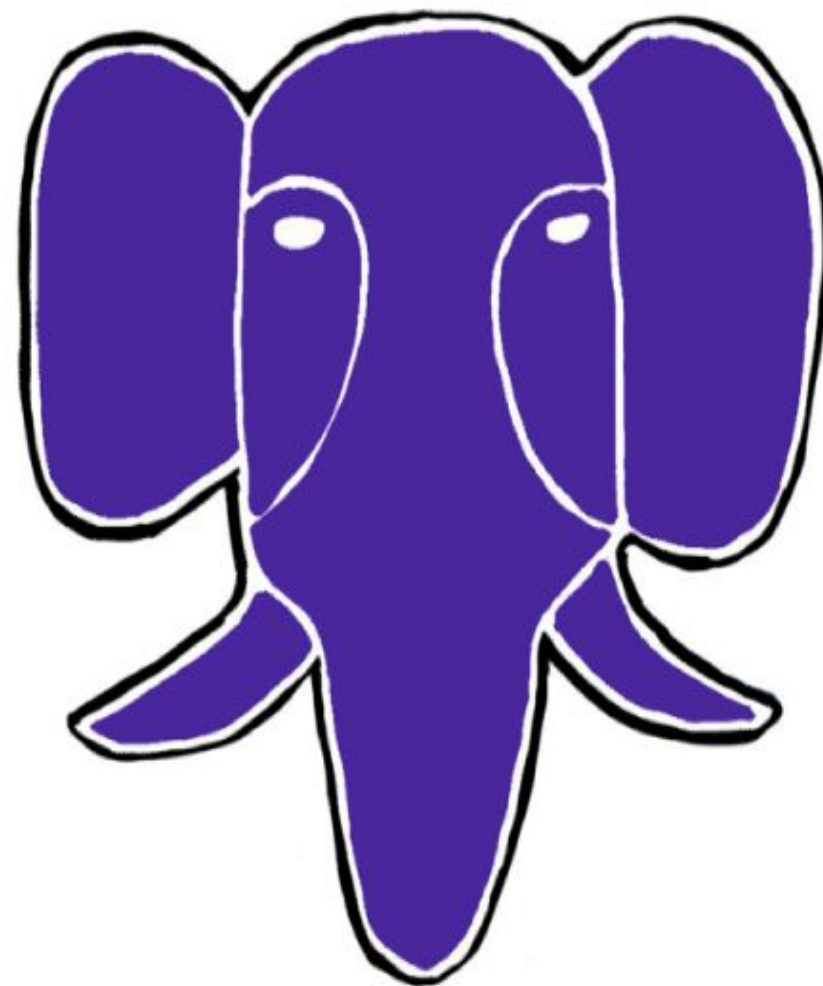
