



# Работа с СУБД **SQL**

## Индексы.

• REC Проверить, идет ли запись

**Меня хорошо видно  
&& слышно?**



# Проверка

- настройка микрофона и аудио
- проверка работы чата





# Индексы. Структуры данных.

# Индекс

- служебная структура данных
- некий способ отображения ключа в данные
- ускоряют поиск, сортировку
- требуют доп. место для хранения
- обновляются при вставке (модификации данных)

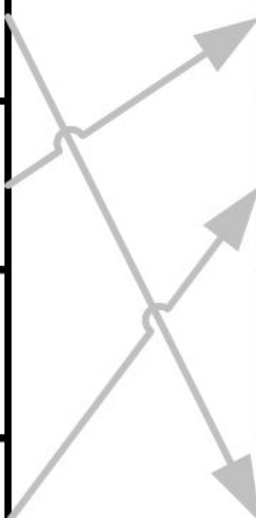
## Пример из жизни - оглавление

Индекс

Ключ 1	P1
Ключ 2	P2
...	...
Ключ N	PN

Данные

Запись 1
Запись 2
...
Запись N



# Какие индексы бывают

Уникальные	Не уникальные
Простые	Составные
Кластерные	Некластерные

# Вопрос

Много индексов - хорошо или не очень?



# Применимость

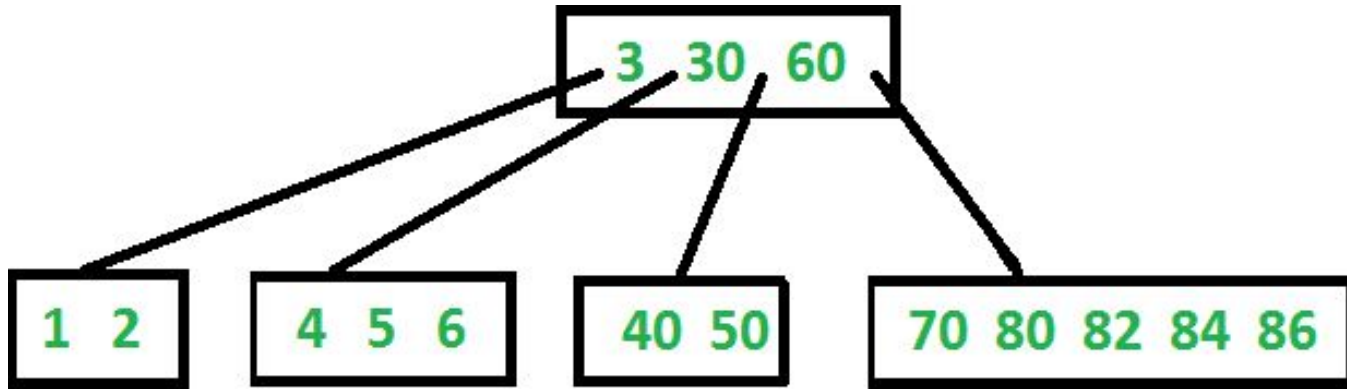
- Ускоряют запросы
- Позволяют делать constraints (UNIQUE, FOREIGN KEY)

# У нас много данных

- оперативная память очень ограничена
- будем использовать внешний носитель (привет, медленный HDD)
- $\log N$  обращений к медленной памяти

Можем ли мы придумать что-то лучше?!

