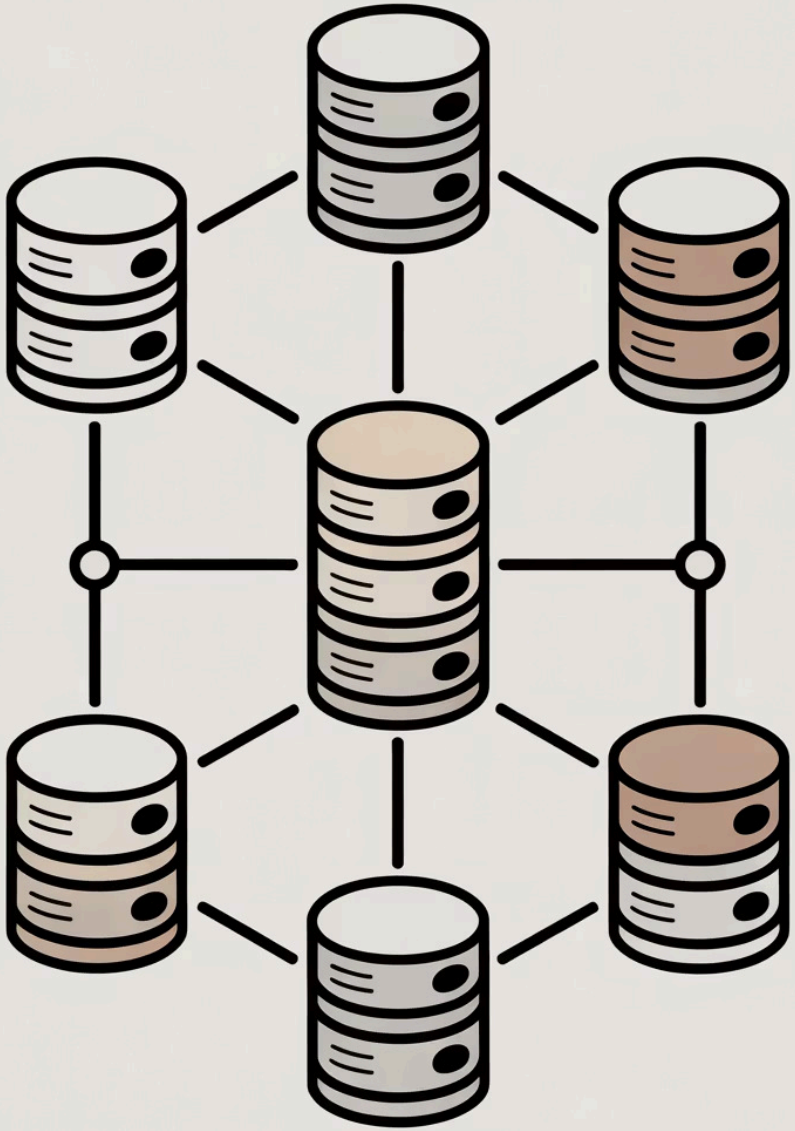


CEFET/RJ

# Sistema de Banco de Dados Distribuído

Um sistema de banco de dados distribuído representa uma arquitetura moderna onde múltiplos sites geograficamente dispersos trabalham de forma coordenada, mantendo autonomia operacional. Esses sites são caracterizados por um acoplamento fraco, o que significa que não compartilham componentes físicos como memória ou processadores.

**Eduardo Ogasawara**

eduardo.ogasawara@cefet-rj.br
https://eic.cefet-rj.br/~eogasawara

# Bancos de Dados Distribuídos Homogêneos

## Sistemas Homogêneos

Em um banco de dados distribuído homogêneo, todos os sites possuem software idêntico e estão plenamente cientes um do outro. Eles concordam em cooperar ativamente no processamento de solicitações do usuário.

Cada site entrega parte de sua autonomia em termos de direito de mudar esquemas ou software, criando um ambiente padronizado. O sistema aparece ao usuário final como um único sistema unificado, abstraindo a complexidade da distribuição.

## Sistemas Heterogêneos

Em contraste, um banco de dados distribuído heterogêneo permite que diferentes sites utilizem esquemas e software distintos. A diferença no esquema representa um grande desafio para o processamento de consultas.

A diferença no software é um problema significativo para o processamento de transações. Os sites podem não estar cientes um do outro e geralmente oferecem apenas facilidades limitadas para cooperação no processamento de transações distribuídas.

