNCOMMITED	Possível	Possível	Possível
READ COMMITED	Não ocorre	Possível	Possível
REPEATEBLE READ	Não ocorre	Não ocorre	Possível
SERIALIZABLE	Não ocorre	Não ocorre	Não ocorre

Leitura suja: uma segunda transação lê um valor antes que a primeira
Leitura não reproduzível: uma transação lê o mesmo valores, com valores diferentes
Leitura fantasma: uma transação procura linha, mas encontra linhas erradas feitas por outra transção

O comando SET TRANSACTION

Exemplos:

1) SET TRANSACTION READ ONLY,
ISOLATION LEVEL READ UNCOMMITED

Indica que nenhum update será permitido por comandos SQL executados como parte da transação.
É o nível mais baixo de isolação

2) SET TRANSACTION READ WRITE,
ISOLATION LEVEL SERIALIZABLE

Permite updates durante a transação como também consultas.
É o nível de isolação mais alto

Durabilidade

Um banco de dados gerencia dados importantes

Garante a segurança e a durabilidade no caso de falhas

Pode-se configurar permissões relativas a quem pode acessar todo o banco de dados ou as tabelas contidas.

Durabilidade/Segurança - Comando Grant

Grant – concede permissões para que outros usuários possam efetuar processamento nas tabelas

```
GRANT SELECT, UPDATE ON PRODUTOS
    TO Departamento

CREATE USER Diana  IDENTIFIED BY ‘senha';
GRANT ALL ON aulas.* TO Diana;
GRANT SELECT ON aulas.alunos TO Jaqueline;
GRANT USAGE ON *.* TO Gabriel WITH
MAX_QUERIES_PER_HOUR 90;
```

Durabilidade/segurança - Comando Grant

Comando	Resultado
SELECT	Permite que usuários pesquisem linhas numa tabela
INSERT	Permite que usuários acrescentem linhas numa tabela
UPDATE	Permite que usuários atualizem linhas numa tabela
DELETE	Permite que usuários apaguem linhas numa tabela
ALL	Concede todos os privilégios

BANCO DE DADOS

SISTEMA DE RECUPERAÇÃO

Prof. Dr. Vladimir Costa de Alencar

valencar.com

vladimir.uepb@gmail.com

Sistema de Recuperação

Um banco de dados precisa ter um mecanismo para proteger os dados em caso de falha

Para garantir a durabilidade das transações, é obrigatório que nenhuma falha possa criar dados incorretos

Para proteger a si mesmo de falhas, um BD executa várias operações, o que inclui cópias de segurança e logs de transação

Tipos de Falhas

Falhas na Transação

Falhas no Sistema

Falhas de Mídia

FALHAS NA TRANSAÇÃO

Erro lógico
A transação não pode mais continuar devido a alguma condição interna:

- Entrada defeituosa
- Dados não encontrados
- Estouro
- Limites de recursos ultrapassados

Erro do Sistema

- Deadlock

FALHAS NO SISTEMA

Defeito de hardware ou um bug de software de banco de dados ou no sistema operacional

