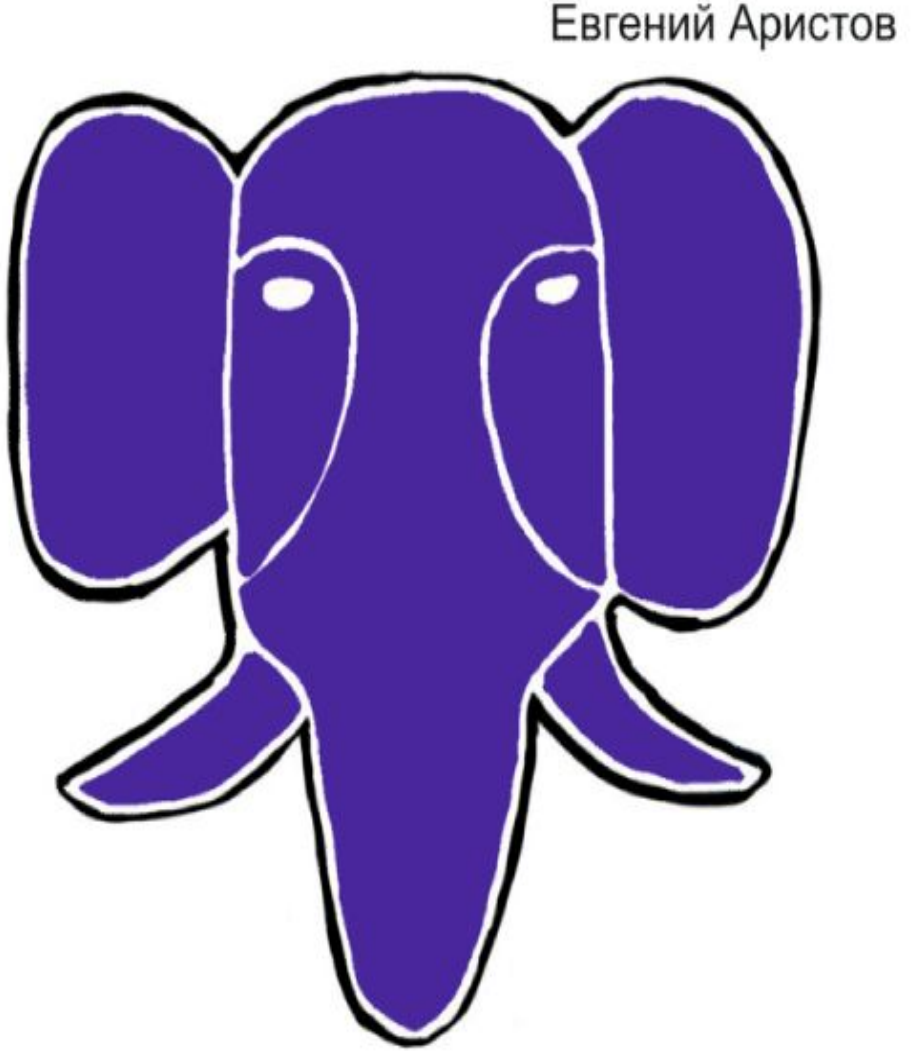
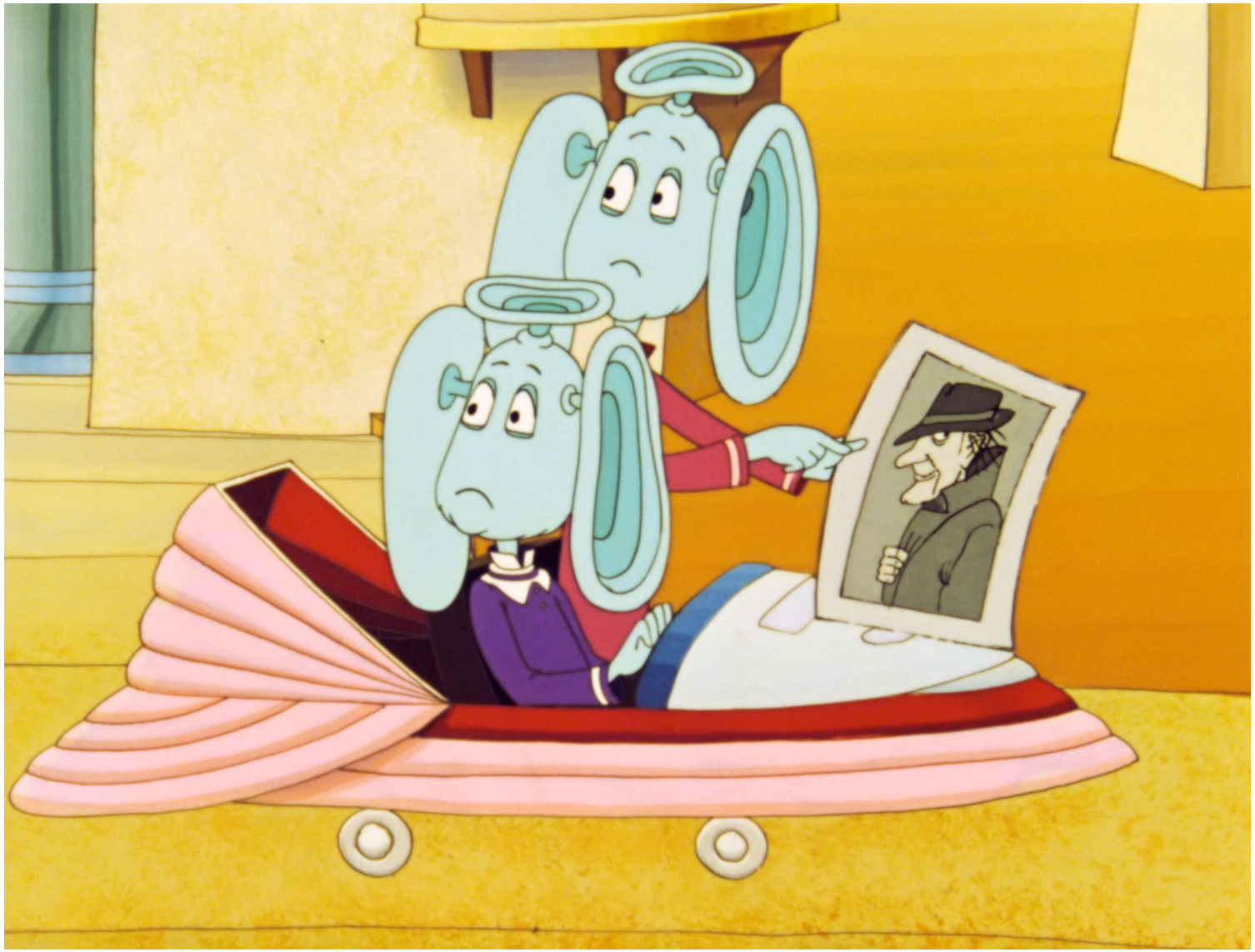
INDEXES