# Armazenamento de Dados Distribuído

No contexto do modelo de dados relacional, existem duas estratégias fundamentais para organizar dados em sistemas distribuídos: replicação e fragmentação. Essas abordagens podem ser utilizadas individualmente ou de forma combinada para otimizar o desempenho e a confiabilidade.

## Replicação

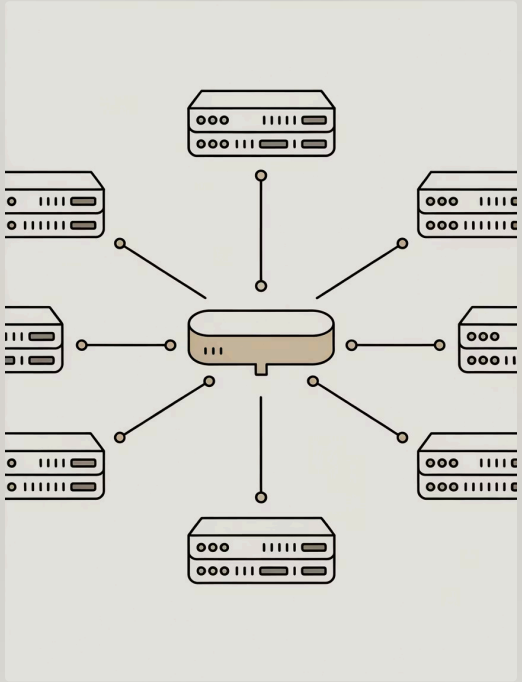
O sistema mantém múltiplas cópias idênticas dos dados, armazenadas em diferentes sites. Isso proporciona recuperação mais rápida em caso de falhas e melhora significativamente a tolerância a falhas do sistema.

## Fragmentação

A relação é particionada em vários fragmentos menores, cada um armazenado em sites distintos. Isso permite distribuir a carga de trabalho e aproximar os dados dos locais onde são mais frequentemente acessados.

## Abordagem Híbrida

Replicação e fragmentação podem ser combinadas de forma inteligente. A relação é primeiro particionada em fragmentos, e então o sistema mantém múltiplas réplicas idênticas de cada fragmento, maximizando disponibilidade e desempenho.



# Replicação de Dados

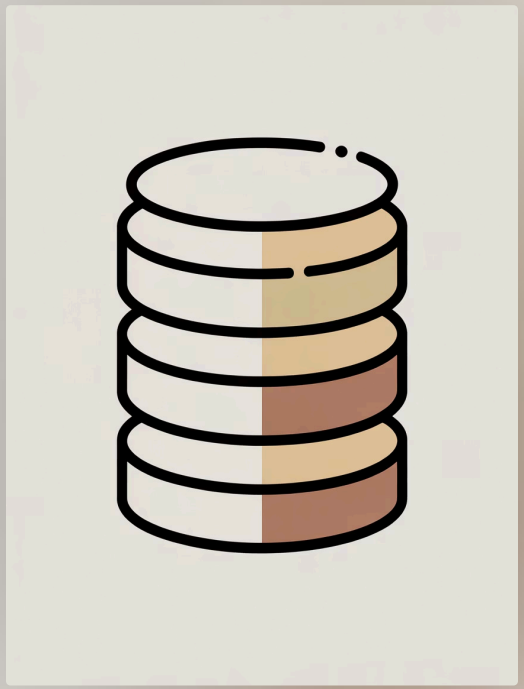
A replicação de dados é uma estratégia essencial em sistemas distribuídos que visa aumentar a disponibilidade e o desempenho. Quando múltiplas cópias dos dados são mantidas em diferentes sites, o sistema ganha resiliência contra falhas e permite que consultas sejam executadas localmente.

## Vantagens da Replicação

- Maior disponibilidade dos dados
- Melhor desempenho de leitura
- Tolerância a falhas de sites individuais
- Redução da latência de acesso

## Desafios da Replicação

- Necessidade de sincronização entre réplicas
- Maior uso de espaço de armazenamento
- Complexidade em operações de atualização
- Garantia de consistência entre cópias



# Fragmentação de Dados

A fragmentação divide uma relação em partes menores distribuídas por diferentes sites. Existem duas abordagens principais: fragmentação horizontal, que divide as tuplas (linhas) da relação, e fragmentação vertical, que divide os atributos (colunas).

## Fragmentação Horizontal

Divide a relação em subconjuntos de tuplas, mantendo todos os atributos em cada fragmento. Cada site armazena um grupo específico de linhas.