O sistema para, mas não corrompe o conteúdo do banco de dados

- System Crash, Deadlock
- Bugs no código

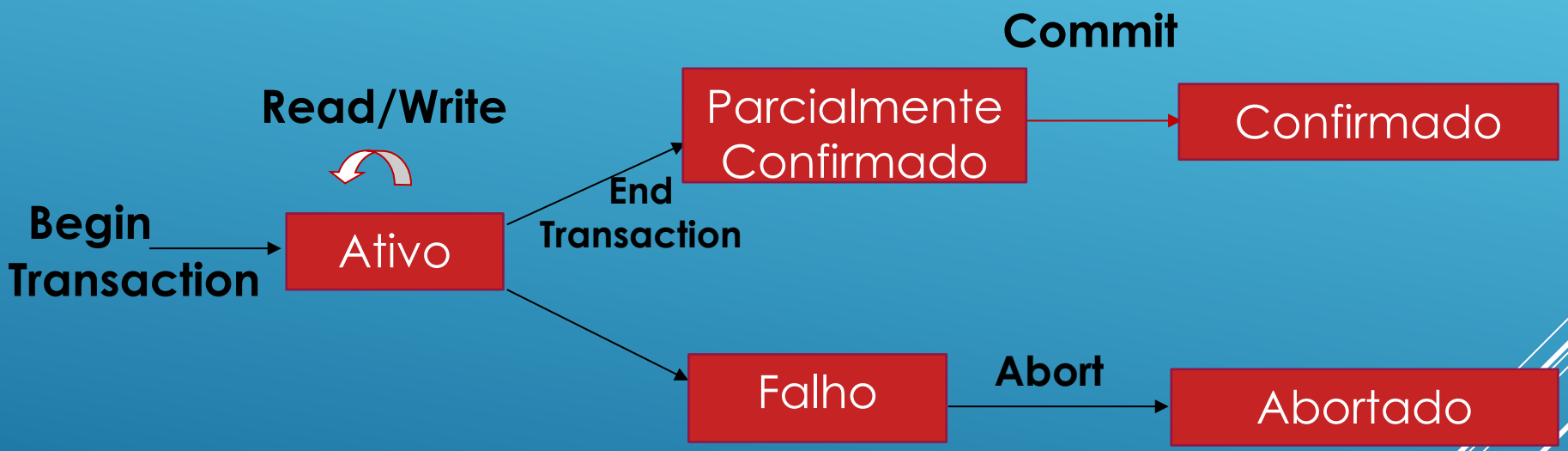
FALHAS DE MÍDIA

Falha no HD

Falha durante a transferência de dados

- Backups são usados para a recuperação de falhas (fitas, HD, etc)

Diagramas de estado de uma transação



Recuperação e atomicidade

Ex. Tranferencia bancaria (A = 1.000, B = 2.000)

T1

LER (A)
LER (B)
A = A – 500
B = B + 500
ESCREVE (A)
ESCREVE (B)
COMMIT

FALHA

Como o conteúdo da memória foi perdido, não conhecemos o destino da transação

Recuperação e atomicidade

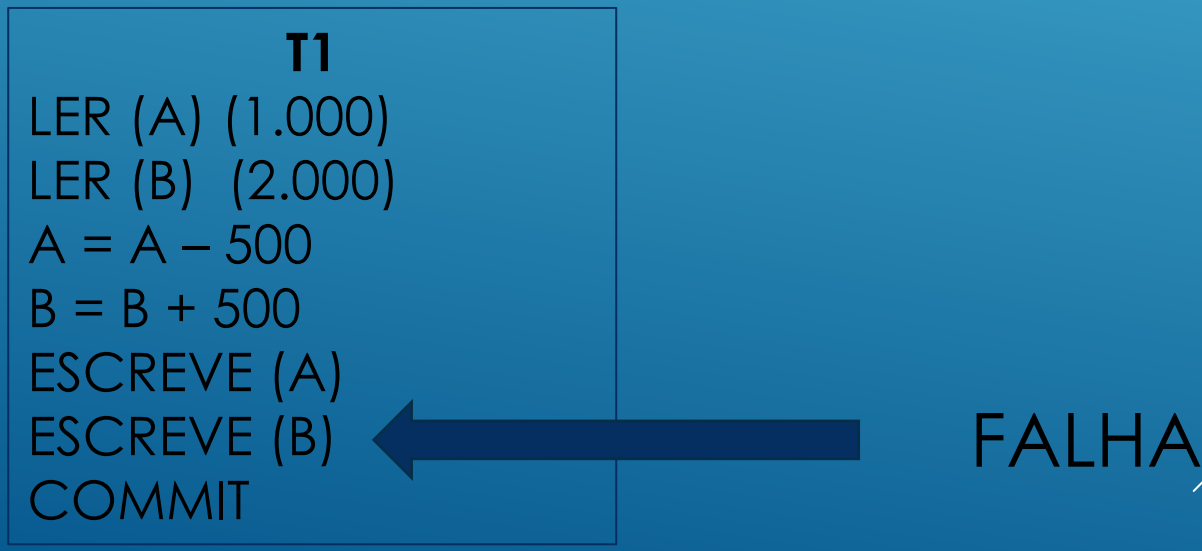
Para recuperar a transação T1, podemos invocar:

- **Executar T1 novamente**

Resultado: A = 0 (Estado inconsistente)

- **Não executar T1 novamente**

Resultado: A = 500, B = 2.500 (Estado Inconsistente)



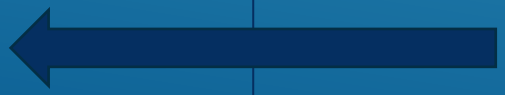
Recuperação e atomicidade

Nos dois casos, o banco de dados entra num estado **inconsistente**.

