B-Tree (XtraDB)

Samila Rodrigues

21 de agosto de 2025

B-Tree

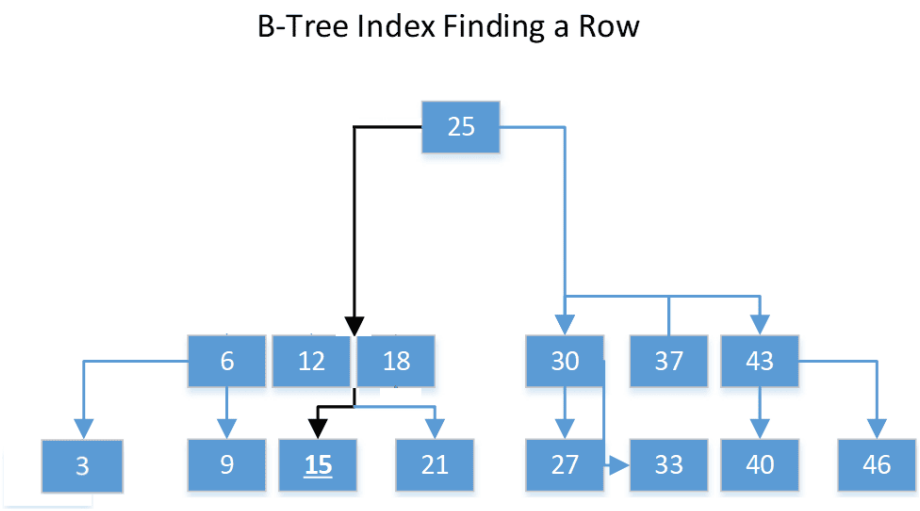


Figura 1: Exemplo de busca em um índice B-Tree. O caminho em preto mostra como a árvore é percorrida até encontrar o valor desejado (neste caso, 15). Cada decisão reduz o espaço de busca, tornando a operação eficiente.

A imagem acima ilustra como um índice B-Tree é utilizado para localizar rapidamente um valor em uma tabela. O algoritmo começa na raiz, segue para o nó intermediário apropriado e, por fim, chega à folha onde está o valor buscado. O caminho percorrido é destacado, mostrando a eficiência da estrutura para buscas.

A imagem acima também mostra a estrutura de um índice B-Tree clusterizado, comum em bancos de dados. O nó raiz referencia nós intermediários, que por sua vez apontam para folhas. As folhas armazenam os dados ordenados e são ligadas entre si, facilitando operações de varredura sequencial (range scan).

B-Tree (Árvore-B) é uma estrutura de dados em árvore balanceada, n-ária, projetada para funcionar eficientemente em sistemas que leem e gravam grandes blocos de dados, como bancos de dados e sistemas de arquivos. Diferente de árvores binárias, cada nó pode conter múltiplas chaves e filhos, reduzindo a altura da árvore e, consequentemente, o número de acessos ao disco.

Características Principais

- **Balanceamento automático:** Todas as folhas estão sempre no mesmo nível, garantindo desempenho consistente.
- **Nós com múltiplas chaves:** Cada nó armazena até $2t - 1$ chaves (onde t é o grau mínimo), permitindo que cada nó tenha entre t e $2t$ filhos.
- **Operações eficientes:** Busca, inserção e remoção são realizadas em tempo $O(\log n)$.
- **Otimização para disco:** Como cada nó pode armazenar muitos registros, minimiza-se o número de leituras/escritas em disco.

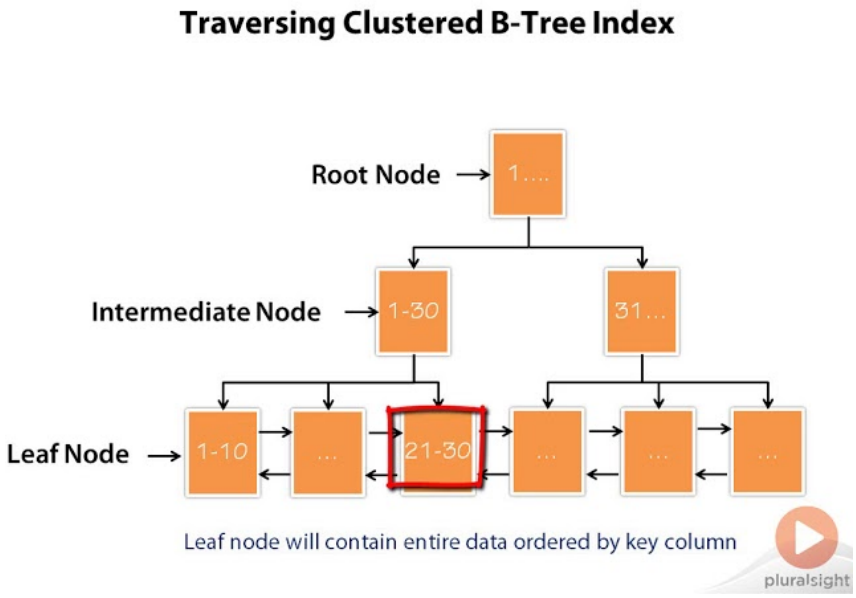


Figura 2: Representação visual de um índice B-Tree clusterizado. Mostra a hierarquia entre nó raiz, nós intermediários e folhas, além da ligação entre folhas (útil para buscas por faixa).