## Fragmentação Vertical

Divide a relação por atributos, mantendo todas as tuplas, mas apenas algumas colunas em cada fragmento. Um identificador de tupla permite reconstruir a relação completa.

## Fragmentação Mista

Combina fragmentação horizontal e vertical, aplicando-as sucessivamente a qualquer profundidade necessária para otimizar o armazenamento e acesso.

# Vantagens da Fragmentação

## Fragmentação Horizontal

A fragmentação horizontal oferece benefícios significativos para o processamento distribuído. Ela permite o processamento paralelo sobre fragmentos de uma relação, distribuindo a carga computacional entre múltiplos sites.

Além disso, possibilita que uma relação seja dividida de modo que as tuplas sejam localizadas onde são acessadas com mais frequência, reduzindo drasticamente a latência de acesso e o tráfego de rede entre sites.

| nomeAgencia | numeroConta | Saldo |
|---|---|---|
| Hillside | A-305 | 500 |
| Hillside | A-226 | 336 |
| Hillside | A-155 | 62 |

$$conta_1 = \sigma_{nomeAgencia="Hillside"}(conta)$$

| nomeAgencia | numeroConta | Saldo |
|---|---|---|
| Valleyview | A-177 | 205 |
| Valleyview | A-402 | 10000 |
| Valleyview | A-408 | 1123 |
| Valleyview | A-639 | 750 |

$$conta_2 = \sigma_{nomeAgencia="Valleyview"}(conta)$$

## Fragmentação Vertical

A fragmentação vertical permite que as tuplas sejam divididas de modo que cada parte seja armazenada onde é acessada com mais frequência, otimizando o acesso a atributos específicos.

O atributo id-tupla permite a junção eficiente de fragmentos verticais quando necessário reconstruir a relação completa. Também permite o processamento paralelo, com diferentes sites processando diferentes conjuntos de atributos simultaneamente.

| nomeAgencia | nomeCliente | id |
|---|---|---|
| Hillside | Lowman | 1 |
| Hillside | Camp | 2 |
| Valleyview | Camp | 3 |
| Valleyview | Kahn | 4 |
| Hillside | Kahn | 5 |

$$infoFuncionario_1 = \pi_{nomeAgencia,nomeCliente,id}(infoFuncionario)$$

| numeroConta | saldo | id |
|---|---|---|
| A-177 | 205 | 1 |
| A-402 | 10000 | 2 |
| A-408 | 1123 | 3 |
| A-639 | 750 | 4 |
| A-747 | 5000 | 5 |

$$infoFuncionario_2 = \pi_{numeroConta,saldo,id}(infoFuncionario)$$

# Transparência de Dados

A transparência de dados refere-se ao grau em que o usuário do sistema pode permanecer desavisado dos detalhes de como e onde os itens de dados são armazenados em um sistema distribuído. Este conceito é fundamental para simplificar a interação do usuário com sistemas complexos.

## Transparência de Fragmentação

O usuário não precisa saber que os dados estão fragmentados em partes menores. Ele interage com uma visão lógica unificada da relação completa.

## Transparência de Replicação

O usuário não precisa estar ciente de que múltiplas cópias dos dados existem. O sistema gerencia automaticamente qual réplica utilizar.

## Transparência de Local

O usuário não precisa saber em qual site físico os dados estão armazenados. Pode acessar qualquer dado sem especificar sua localização.

# Transações Distribuídas

Uma transação distribuída pode acessar dados em múltiplos sites, exigindo coordenação sofisticada para manter as propriedades ACID. A arquitetura envolve componentes especializados em cada site para gerenciar a execução distribuída.

## Gerenciador de Transação Local

Cada site possui um gerenciador de transação responsável por manter um log detalhado para fins de recuperação em caso de falhas. Também participa ativamente na coordenação da execução concorrente das transações em execução nesse site específico.