Правила вебинара

Задаем вопрос в чат

Вопросы вижу, отвечу в момент логической паузы

Если есть вопрос голосом - поставьте знак ? в чат

Если остались вопросы, можно их задать на следующем занятии

Маршрут вебинара

1. Виды индексов
2. Принципы работы индексов
3. Explain

Что такое индекс

https://aristov.tech

Что такое индекс

Оглавление

Индексы

Плюсы индексов:

- Позволяют обеспечить уникальность
- Ускоряют выборку в операциях SELECT
- При выборке данных только индексного поля, данные из таблицы не выбираются
- Увеличение скорости сортировки по индексному полю

Минусы индексов:

- Индексы требуют дополнительного места
- Необходимо перестраивать индексы при операциях UPDATE, DELETE, INSERT
- При большом количестве индексов оптимизатору сложно выбрать какой использовать

Виды индексов

https://aristov.tech

Синтаксис создания индекса


```
CREATE [ UNIQUE ] INDEX [ CONCURRENTLY ] [ [ IF NOT EXISTS ] имя ] ON имя_таблицы [ USING метод ]
    ( { имя_столбца | ( выражение ) } [ COLLATE правило_сортировки ] [ класс_операторов ] [ ASC | DESC ] [
NULLS { FIRST | LAST } ] [, ...] )
    [ INCLUDE ( имя столбца [,...]) ] для Btree и Gist
    [ WITH ( параметр_хранения = значение [, ... ] ) ]
    [ TABLESPACE табл_пространство ]
    [ WHERE предикат ]
```

https://aristov.tech

Виды индексов

Видов индексов в Постгресе довольно много:

- Btree
- Hash
- GiST
- SP-GiST
- GIN
- BRIN

Виды индексов

Битри (btree) или сбалансированное дерево (**default**)
Используется в 99% индексов.

- применим для любого типа, который можно отсортировать в чётко определённом линейном порядке.
- работает с операторами сравнения >, <, =, >=, <=, BETWEEN и IN и условия пустоты IS NULL и IS NOT NULL

Хэш-индекс (hash)

- Работает только с условием равенства (=). В условиях IS NULL и IS NOT NULL также не используется.