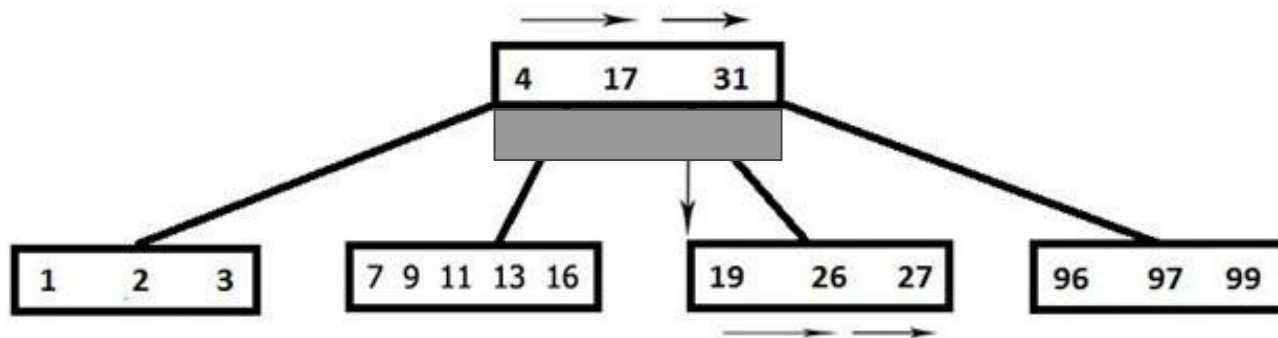
# B-tree



# Характеристики

- сильно ветвистое сбалансированное дерево
- $t$  - минимальная степень
- минимум  $t - 1$ , не более  $2t - 1$  ключей (кроме корневого) небольшая высота дерева

# Поиск в B-tree



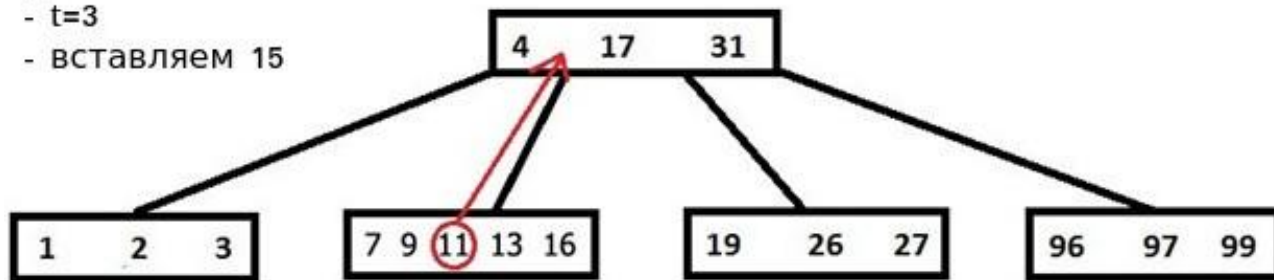
# Поиск в B-tree

- алгоритм аналогичен бинарному, но дальнейший выбор не из 2х, а из нескольких
- поиск за  $O(t \log t(n))$
- ЧО обращений к диску  $O(\log t(n))$

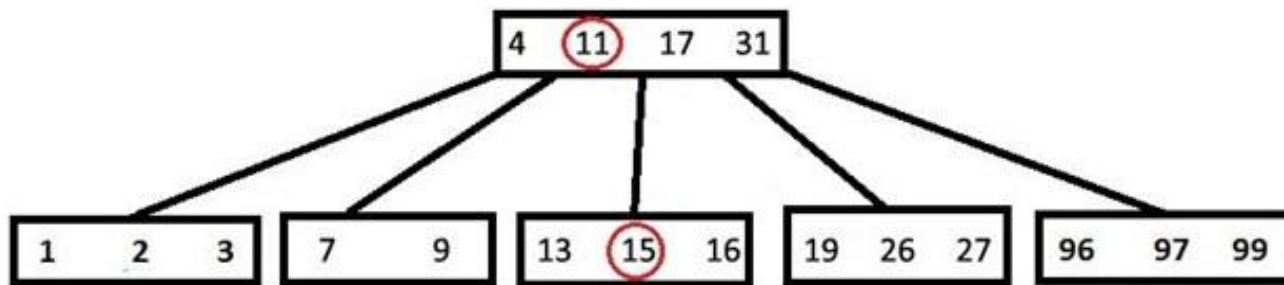
# Добавление в B-tree

Дано:

- $t=3$
- вставляем 15



## Добавление в B-tree





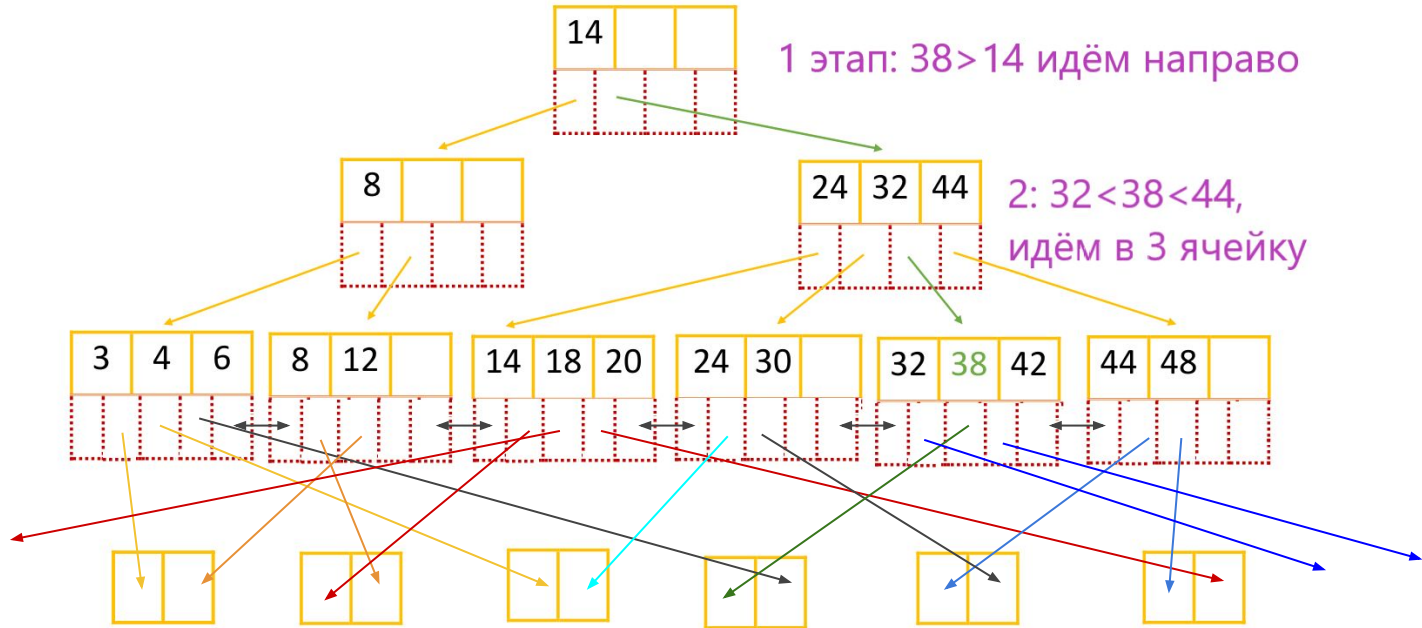
# Добавление в B-tree

- нельзя добавить ключ в уже заполненный узел
- разбиение на 2 по  $t-1$  элементу может
- привести к увеличению высоты
- добавление за  $O(t \log t(n))$
- НО обращений к диску  $O(\log t(n))$