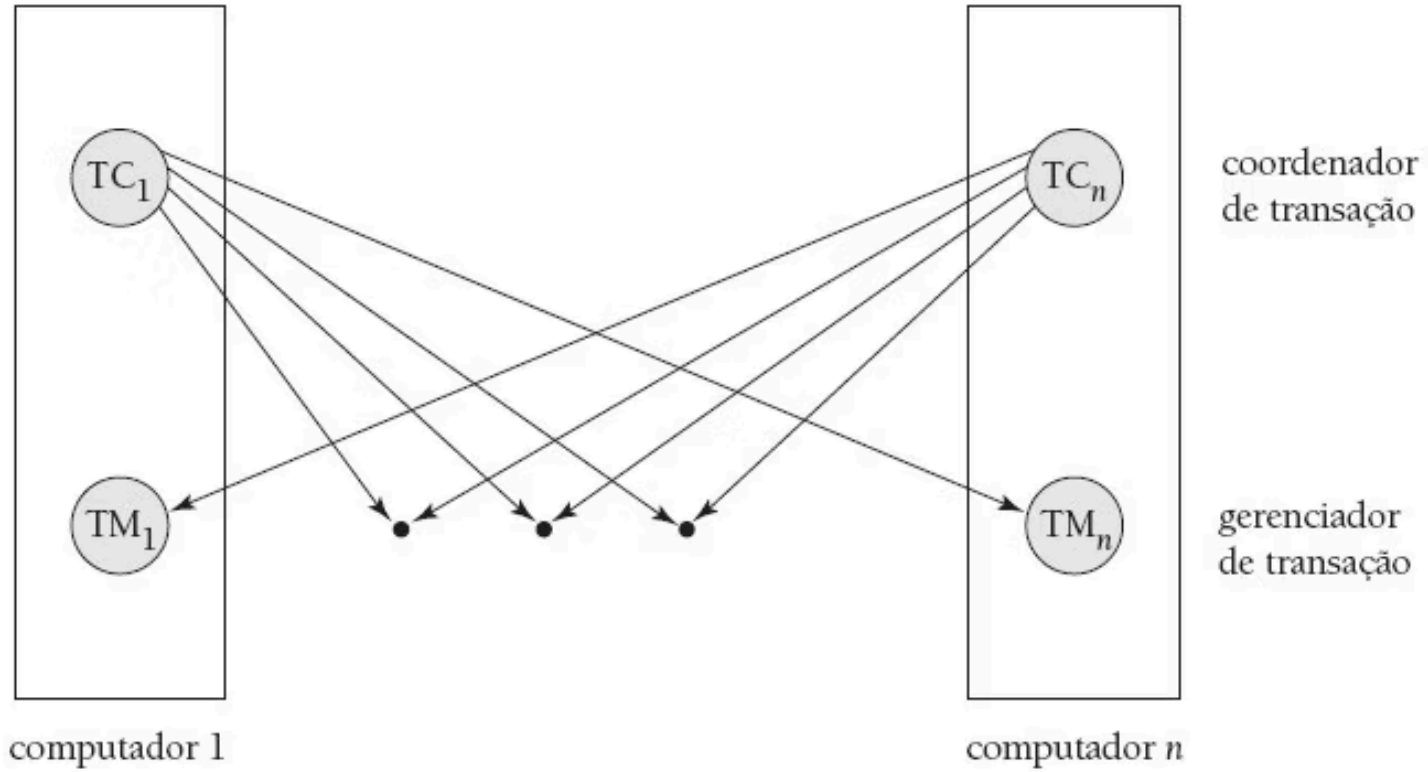
## Coordenador de Transação

Cada site possui um coordenador de transação com responsabilidades abrangentes. Ele inicia a execução das transações que originam no site e distribui sub-transações nos sites apropriados para execução paralela.

O coordenador também gerencia o término de cada transação, garantindo que ela seja confirmada em todos os sites participantes ou abortada neles todos, mantendo a atomicidade global.

# Arquitetura de Sistema de Transação

A arquitetura de um sistema de transação distribuído ilustra a interação entre os diversos componentes. O diagrama mostra como o coordenador de transação se comunica com os gerenciadores de transações locais em cada site, orquestrando a execução distribuída e garantindo consistência por meio de protocolos de commit.



# Falhas Exclusivas aos Sistemas Distribuídos

Sistemas distribuídos enfrentam tipos específicos de falhas que não ocorrem em sistemas centralizados. Compreender essas falhas é essencial para projetar mecanismos de recuperação robustos.

## Falha de um Site

Um site individual pode falhar completamente, tornando-se inacessível. O sistema deve continuar operando com os sites restantes, garantindo disponibilidade dos dados por meio de réplicas.

## Falha de Enlace de Comunicação

Links de comunicação específicos podem falhar, interrompendo rotas de comunicação. Protocolos de rede lidam com isso roteando mensagens por meio de enlaces alternativos disponíveis.

## Perda de Mensagens

Mensagens podem ser perdidas durante a transmissão na rede. Esse problema é geralmente tratado por protocolos de controle de transmissão de rede, como TCP-IP, que garantem entrega confiável.