Modificamos o banco de dados **sem ter certeza** de que a transação será **confirmada**

Exitem formas de recuperação para alcançar a **atomicidade**

T1
LER (A) (1.000)
LER (B) (2.000)
A = A – 500
B = B + 500
ESCREVE (A)
ESCREVE (B)
COMMIT



FALHA

Recuperação baseada em log

A Estrutura mais utilizada para registrar as informações é o **log**.

O log é uma sequência de registros contendo todas as atividades de atualização no BD

Um registro de log de atualização descreve uma única escrita no banco de dados

Recuperação baseada em log

É um arquivo que mantém todas as operações que as transações efetuaram no BD

É usado para recuperar o sistema na ocorrência de falhas

É mantido em disco e em fita (backup)

Recuperação baseada em log

A Estrutura do registro de log:

- Identificador de transação
- Identificador do item de dados
- Valor Antigo
- Valor Novo
- Ex. <T2, WRITE (A), 100, 200>

Recuperação baseada em log

Exemplo de log:

<T1, start> - a transação foi iniciada

<T1, WRITE(X), Valor_Antigo, Valor_Novo>
- a transação T1 realizou uma escrita sobre o item de dados X
V1 = valor antes da escrita
V2 = valor após a escrita

<T1, commit> - a transação T1 foi confirmada

<T1, abort> - a transação T1 foi abortada

Modificação do Banco de dados adiada

Garante a atomicidade da transação

Registra todas as atividades no log

Adia a execução de todas as **operações WRITE** de uma transação até que a transação seja parcialmente confirmada

Modificação do Banco de dados adiada

Todos os **Writes** serão atrasados até que **Ti** confirme parcialmente

Todos os **Writes** serão gravados num arquivo de log

Quando **commit** ocorre → BD é atualizado

Modificação do Banco de dados adiada

Log: mantém todas as atualizações do BD. Entradas no log para cada **id, transaction**.

1. <Ti, início>
2. < Ti, Write, X, Valor.novo >
3. < Ti, commit >

- Usando o log, o sistema pode resolver qualquer falha que não danifique o BD físico.

Modificação do Banco de dados adiada

Exemplo: Sejam duas transações

T1: transferência de fundos de $50 da conta A para B

T2 : saque de $100 da conta C

Implementadas como segue:

T1: Begin_Transaction_T1	T2: Begin_Transaction_T2
Read(A, a1);	Read(C, c1);
a1 := a1 – 50;	c1 := c1 – 100;
Write (A, a1);	Write (C, c1);
Read (B, b1);	End_Transaction_T2;
b1 := b1 + 50;	
Write (B, b1);	
End_Transaction_T1;	

Valores inicias das contas: A=$1000, B=$2000, C=$700.

Suponha que T2 é executada depois de T1.

Modificação do Banco de dados adiada

Log para execução completa de **T1** e **T2**:

< T1 , Início >

< T1 , A, 950 >

< T1 , B, 2050 > **A = 950, B = 2050**

< T1 , commit >

< T2 , Início >

< T2 , C, 600 >

< T2 , commit > **C = 600**

Modificação do Banco de dados adiada

Vamos simular alguns casos de falhas:

1) Falha ocorre após **< T1, B, 2050 >** ser gravado no log. O log na hora da falha é:

< T1 , Início >

< T1 , A, 950 >

< T1 , B, 2050 >

- No restart não precisa recuperação, pois não há commit.