GIN индекс (Generalized Inverted Index) или обобщённый инвертированный индекс

- Применяется к составным типам, работа с которыми осуществляется с помощью ключей: массивы, jsonb.
- Предназначается для случаев, когда индексируемые значения являются составными, а запросы ищут значения элементов в этих составных объектах.
- Самый распространённый вариант использования индексов GIN & GiST - полнотекстовый поиск по аналогии с Google/Yandex.

Виды индексов (практически не используются)

GiST индекс (Generalized Search Tree) - обобщённое поисковое дерево.

Базовый шаблон, на основе которого могут реализовываться произвольные схемы индексации.

Для построения используют один из нескольких алгоритмов, наиболее подходящих под тип индексируемого поля, поэтому набор операторов зависит от типа поля.

Применяется для специфических типов данных: геометрии, сетевые адреса, диапазоны.

SP-GiST индекс (Space-Partitioned GiST).

Это GiST с разбиением пространства.

Метод поддерживает деревья поиска с разбиением, что позволяет работать с различными несбалансированными структурами данных (деревья квадрантов, k-мерные и префиксные деревья).

Как и GiST, SP-GiST позволяет разрабатывать дополнительные типы данных с соответствующими методами доступа.

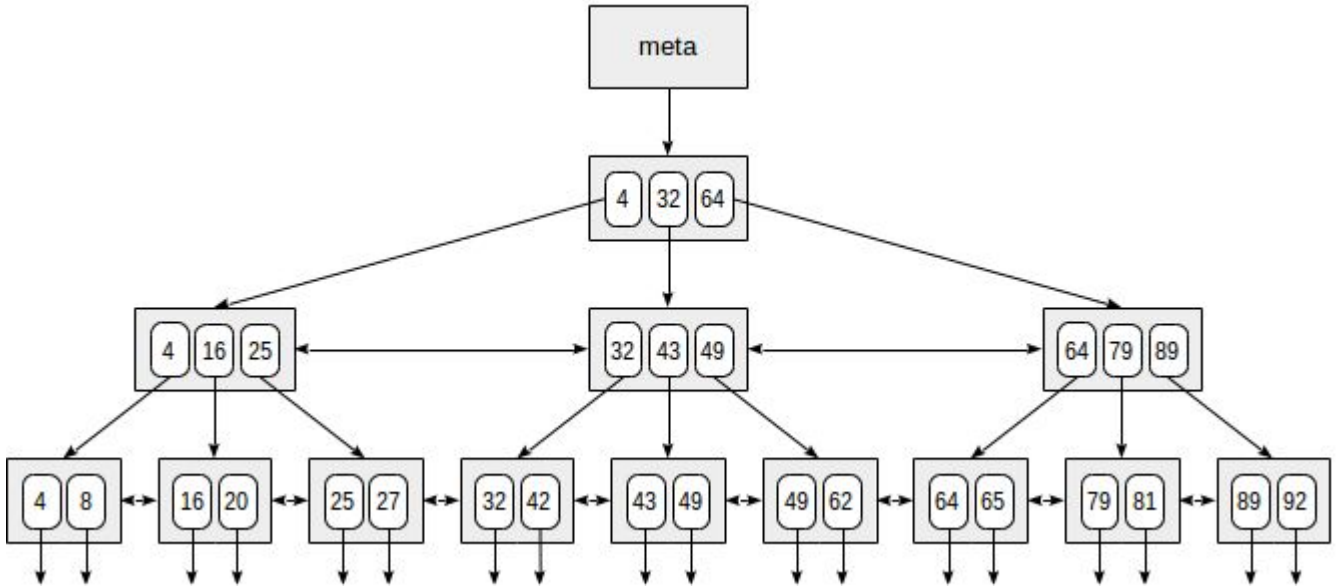
BRIN - block range index (с 11 версии) - блоковый индекс

Постранично раскладывает данные по времени. Может использовать например в таблицах логирования, GPS трекинга, IoT

https://www.crunchydata.com/blog/postgres-indexing-when-does-brin-win

Btree (на самом деле btree+)