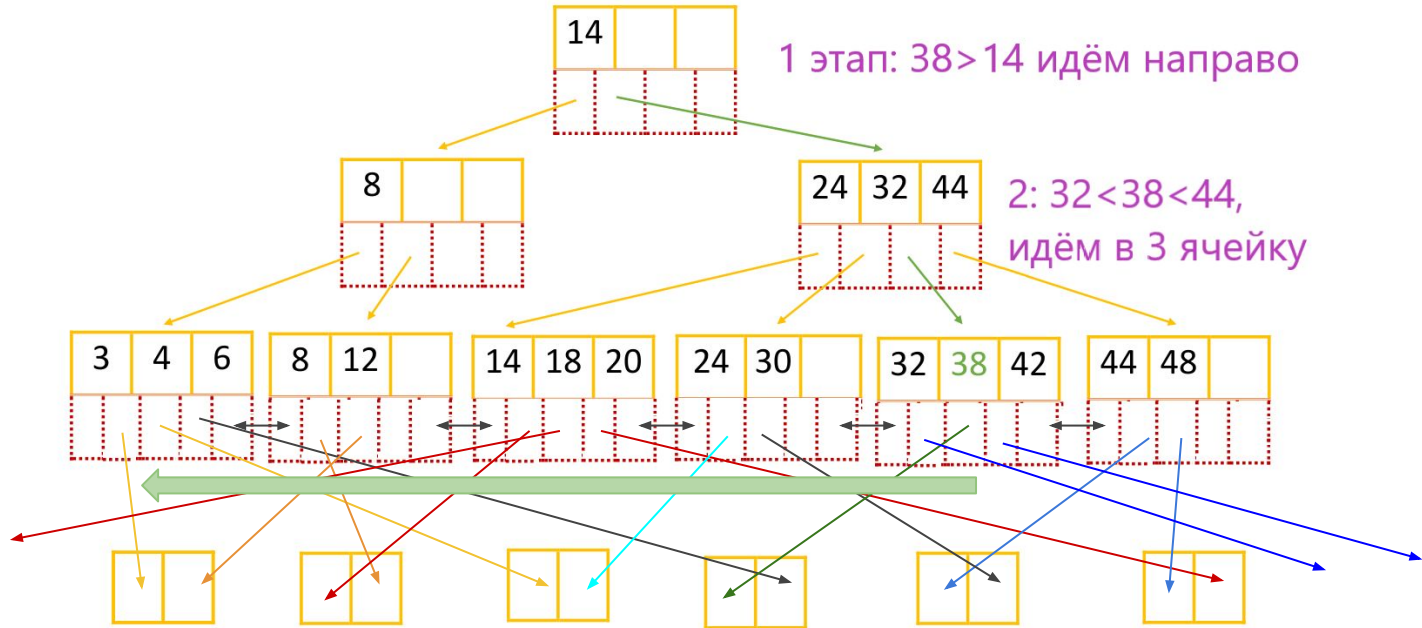
# Какой $t$ мне взять?

- больше  $t \Rightarrow$  меньше высота дерева
- зависит от размера блока на диске
- зависит от объема оперативной памяти
- обычно от 50 до 2000
- $t=1001$  и 1млрд записей  $\Rightarrow$  3 операции для любого ключа

# B+tree. Поиск по равенству = 38



# B+tree. Поиск по неравенству $< 38$





# Индексы. Практика.

# Индексы в Postgres

- Btree
- Hash
- GIST
- SP-GIST
- GIN
- BRIN
- Функциональные
- Частичные

# Составные индексы

- Индекс может быть многоколоночным
- Порядок столбцов в индексе ВАЖЕН
- В дерево пишется весь кортеж (5, 3, 4)
- Правило сравнения работает слева-направо: (1, 2, 3) < (1, 3, 1)
- Выбирайте индекс с большим количеством полей: (usr\_id) и (usr\_id, added)

# Кейс

- Сайт состоит из постов и комментариев к ним
- Таблица с постами
- Таблица с комментариями (comment\_id (PK), post\_id, time, author, ...)
- Пользователь жалуется, что сайт при попытке посмотреть комментарии к посту тормозит
- Что можно сделать с базой?

```
select comment_id, post_id from comments where post_id = ?
```



# Решение

```
select * from comments where post_id = 1
```

Сделаем в качестве PRIMARY KEY (post\_id, comment\_id)  
comment\_id UNIQUE NOT NULL

или

IDX (post\_id)

# Как Postgres использует индексы?

- Поиск данных
- Сортировка
- Избежание чтения из таблицы (покрывающие/covering индексы)
- Специальные оптимизации

# Проектирование

- uuid text - 32 байта - uuid uuid 16 байт
- Меньше индексов - лучше
- Меньше джойнов - лучше
- Составные типы данных вместо отдельных таблиц

# Покрывающий индекс

Если решили идти по индексу, а он содержит уже все необходимые данные, то в таблицу не лезем.