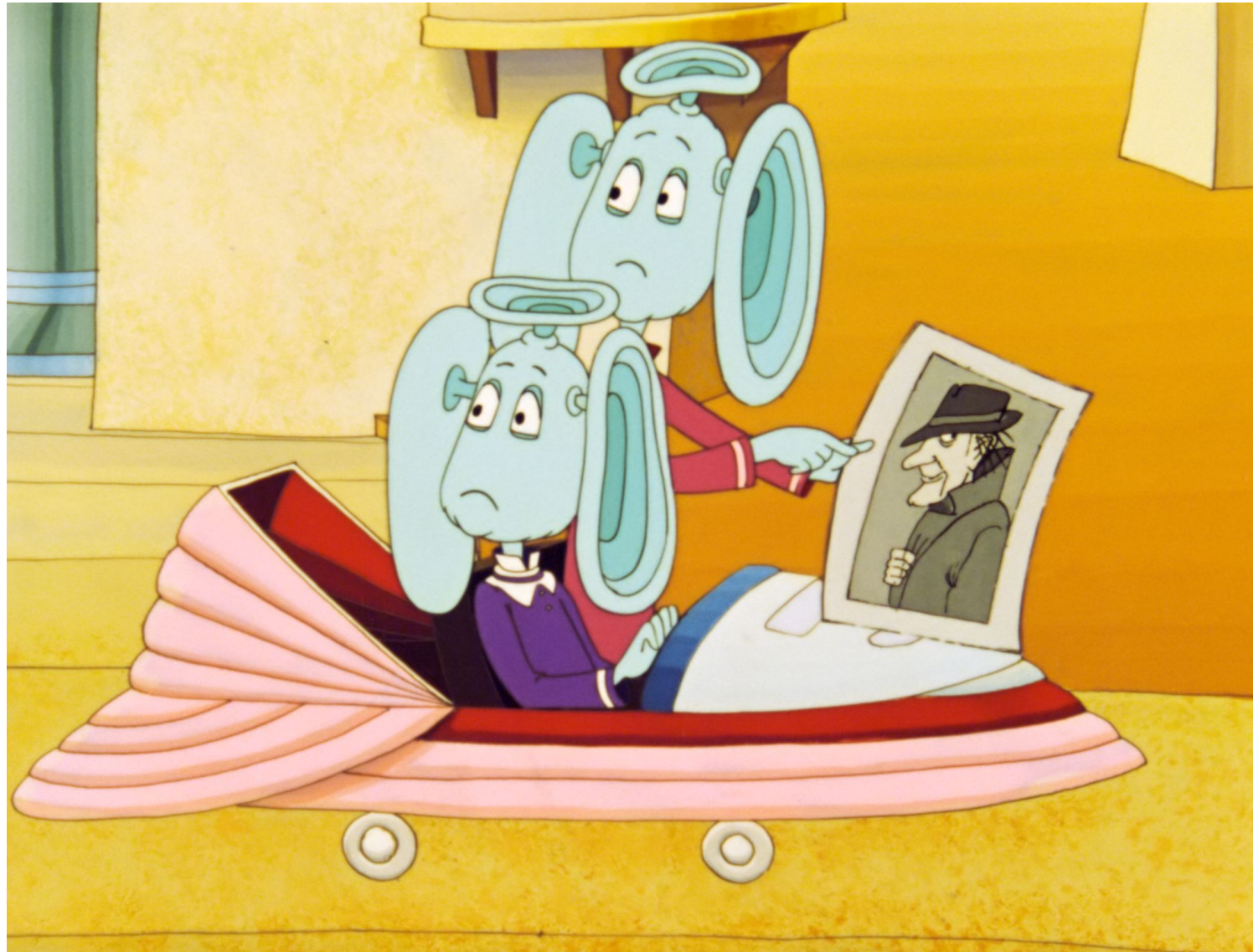