- **Reorganização dinâmica:** Durante inserções e remoções, os nós podem ser divididos ou fundidos para manter as propriedades da árvore.

Busca: A busca em uma B-Tree é semelhante à busca binária, mas realizada sobre um vetor de chaves em cada nó. Se a chave não está no nó, segue-se para o filho apropriado.

Inserção: Ao inserir uma nova chave, se o nó estiver cheio, ele é dividido e a chave mediana sobe para o nó pai. Esse processo pode propagar até a raiz, aumentando a altura da árvore.

Remoção: A remoção é mais complexa, pois pode exigir fusão de nós ou redistribuição de chaves entre irmãos para manter o número mínimo de chaves por nó.

Exemplo de Aplicação

Em bancos de dados relacionais, índices B-Tree são usados para acelerar buscas, ordenações e junções. Por exemplo, ao criar um índice em uma coluna, o SGBD constrói uma B-Tree onde cada chave aponta para o registro correspondente.

Vantagens

- Excelente desempenho para grandes volumes de dados.
- Suporte eficiente a operações de faixa (range queries).
- Estrutura robusta para ambientes com muitas inserções e deleções.

Limitações

- Não é ideal para buscas por igualdade em campos não indexados.
- Não é eficiente para dados que mudam frequentemente de ordem (ex: índices de colunas com alta cardinalidade e atualização constante).

XtraDB

A imagem abaixo ilustra um cluster Percona XtraDB, onde múltiplos nós (servidores) participam de uma comunicação em grupo. Todos os nós podem receber operações de leitura e escrita, e as alterações

são propagadas entre eles para garantir alta disponibilidade e tolerância a falhas. Esse modelo é muito utilizado para bancos de dados críticos que exigem redundância e escalabilidade.

XtraDB é um mecanismo de armazenamento desenvolvido pela Percona como uma evolução do InnoDB, amplamente utilizado em bancos de dados MySQL e MariaDB. Ele foi criado para superar limitações do InnoDB, especialmente em ambientes de alta concorrência e grandes volumes de dados.

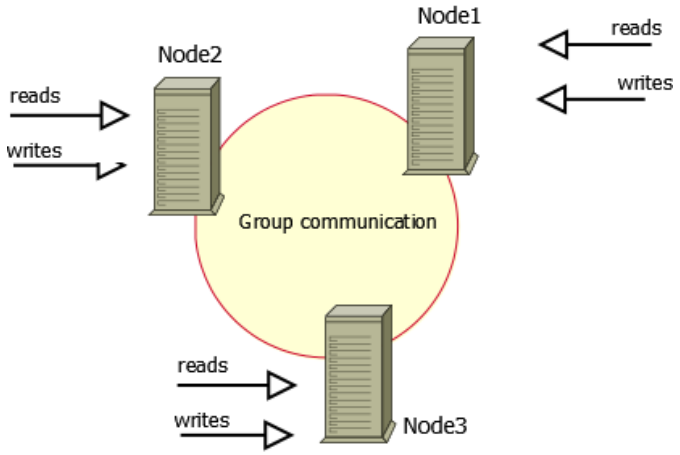


Figura 3: Arquitetura de um cluster Percona XtraDB Cluster. Cada nó pode realizar leituras e escritas, e a comunicação em grupo garante a consistência dos dados entre os nós.

Principais Melhorias em Relação ao InnoDB

- **Gerenciamento de buffer pool aprimorado:** Permite múltiplas instâncias de buffer pool, melhorando o uso de memória em servidores com muitos núcleos.
- **Controle de concorrência:** Algoritmos otimizados para gerenciamento de bloqueios (locks) e threads, reduzindo contenção e aumentando throughput.
- **Monitoramento avançado:** Exposição de variáveis e métricas detalhadas para análise de performance e tuning.
- **Aprimoramentos em flush e checkpoint:** Melhor controle sobre operações de escrita em disco, reduzindo picos de latência.
- **Recuperação e integridade:** Mecanismos robustos de recuperação após falhas, com suporte a crash recovery e integridade transacional (ACID).
- **Configurações dinâmicas:** Muitas opções de ajuste podem ser alteradas sem reiniciar o servidor.