Можно включить неиндексируемые поля с помощью **INCLUDE**

# Кластерный индекс

В результате кластеризации таблицы её содержимое физически переупорядочивается в зависимости от индекса. Кластеризация является одноразовой операцией: последующие изменения в таблице нарушают порядок кластеризации

<https://www.postgresql.org/docs/current/sql-cluster.html>

# ГОТОВИМ данные

```
CREATE TABLE tbl (  
  id SERIAL NOT NULL PRIMARY KEY,  
  a bigint NOT NULL,  
  b bigint NOT NULL,  
  c bigint NOT NULL,  
  d bigint NOT NULL,  
  e bigint NOT NULL,  
  str text NOT NULL  
);  
INSERT INTO tbl (a,b,c,d,e,str)  
SELECT random()*1000, random()*1000, random()*1000, random()*1000,random()*1000,  
md5(random())::text  
FROM generate_series(1, 200000) s(i);
```

# Запросы на поиск

```
create index a on tbl(a);  
select a, b from tbl where a = 1
```

```
create index a_b on tbl(a,b);  
select a, b from tbl where a=1 and b=3;
```

- Поиск работает по префиксам индексов.
- Используем explain!

# Запросы на поиск

```
create index a_b_c on tbl(a, b, c)
```

Хорошие запросы: where

- **a=0**
- **a>0**
- **a=0 and b>4**
- **a=0 and b=2**
- **a=0 and b=2 and c>4**
- **a = 0 and b in (2, 4) and c > 3**



# Запросы на поиск

Плохие запросы: where

- $b = 6$
- $b > 3$

Частичное использование индексов:

- $a > 0$  and  $b = 4$
- $a = 0$  and  $b > 3$  and  $c = 2$
- $a = 0$  and  $b > 2$  and  $c > 3$

# Запросы на поиск

**Срабатывает ли покрывающий индекс (индекс на a, b, c)?**

```
select a, b from tbl where a=0 and b=3
```

```
select a,b,c from tbl where a=0 and b=3
```

```
select a,b,c,d from tbl where a=0 and b=3
```

```
select a,b,c,id from tbl where a=0 and b=3
```

# EXPLAIN

## QUERY PLAN

```
-----  
Index Only Scan using a_b_c on tbl (cost=0.42..7.90 rows=199 width=16) (actual time=1.453..1.574 rows=202 loops=1)  
  Index Cond: (a = 60)  
  Heap Fetches: 0  
Planning Time: 0.996 ms  
Execution Time: 1.686 ms  
(5 rows)
```

# Виды проходов по индексу

- Seq Scan - последовательный просмотр таблицы
- Index Scan - просмотр по индексу и в таблице
- Index Only Scan - просмотр по индексу без обращения к таблице
- Bitmap Heap Scan - оптимизация с построения битовых карт для поиска

# Сортировка

## Хорошие запросы:

`select * from tbl`

- `order by a limit 10;`
- `where a=1 order by b limit 10;`
- `where a>0 order by a limit 10;`
- `order by a desc, b desc limit 10;`
- `order by a asc, b asc limit 10;`

# Когда индексы не работают

- Часть выражения:  $a + 1 = 3$ ,  $a = \text{func}(2)$
- Преобразование типов:  $\text{str} = 1$
- Несоответствие кодировок
- Индекс не существует или невалидный
- Используется дубликат этого индекса

# Как выбирается индекс

- План выбирается динамически с учетом «костов»
- Использование таблицы статистики

# Доп.материал

-- Простой индекс

<https://www.db-fiddle.com/f/chBbJBYj6GMdyqCxXLBPEG/1>

-- Уникальный индекс

<https://www.db-fiddle.com/f/4FcEUB9zDZZWnkcJLvEqUq/2>

-- Составной индекс

<https://www.db-fiddle.com/f/v4KvbgaVVD4NDDjCBq4x5f/1>

-- особенности использования 2 и далее индексов в составном

<https://www.db-fiddle.com/f/v4KvbgaVVD4NDDjCBq4x5f/3>



# Доп.материал

-- Покрывающий индекс

<https://www.db-fiddle.com/f/33g3xrNmhpCt3a65rEUJJf/1>

-- Функциональный индекс

<https://www.db-fiddle.com/f/TZpgd6F4zofvbGCn1TOaR/0>

-- Частичный индекс

<https://www.db-fiddle.com/f/8bJbb6bZUTD8mOpq5v7Wvd/0>

# ДЗ №4

<https://github.com/DWH-course-examples/SQL-postgres/blob/main/homework/task4.md>

**СПАСИБО ЗА ВНИМАНИЕ!**

#аис  
#учисьваис