<
<=
=
>=
>
BETWEEN
IN
NULL
NOT NULL



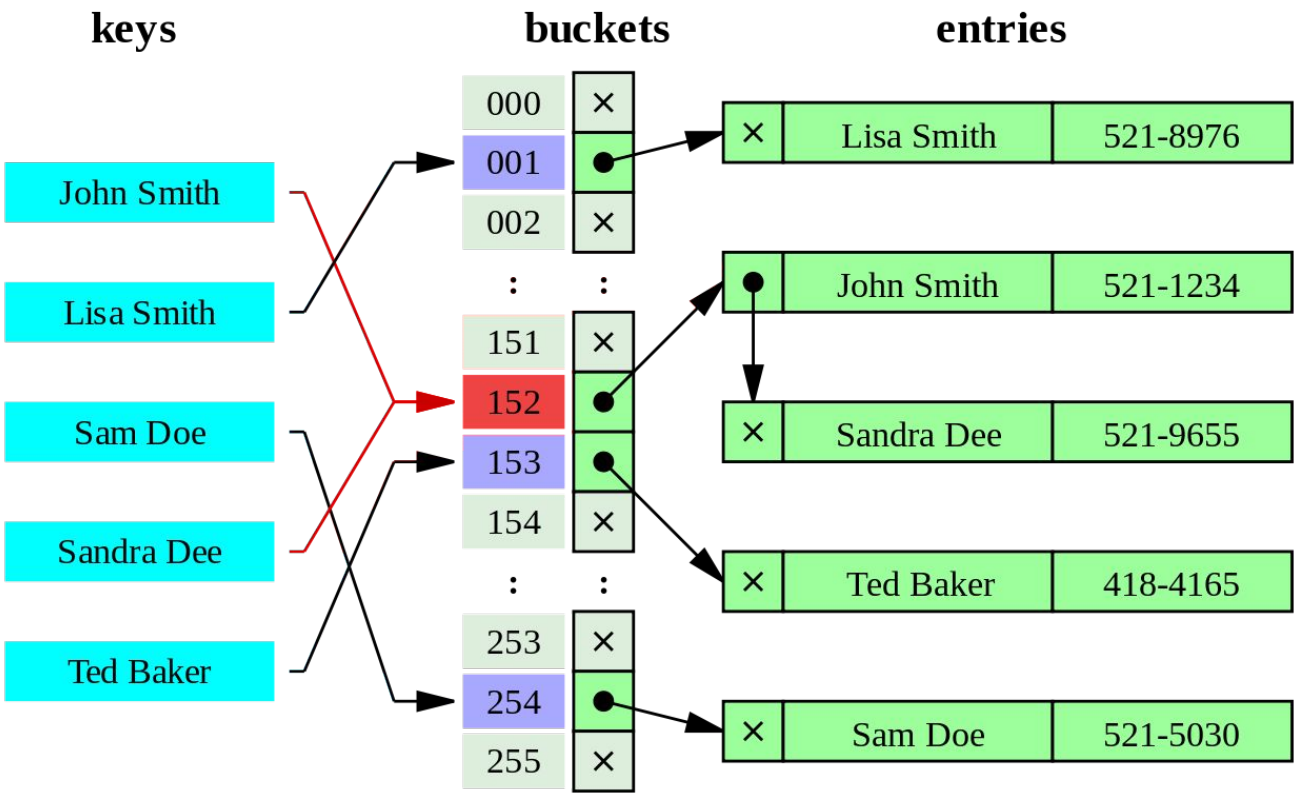
LIKE
ILIKE
~*
ORDER BY

Начиная с версии Postgresql 13 – Btree может весить меньше за счет дедупликации (default)
https://www.cybertec-postgresql.com/en/b-tree-index-deduplication/