Евгений Аристов

Открытый вебинар Troubleshooting PostgreSQL Indexes

<https://aristov.tech>





Аристов
Евгений
Николаевич



<https://aristov.tech>

<https://aristov.tech>

Founder & CEO aristov.tech

25 лет занимаюсь разработкой БД и ПО

Архитектор высоконагруженных баз данных и инфраструктуры

Спроектировал и разработал более ста проектов для финансового сектора, сетевых магазинов, фитнес-центров, отелей.

Сейчас решаю актуальные для бизнеса задачи: аудит и оптимизация БД и инфраструктуры, миграция на PostgreSQL, обучение сотрудников.

Автор более 10 практических курсов по PostgreSQL, MySQL, Mongo и др..

Автор книг по PostgreSQL. Новинка [PostgreSQL 16: лучшие практики оптимизации](#)

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 написать мне через сайт aristov.tech

Маршрут вебинара

- ❖ Принципы работы индексов
- ❖ Особенности массовой вставки данных в PostgreSQL
- ❖ Когда индексы не работают
- ❖ Тонкости построения секционированных индексов
- ❖ Кластерный индекс
- ❖ Коротко о проекте aristov.tech
- ❖ Розыгрыш книг и скидки 30% на курс
- ❖ Ответ на вопросы

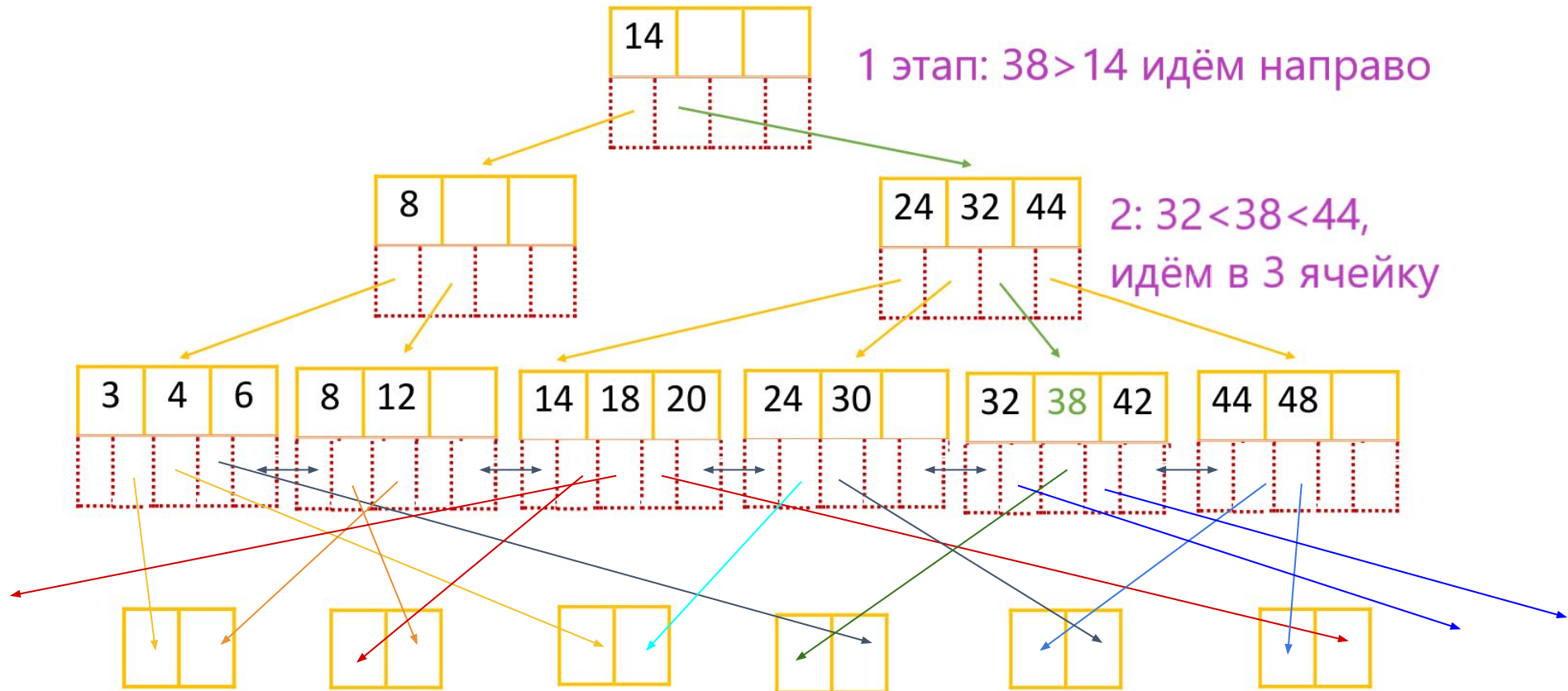
Индексы как серебряная пуля. или нет?

