?????????????? ??????, ??? ?????? ??????? ?????.

> В реляционных СУБД большие таблицы могут содержать тысячи записей, поиск элемента без индекса стоит O(n). Из-за этого используют индексы, сбалансированное дерево является одним из самых популярных индексов

Сбалансированное дерево - структура данных, в которой высота левого и правого поддерева каждого узла отличается не более чем на единицу.

Основные свойства

1. Высота дерева минимальна
2. Все листьях находятся примерно на одном уровне
3. При вставке или удалении структура автоматически перестраивается

> Сложность в сбалансированном дереве всегда составляет O(log(n))

Сбалансированные деревья не используются напрямую, в базах данных(postgresql, mysql, oracle) используются **B-Tree или B+Tree**

Особенности

1. Каждый узел хранит несколько ключей
2. Каждый узел может иметь множество потомков
3. Дерево получается низким(3-4 уровня для миллиона записей)

B-Tree

Пример B-Tree

> Количество узлов выходящих из родителя равно **количество узлов родителя + 1**

>

> Поиск выполняется следующим образом - B-Tree начинает с верхнего узла и сравнивает значения(сначала меньше/больше, а потом равно ли оно), например возьмем **7**. B-Tree начинает слева направо и смотрит:
>
> - 7<3 - нет, значит идем дальше направо
> - 7>3 - да, значит идем дальше направо
> - 7>6 - да, значит идем дальше направо
> - 7>10 - нет, значит нужная нам ветка между 6 и 10, т.к. 6<7<10
> - также смотрим в узле между 6 и 10(789), слева направо
> - 7=7 - да, нашли

Пример использования

1. Пользователь делает поле, например **name**, индексом
2. B-Tree записывает в себя отсортированный список имён и распределяет их по дереву
3. При запросе вида `SELECT * FROM users WHERE name='hyilo'` происходит следующее
   1. B-Tree смотрит на верхний узел и сравнивает то, что мы хотим найти с записями
   2. До того момента пока B-Tree не найден элемент он будет проходить по узлам

      > ВАЖНО! B-Tree оперирует знаками <,>,= это значит что индексироваться могут и строки, и числа

B+Tree ???????? ????? ?????, ??? ? B-Tree, ?? ? ???? ???????? ????????? ?????? ?? ???????, ?? ??????? ????? ???????? ??? - ?????? ??????????

```
       [30 | 50]
      /    |    \
[10 | 20] [35 | 40] [55 | 60]
```

> Поиск выполняется следующим образом - B+Tree начинает с верхнего узла и сравнивает значения меньше/больше, например возьмем **27**. B+Tree начинает слева направо и смотрит:
>
> - 27<30 - да, значит идем ниже
> - 27<10 - нет, значит проверяем дальше этот узел
> - 10<27<20 - нет, значит точно верно что 20<27
> - дальше идут такие же проверки, пока не найдем значение

Сложности

Операция	Без индекса	С B-Tree
Поиск	O(n)	O(log n)
Вставка	O(n)	O(log n)
Удаление	O(n)	O(log n)

??????? ?????? ????????

> У дерева есть строгие правила по узлам, для разных структур данных оно может иметь разный порядок(m), в большинстве случаях берется m=4

B-Tree порядка m означает, что в одном узле:

- минимум ⌈m/2⌉ − 1 ключей
- максимум m − 1 ключей

Количество детей:

- от ⌈m/2⌉ до m

Пример

m	Кол-во узлов	Кол-во детей
4	1-3	2-4
6	2-5	3-6
8	3-7	4-6

B-Tree

1. ????? ?????

Новое значение всегда вставляется **в листовой узел**.

1. Начинает поиск с корня
2. Сравнивает вставляемый ключ с ключами в узле
3. По диапазону выбирает нужную ветку
4. Повторяет, пока не дойдёт до листа

2. ??????? ? ????

- Ключ вставляется в отсортированном порядке
- Структура дерева не меняется

Пример: Лист `[10, 20, 40]`, вставляем `30` → `[10, 20, 30, 40]`

3. ???????????? ????

Если после вставки в узле стало слишком много ключей

1. Узел делится на два
2. Средний ключ поднимается в родительский узел
3. Левые ключи остаются в левом узле, правые — в правом

Пример:

```
[10 | 20 | 30 | 40 | 50]   ← переполнение
```

Средний ключ `30` поднимается вверх:

```
          [30]
         /    \
   [10 | 20]  [40 | 50]
```

Если родительский узел тоже переполняется после добавления среднего ключа, то операция повторяется **рекурсивно вверх**.

Если переполняется корень:

- Создаётся новый корень
- Высота дерева увеличивается на 1

B+Tree

Работает практически также

- Средний ключ **копируется в родителя**
- Все реальные данные остаются в листьях

Revision #1
Created 2026-01-14 19:28:58 UTC by Fedmog1lnkv
Updated 2026-01-15 15:19:27 UTC by Fedmog1lnkv