https://aristov.tech

Hash

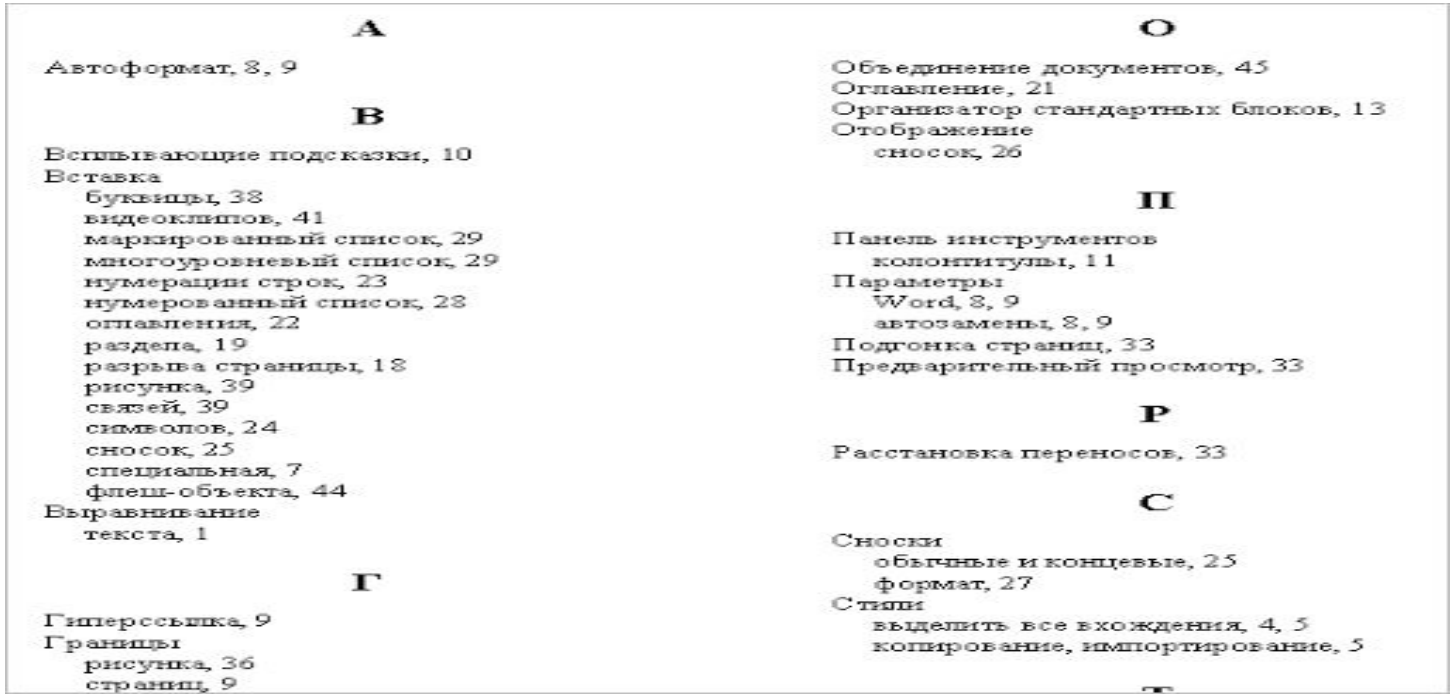
Только =



https://aristov.tech

Gin, Gist

- Используются для работы с текстом (более предпочтителен Gin) и геоданными (Gist)
- Тип столбца должен быть tsvector или tsquery (в случае GIST)
- Gin похож на алфавитный указатель



https://aristov.tech

Brin

В отличие от индексов, с которыми мы уже познакомились, идея BRIN не в том, чтобы быстро найти нужные строки, а в том, чтобы избежать просмотра заведомо ненужных. Это всегда ***неточный индекс***: он вообще не содержит TID-ов табличных строк.

Для больших объемов, где физическое расположение коррелирует со значением

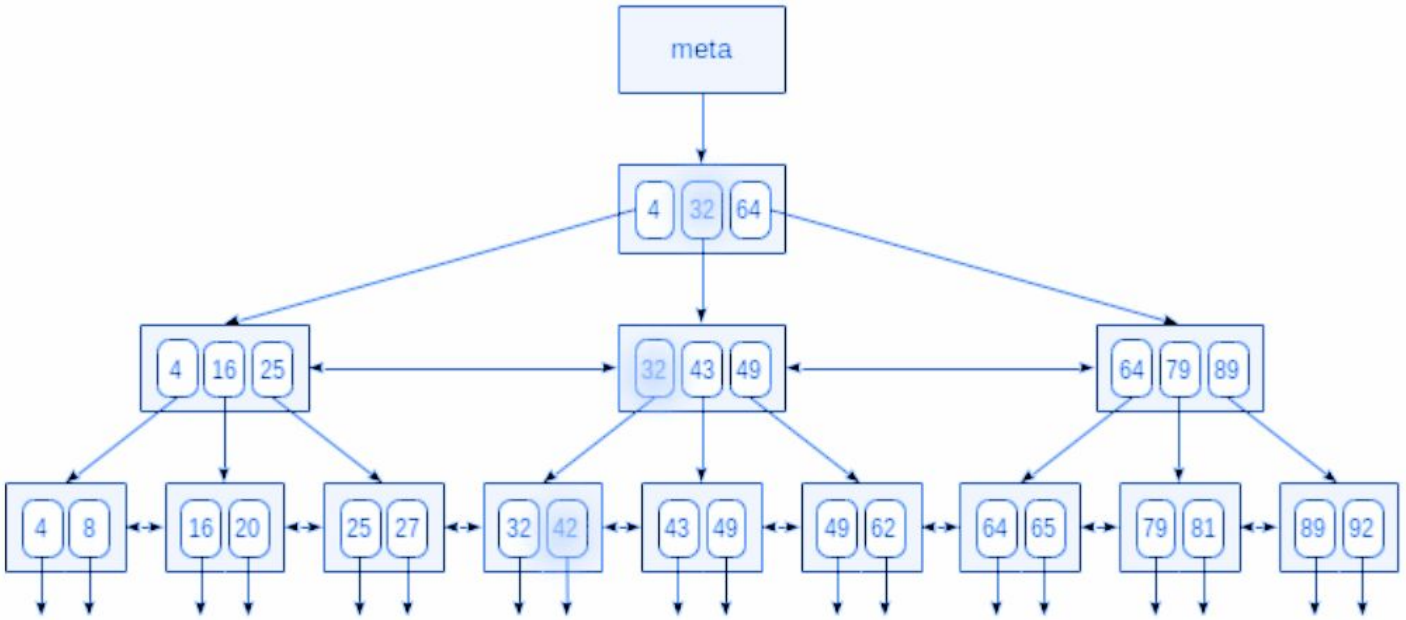
https://habr.com/ru/company/postgrespro/blog/346460/