Modificação do Banco de dados adiada

2) Falha ocorre após **< T2, C, 600 >** ser gravado no log. Log na hora da falha:

< T1 , Início >

< T1 , A, 950 >

< T1 , B, 2050 >

< T1 , commit >

< T2 , Início >

< T2 , C, 600 >

- Quando restart ocorre → **REDO (T1) = refaz T1**

Modificação do Banco de dados adiada

3) Falha ocorre após **< T2, commit >** ser gravado no log.
Log na hora da falha:

< T1 , Início >

< T1 , A, 950 >

< T1 , B, 2050 >

< T1 , commit >

< T2 , Início >

< T2 , C, 600 >

< T2 , commit >

- Quando restart ocorre → **REDO (T1)** e **REDO(T2)**

Modificação do Banco de dados adiada

4) Segunda falha ocorre durante recuperação da 1ª falha.

Algumas mudanças podem ter sido feitas no BD como consequência de **REDO**, mas todas as mudanças foram feitas.

Solução: executar o algoritmo de recuperação novamente.

Pontos de Verificação (Checkpoints)

Motivação:

Quando ocorrer uma falha, o sistema não precisa pesquisar todo o log, nem refazer transações que não precisem de **REDO**

Diminuir overhead de recuperação

Solução: Checkpoints

Periodicamente, um registro de checkpoint é gravado no log

Pontos de Verificação (Checkpoints)

Cada gravação requer as seguintes ações:

1. Forçar a gravação do LOG em disco
2. Gravar os buffers do BD em disco
3. Gravar um registro de checkpoint no LOG
4. Gravar o endereço do último checkpoint num arquivo de restart.

Ex.: Seja um conjunto de **Tis** $\{T_0, T_1 , \ldots , T_{100}\}$ executando na ordem dos índices. Suponha que ocorreu um checkpoint (último) durante a transação $\mathbf{T_{68}}$.

Recuperar {Ti / 68 <= i <= 100}

BANCO DE DADOS

OTIMIZAÇÃO DE CONSULTA

Prof. Dr. Vladimir Costa de Alencar

valencar.com

vladimir.uepb@gmail.com

Otimização de Consulta

É o processo de selecionar o plano de avaliação de consulta mais eficiente

O processo ocorre no nível da álgebra relacional

Otimização de Consulta

Da álgebra relacional

Seleção(σ - sigma)

$$\boldsymbol{\sigma}_{Salário>1.500}(Funcionário)$$

Projeção (π)

$$\boldsymbol{\pi}_{\text{Matricula, Nome}}(Funcionário)$$

Otimização de Consulta

Linguagens de alto nível (ex.SQL) podem ter consultas com alto tempo de processamento .

Existem várias maneiras de escrever uma query.

Solução: usar um otimizador de consultas para escolher a melhor forma, dentro das suas possibilidades, de executar uma consulta