Arquitetura e Funcionamento

O XtraDB utiliza uma arquitetura baseada em páginas de dados, armazenando registros em blocos de tamanho fixo (geralmente 16KB). Cada tabela é armazenada em um tablespace próprio ou compartilhado, e os dados são organizados em índices B-Tree.

Transações: Suporte completo a transações ACID, com controle de concorrência multiversão (MVCC), permitindo leituras consistentes sem bloqueios.

Índices: Suporte a índices primários (clustered) e secundários (non-clustered), ambos implementados como B-Trees.

Compressão: Suporte a compressão de dados e índices para economizar espaço em disco.

Exemplo de uso: Em ambientes OLTP (Online Transaction Processing), XtraDB é preferido por sua escalabilidade e capacidade de lidar com milhares de transações simultâneas.

Relação entre B-Tree e XtraDB

O XtraDB utiliza B-Trees como base para a implementação de seus índices, tanto primários quanto secundários. A estrutura B-Tree é fundamental para garantir operações rápidas e eficientes de busca, inserção e deleção.

Como os Índices B-Tree Funcionam no XtraDB

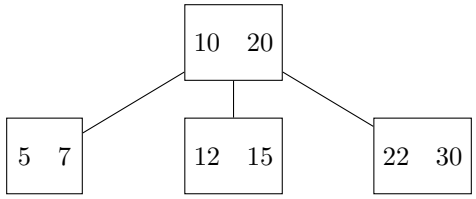
- **Índice primário (clustered):** Os dados da tabela são armazenados fisicamente na ordem do índice primário, que é uma B-Tree. Cada folha da árvore contém o registro completo.
- **Índices secundários:** Também implementados como B-Trees, mas as folhas contêm ponteiros (chaves primárias) para os registros reais.
- **Busca eficiente:** A navegação pela B-Tree permite localizar rapidamente registros, mesmo em tabelas com milhões de linhas.
- **Operações de faixa:** Consultas que buscam intervalos de valores (ex: BETWEEN, ORDER BY) são altamente otimizadas.
- **Manutenção automática:** O XtraDB gerencia automaticamente splits e merges de nós para manter a árvore balanceada.

A estrutura B-Tree é essencial para o desempenho do XtraDB, pois permite que o mecanismo de armazenamento realize operações de leitura e escrita de forma eficiente, mesmo sob alta concorrência e grandes volumes de dados. O XtraDB, por sua vez, aprimora o uso das B-Trees com algoritmos otimizados, gerenciamento avançado de memória e suporte a transações robustas, tornando-se uma escolha ideal para bancos de dados de missão crítica.

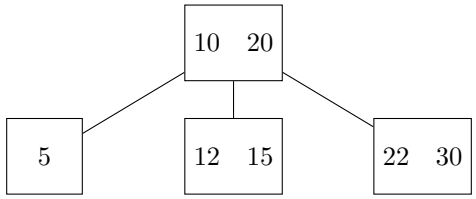
Exemplos

5. Remoção em B-Tree (antes e depois)

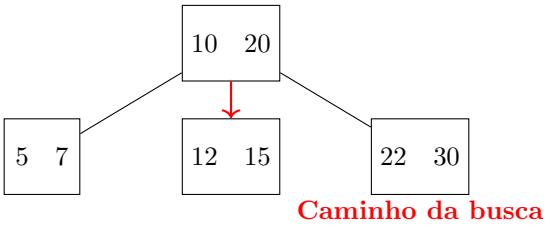
Antes da remoção de 7:



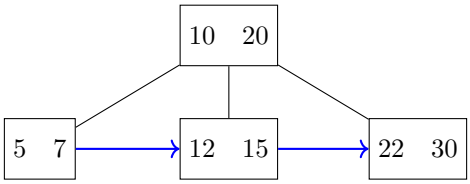
Após remoção de 7:



6. Busca em B-Tree (buscando 15)



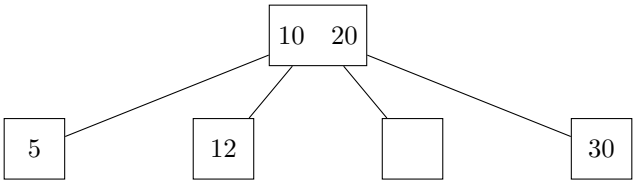
7. Range Scan em B+Tree



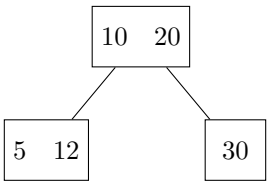
Range scan (varredura)

8. Fragmentação e Reorganização de Nós

Fragmentação:

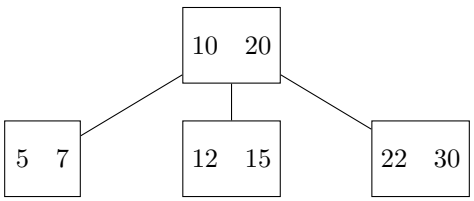


Após reorganização:



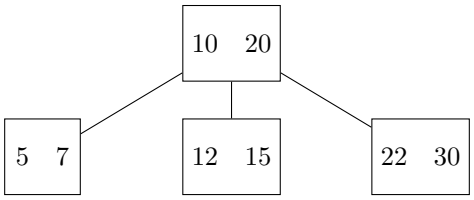
Exemplos

1. Estrutura de uma B-Tree (grau 2)

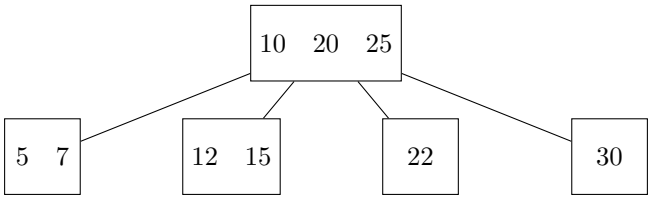


2. Inserção e Split em B-Tree

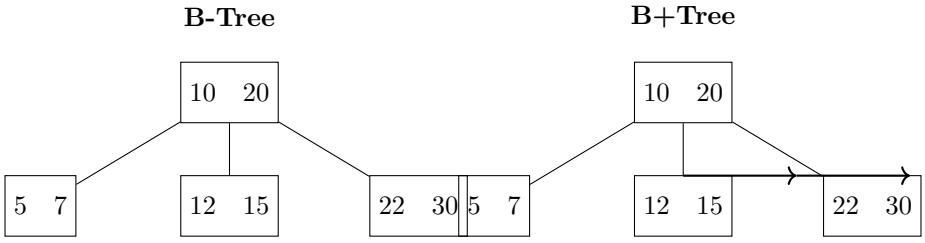
Antes da inserção:



Após inserir 25 (split do nó direito):

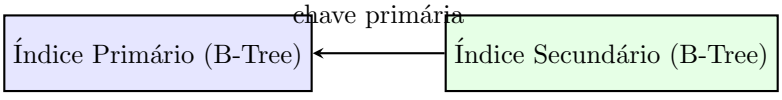


3. Diferença entre B-Tree e B+Tree



Na B+Tree, os dados completos estão apenas nas folhas, que são ligadas entre si.

4. Organização dos Índices no XtraDB



B-Tree vs B+Tree

A B+Tree é uma variação da B-Tree muito utilizada em bancos de dados modernos. Suas principais diferenças são:

- Em B+Tree, apenas as folhas armazenam os dados completos; os nós internos armazenam apenas chaves para navegação.
- As folhas são ligadas por ponteiros, facilitando operações de varredura sequencial (range scan).
- B+Tree é mais eficiente para buscas de faixa e ordenação.

Pseudo-código de Inserção em B-Tree

```
Inserir(chave, árvore):
    Se raiz está cheia:
        Crie novo nó s
        s torna-se a nova raiz
        s tem um filho, a antiga raiz
        Divida o filho s
        Insira chave em s
    Senão:
        Insira chave na raiz
```

Pseudo-código de Remoção em B-Tree

```
Remover(chave, árvore):
    Se chave está em folha:
        Remova chave
    Senão:
        Substitua chave pelo predecessor ou sucessor
        Remova recursivamente
    Se nó tem menos que t chaves:
        Realize fusão ou redistribuição
```

MVCC no XtraDB

O XtraDB implementa o controle de concorrência multiversão (MVCC), permitindo que múltiplas transações leiam e escrevam simultaneamente sem bloqueios. Cada transação enxerga uma "foto"consistente dos dados no momento em que começou, garantindo isolamento e alta performance.

Fragmentação e Reorganização de Índices

Com o tempo, inserções e deleções podem causar fragmentação nos índices B-Tree, reduzindo a eficiência. O XtraDB oferece comandos como `OPTIMIZE TABLE` para reorganizar e compactar os dados, restaurando a performance.

Tuning e Boas Práticas para XtraDB

- Ajuste do tamanho do buffer pool para aproveitar toda a RAM disponível.
- Uso de múltiplas instâncias de buffer pool em servidores com muitos núcleos.
- Monitoramento de métricas como `innodb_row_lock_waits` e `innodb_buffer_pool_reads`.
- Manutenção regular dos índices e tabelas.

Limitações e Alternativas

- Índices B-Tree não são ideais para buscas por similaridade ou texto completo.
- Para buscas textuais, melhor usar índices Fulltext.
- Para buscas espaciais ou por proximidade, o índices GiST ou R-Tree.
- Índices Hash podem ser mais rápidos para buscas por igualdade, mas não suportam range scan.