## Partição de Rede

Uma rede é considerada particionada quando é dividida em dois ou mais subsistemas sem conexão entre eles. Cada subsistema pode consistir em um ou mais nós, e particionamento geralmente não é distinguido de falhas de site.

# Protocolos de Commit

Protocolos de commit são mecanismos fundamentais usados para garantir a atomicidade entre os sites em transações distribuídas. Uma transação executada em múltiplos sites precisa ser confirmada em todos os sites ou abortada neles todos - não é aceitável ter confirmação parcial.

## Protocolo 2PC

O protocolo de commit em duas fases (2PC) é amplamente utilizado na indústria. Ele divide o processo de commit em duas fases distintas: preparação e commit, garantindo que todos os sites concordem antes da confirmação final.

## Protocolo 3PC

O protocolo de commit em três fases (3PC) é mais complexo e dispendioso que o 2PC, mas evita algumas de suas desvantagens. Adiciona uma fase intermediária que permite melhor tratamento de falhas e evita bloqueios indefinidos.

# Protocolo de Commit em Duas Fases (2PC)

O protocolo 2PC é o método mais comum para coordenar commits distribuídos. Ele opera em duas fases distintas, cada uma com responsabilidades específicas para garantir que a decisão de commit ou abort seja tomada de forma atômica em todos os participantes.

Considere que **T** seja uma transação iniciada no site $S_i$ e considere que o coordenador de transação em $S_i$ seja $C_i$

### Preparação

O coordenador envia uma solicitação de preparação para todos os participantes. Cada participante verifica se pode confirmar e responde com voto positivo ou negativo.

1

2

### Decisão

Com base nos votos, o coordenador decide por commit (se todos votaram positivo) ou abort (se algum votou negativo) e notifica todos os participantes.

Este protocolo garante que todos os sites cheguem à mesma decisão final, mantendo a consistência global do banco de dados distribuído.

# Fase 1: Obtendo uma Decisão

A primeira fase do protocolo 2PC é crítica para coletar os votos de todos os participantes e determinar se a transação pode ser confirmada com segurança.

01

## Envio de Prepare

- O coordenador pede a todos os participantes para que se preparem para confirmar a transação $T_i$.
- $C_i$ acrescenta os registros <prepare **T**> ao log e força o log para o armazenamento estável
- envia mensagens prepare **T** a todos os sites em que T foi executado.

02

## Preparação Local

- §Ao receber a mensagem, o gerenciador de transação no site determina se pode confirmar a transação
- §se não, acrescente um registro <no **T**> ao log e envie mensagem abort T para $C_i$
- §se a transação puder ser confirmada, então:
- §acrescente o registro <ready **T**> ao log
- §force todos os registros para **T** ao armazenamento estável
- envie mensagem <ready**T**> para $C_i$

# Fase 2: Registrando a Decisão

Na segunda fase, o coordenador comunica a decisão final a todos os participantes, e cada site executa as ações apropriadas para completar ou desfazer a transação.

## Tomada de decisão

T pode ser confirmado se $C_i$ recebeu uma mensagem <ready **T**> de todos os sites participantes: caso contrário, T precisa ser abortados.

## Registro da decisão

O coordenador acrescenta um registro de decisão, <commit **T**> ou <abort T>, ao log e força o registro para o armazenamento estável. Uma vez no armazenamento estável, o registro é irrevogável (mesmo se houver falhas).

## Divulgação da decisão

O coordenador envia uma mensagem a cada participante informando sobre a decisão (commit ou abort).

## Confirmação local

Os participantes tomam a ação apropriada no local.

# Tratamento de Falhas - Falha no Site

Quando um site participante falha durante o protocolo 2PC, o tratamento depende de em qual fase a falha ocorreu e qual o estado registrado no log do site.

### Falha Antes do Voto

Se o participante falha antes de votar, após a recuperação ele verifica seu log e não encontra registro de preparação. Neste caso, ele pode unilateralmente decidir por abort, pois não fez nenhum compromisso.

### Falha Após Votar Sim

Se o participante falha após votar "sim" mas antes de receber a decisão do coordenador, após recuperação ele fica bloqueado esperando a decisão. Deve consultar outros participantes ou aguardar o coordenador se recuperar.

### Falha Após Receber Decisão

Se a falha ocorre após receber a decisão mas antes de completar a ação, após recuperação o site consulta seu log, encontra a decisão e completa o commit ou abort conforme registrado.