https://aristov.tech

Btree

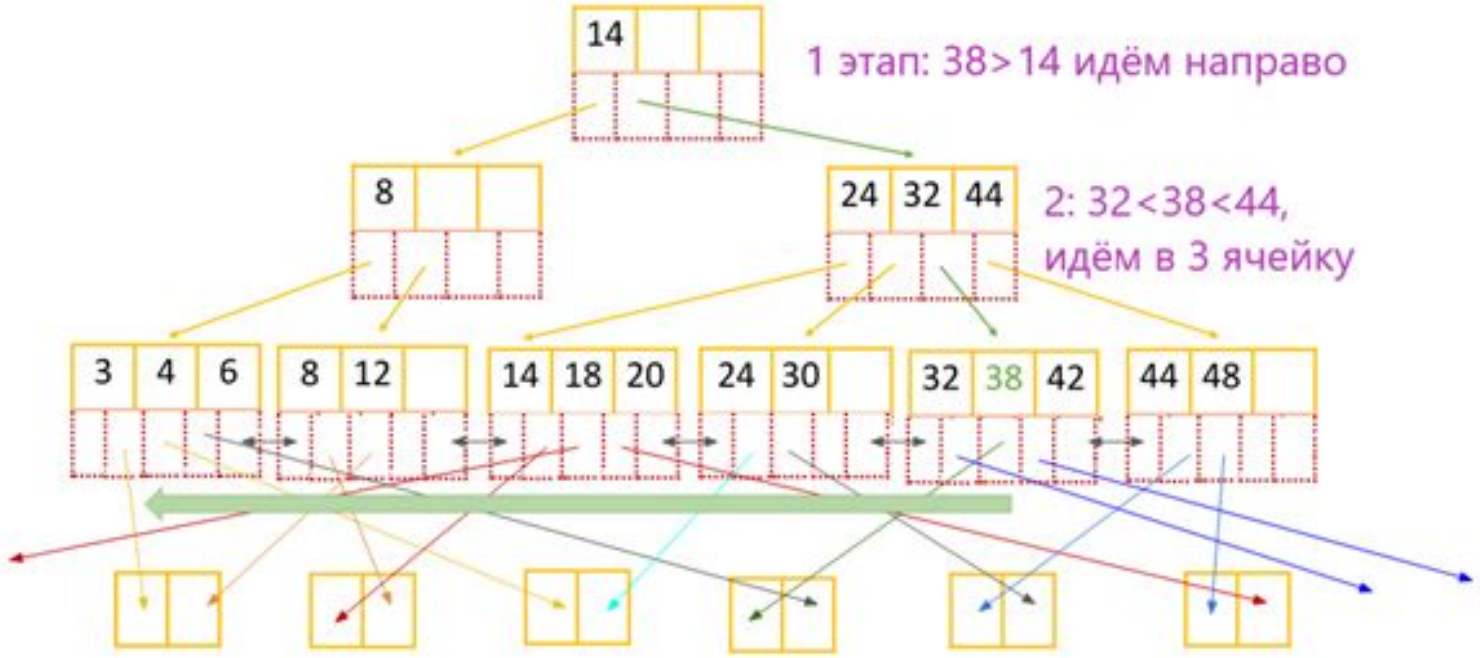
https://aristov.tech

Btree. Поиск по значению 42



https://aristov.tech

Btree. Поиск по неравенству <= 38



https://aristov.tech

Принцип работы Btree

Плюсы:

- довольно быстрый индекс - сложность O(log n) Вычислительная сложность — Википедия
- широкий диапазон условий для выборки по условиям

Минусы:

- если при очередном добавлении/изменении/удалении значения требуется балансировка - довольно медленно. Именно поэтому изначально плотность заполнения 90%
- необходимо их обслуживать
- работает хорошо только с простыми типами данных (числа, даты, денежные)
- работает **со строками, только** если ищем по шаблону (like) с начала строки (%АБВ)

Типы индексов

https://aristov.tech

Типы индексов

- Простой индекс
- Уникальный индекс
- Составной индекс
- Покрывающий индекс
- Функциональный индекс (можно использовать свою функцию, ее тип должен быть IMMUTABLE)
- Частичный индекс

Посмотрим на практике:

Типы индексов. Практика

Сгенерируем данные https://www.db-fiddle.com/f/mprRDa3iM1kWCnwDvjUPtX/0

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/v4KvbgaVVD4NDDjCBg4x5f/1

Особенности использования 2 и далее индексов в составном https://www.db-fiddle.com/f/v4KvbgaVVD4NDDjCBg4x5f/3

Покрывающий индекс https://www.db-fiddle.com/f/33g3xrNmhpCt3a65rEUJJf/1

Функциональный индекс https://www.db-fiddle.com/f/TZpqd6F4zofvbGCn1TQaR/0

Частичный индекс https://www.db-fiddle.com/f/8bJbb6bZUTD8mQpq5v7Wvd/1

https://aristov.tech

EXPLAIN

https://aristov.tech

Explain

EXPLAIN [(*параметр* [, ...])] *оператор*
EXPLAIN [ANALYZE] [VERBOSE] *оператор*

Здесь допускается *параметр*:

ANALYZE [*boolean*]
VERBOSE [*boolean*]
COSTS [*boolean*]
SETTINGS [*boolean*]
BUFFERS [*boolean*]
WAL [*boolean*]
TIMING [*boolean*]
SUMMARY [*boolean*]
FORMAT { TEXT | XML | JSON | YAML }