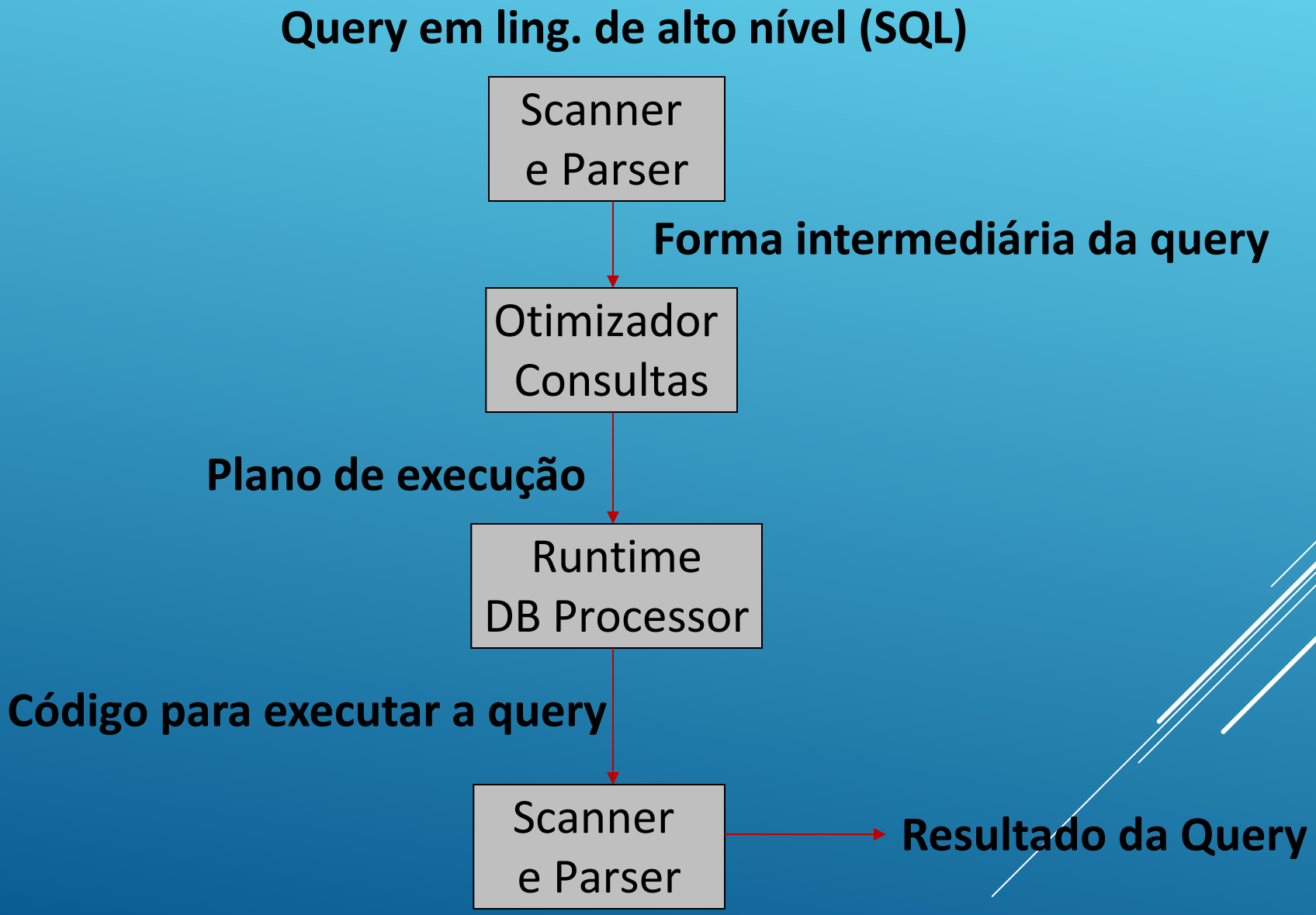
Otimização de Consulta

O sistema gera um código otimizado (sem necessariamente ser o melhor possível)

O processo de otimização leva em consideração:

- A teoria da Álgebra Relacional
- Informações sobre:
  - tamanhos de tabelas
  - seletividade de índices
  - agrupamento de dados

Otimização de Consulta



Otimização de Consulta

Quais operações levam mais tempo para executar?

- **Produto Cartesiano e Junção**

Otimização de Consulta

Ex. A importância de se reduzir o tamanho das tabelas intermediárias

Sejam **A(a1,a2)** e **B(b1,b2)** esquemas relacionais

```
SELECT a.a1
FROM A a, B b
WHERE a.a1 = 'valor' AND a.a2 = b.b1
```

Otimização de Consulta

Estratégia 1:

1) Produto Cartesiano
2) Seleção
3) Projeção

⇒ **Completamente inviável!**

⇒ Só o produto cartesiano gera um resultado intermediário de tamanho muito grande

Otimização de Consulta

Estratégia 2:

1) Seleção de a1 = ‘valor’

2) (A x B) a.a2 = b.b1

3) Projeções são feitas em cada passo

Otimização de Consulta

Conclusão:

Operações que reduzem tabelas:

Seleção (where idade >= 18)

Projeção (todas as tuplas que tenham o atributo Saldo)

Otimização de Consulta

Estratégias gerais de Otimização:

1. Execute seleções o mais rápido possível
   ⇒ Reduz o tamanho das tabelas (resultados intermediários)

2. Combine, quando possível, uma seleção com o produto cartesiano anterior formando uma junção

3. Combine sequências de operações unárias (seleção e projeção)

4. Procure subexpressões comuns e guarde-as caso seja mais eficiente lê-las do que reprocessá-las