# Tratamento de Falhas - Coordenador Falhado

A falha do coordenador é particularmente problemática no protocolo 2PC, pois ele é o único que conhece a decisão final. Os participantes devem cooperar para resolver a situação.

## Estratégias de Recuperação

- Participantes que votaram "não" podem unilateralmente decidir por abort
- Participantes que votaram "sim" ficam bloqueados aguardando a decisão
- Participantes bloqueados podem se comunicar entre si para descobrir a decisão
- Se algum participante já conhece a decisão, pode informá-la aos demais
- Se todos estão bloqueados, devem aguardar a recuperação do coordenador

Este cenário ilustra uma das principais limitações do protocolo 2PC: a possibilidade de bloqueio indefinido quando o coordenador falha após os participantes votarem "sim". O protocolo 3PC foi desenvolvido para resolver este problema específico.

# Tratamento de Falhas - Partição de Rede

Partições de rede criam cenários complexos onde diferentes subconjuntos de sites perdem conectividade entre si, mas continuam operando internamente. O impacto no protocolo 2PC depende de quais sites ficam em cada partição.

## Coordenador e Participantes na Mesma Partição

Se o coordenador e todos os seus participantes permanecem em uma única partição, a falha de rede não tem efeito sobre o protocolo de commit. A transação prossegue normalmente dentro da partição.

## Participantes em Múltiplas Partições

Se o coordenador e seus participantes pertencem a múltiplas partições, os sites que não estão na partição contendo o coordenador interpretam a situação como falha do coordenador e executam o protocolo de tratamento correspondente. Eles ficam bloqueados aguardando a decisão, mas nenhum dano ocorre aos dados.

## Coordenador Isolado de Participantes

Quando o coordenador e os sites estão na mesma partição mas outros participantes ficam isolados, o coordenador percebe a falha dos sites na outra partição e segue o protocolo normal de commit. Se necessário, pode decidir por abort devido à indisponibilidade de sites. Novamente, a consistência é mantida.

# Recuperação e Controle de Concorrência

A recuperação e o controle de concorrência em sistemas distribuídos são mais complexos do que em sistemas centralizados, exigindo coordenação entre múltiplos sites para manter as propriedades ACID das transações.

## Mecanismos de Recuperação

Cada site mantém logs detalhados de transações para permitir recuperação após falhas. Os logs registram operações de undo e redo, checkpoints e decisões de commit. A recuperação coordenada entre sites garante consistência global.

## Controle de Concorrência

Protocolos de bloqueio distribuído ou baseados em timestamp coordenam o acesso concorrente aos dados. Técnicas como bloqueio em duas fases (2PL) são adaptadas para o ambiente distribuído, evitando deadlocks entre sites.

# Controle de Concorrência

O controle de concorrência em sistemas distribuídos adapta esquemas tradicionais para o ambiente distribuído, considerando desafios únicos de comunicação e coordenação entre sites.

## Protocolos de Commit para Atomicidade Global

Considera-se que cada site participa ativamente na execução de um protocolo de commit (como 2PC ou 3PC) para garantir a atomicidade global da transação. Todos os sites devem concordar com o resultado final.

## Consistência de Réplicas

Considera-se que todas as réplicas de qualquer item estão atualizadas e sincronizadas. Protocolos específicos garantem que atualizações sejam propagadas para todas as réplicas de forma consistente, mantendo a integridade dos dados.

## Adaptação de Esquemas Tradicionais

Esquemas de controle de concorrência como bloqueio em duas fases, timestamps e validação otimista são adaptados para uso no ambiente distribuído. Essas adaptações consideram latência de rede, falhas parciais e necessidade de coordenação entre sites.

## Referências





### Elmasri & Navathe

**Fundamentals of Database Systems**

Pearson, 2016

Referência abrangente sobre fundamentos de sistemas de bancos de dados, cobrindo aspectos teóricos e práticos.

### Korth, Sudarshan & Silberschatz

**Database System Concepts**

McGraw-Hill, 2019

Texto fundamental que serviu como base para a maioria dos exemplos apresentados nesta apresentação.

### Özsu & Valduriez

**Principles of Distributed Database Systems**

Springer Nature, 2019

Obra especializada em sistemas de bancos de dados distribuídos, essencial para compreensão avançada.


**Nota:** Os conceitos e exemplos apresentados baseiam-se principalmente na literatura clássica de sistemas de bancos de dados, em especial *Database System Concepts* e *Fundamentals of Database Systems*.