https://www.postgresql.org/docs/current/sql-explain.html

Explain строит план запроса

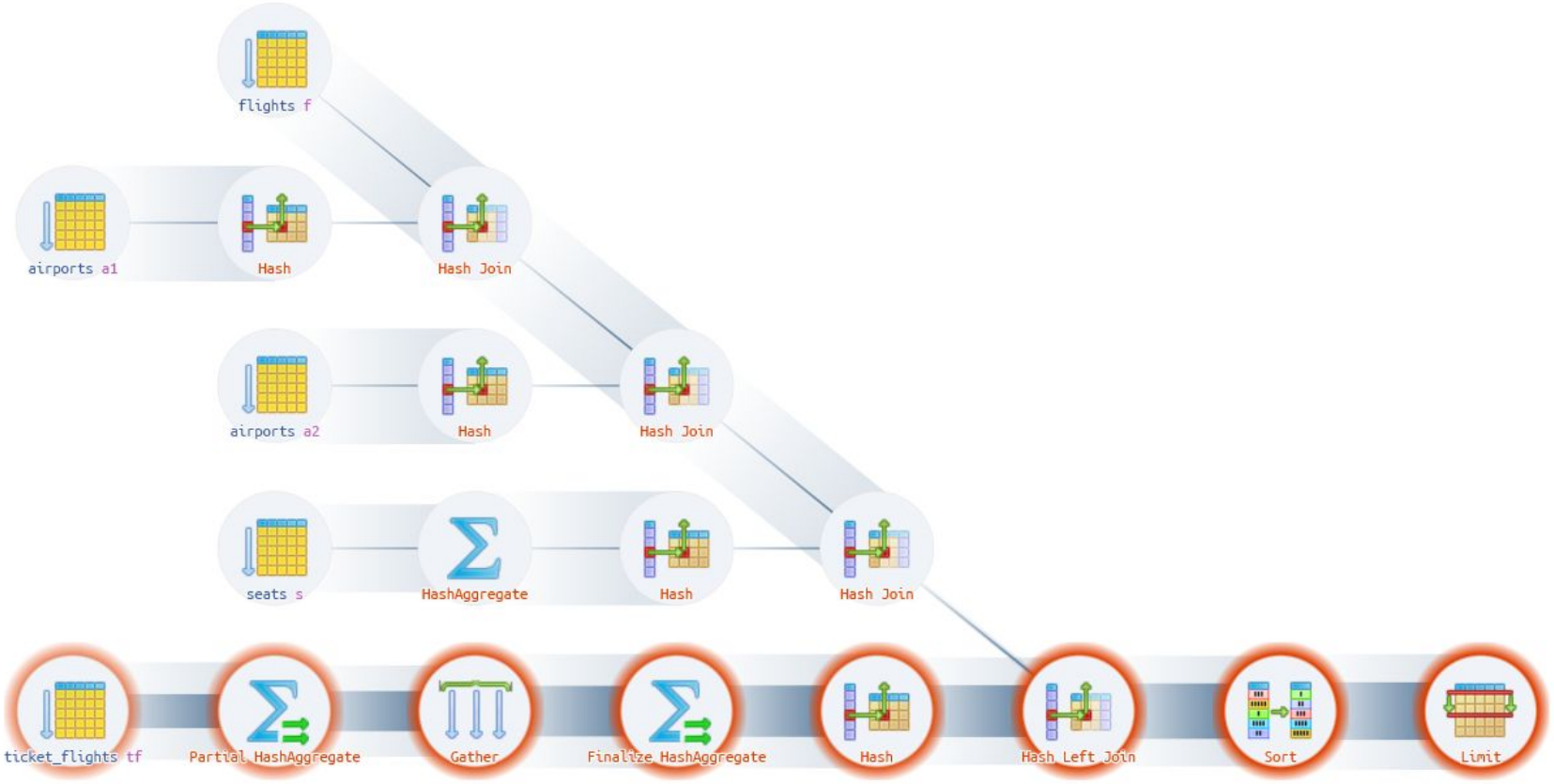
Стоимость запроса измеряется в “попугаях”

```
QUERY PLAN
Sort  (cost=169.51..172.01 rows=1000 width=87)
  Sort Key: f.title
  ->  Hash Join  (cost=41.93..119.68 rows=1000 width=87)
        Hash Cond: (f.film_id = fc.film_id)
        ->  Seq Scan on film f  (cost=0.00..64.00 rows=1000 width=19)
        ->  Hash  (cost=29.43..29.43 rows=1000 width=70)
              ->  Hash Join  (cost=1.36..29.43 rows=1000 width=70)
                    Hash Cond: (fc.category_id = c.category_id)
                    ->  Seq Scan on film_category fc  (cost=0.00..16.00 rows=1000 width=4)
                    ->  Hash  (cost=1.16..1.16 rows=16 width=72)
                          ->  Seq Scan on category c  (cost=0.00..1.16 rows=16 width=72)
```

Более подробно про шаги планирования запроса можно почитать
[Запросы в PostgreSQL: 1. Этапы выполнения]

Explain

https://explain.tensor.ru/



https://aristov.tech

Explain. Типы запросов и соединений

Объясняя необъяснимое. Часть 3 / Хабр
https://habr.com/ru/post/560834/
https://habr.com/ru/company/postgrespro/blog/696680/
https://habr.com/ru/company/postgrespro/blog/412605/
https://habr.com/ru/company/postgrespro/blog/683020/

https://www.postgresql.org/docs/current/using-explain.html

Варианты повлиять на планировщик:
https://www.postgresql.org/docs/current/runtime-config-query.html#RUNTIME-CONFIG-QUERY-CONSTANTS
https://github.com/ossc-db/pg_hint_plan

Отличная идея

https://aristov.tech

Отличная идея

Тупит запрос? Что первое делаем? Создаем индекс конечно!!!

Итого чем больше индексов, тем быстрее работают запросы.

Или это не так?

Отличная идея

Тупит запрос? Что первое делаем? Создаем индекс конечно!!!

Итого чем больше индексов, тем быстрее работают запросы.

- **Размер индексов может превысить размер самой таблицы и при этом значительно**
- **Опять же индексы желательно держать в памяти для скорости работы то**
- **Скорость вставки/обновления/удаления обратно пропорциональная количеству индексов и по итогу может на порядок упасть**

https://aristov.tech

Итоги

https://aristov.tech

Итоги

Более подробно индексы, хинты, оптимизация разбираются на курсе

Остались ли вопросы?

Увидимся на следующем занятии

https://aristov.tech

Спасибо за внимание!

Когда дальше и куда?
В чате напишу
материалы для бесплатного доступа будут появляться на ютубе

Аристов Евгений