Проблематика

- ❖ создал индексы и они не используются(
- ❖ создал и вставка замедлилась
- ❖ массово залил данные и индексы не используются...
- ❖ место стало деваться непонятно куда
- ❖ скорость с индексами медленнее, чем без них
- ❖ а как узнать нужно ли создавать индекс?
- ❖ а что со старыми индексами - работают ли они вообще?
- ❖ да много вопросов на самом деле

B+tree или Btree+. Поиск по равенству = 38

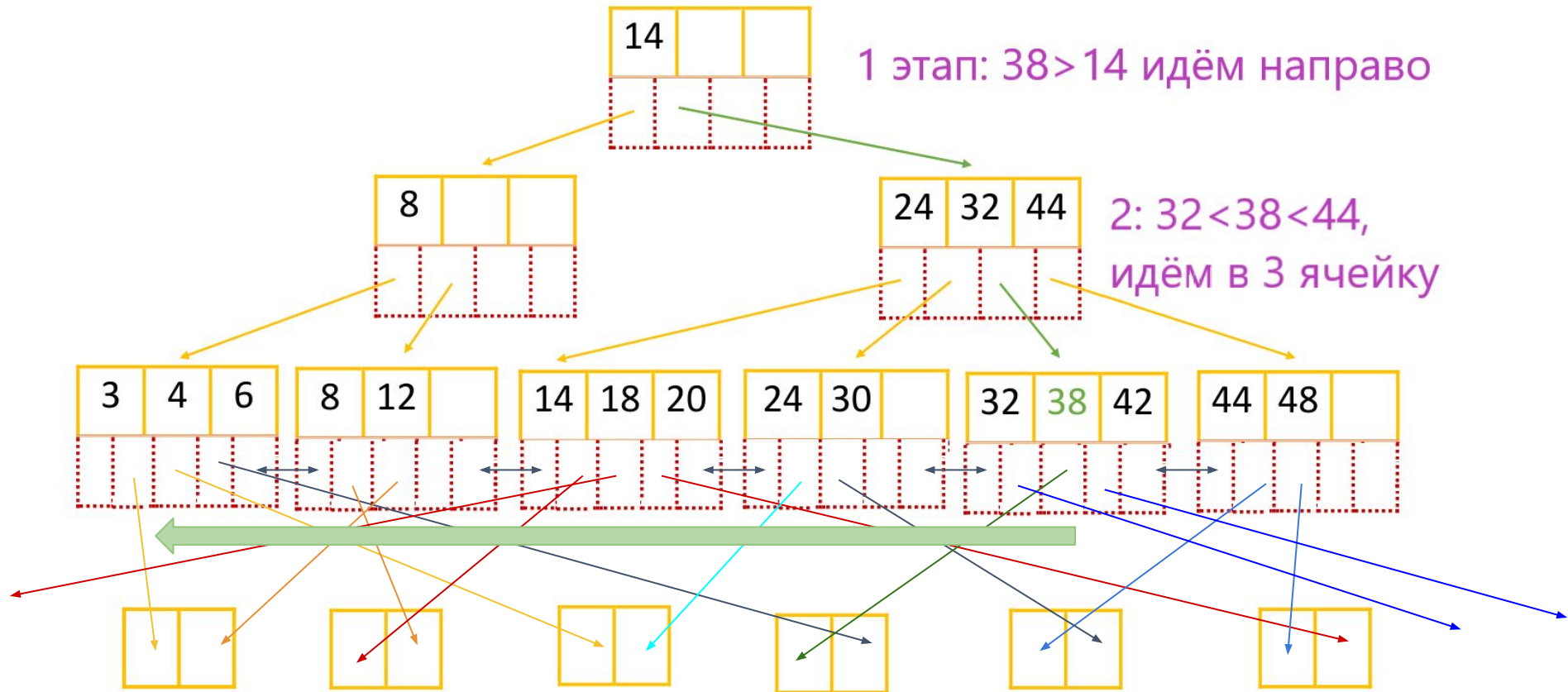
<https://aristov.tech>



<https://aristov.tech>

<https://habr.com/ru/companies/quadcode/articles/696498/>

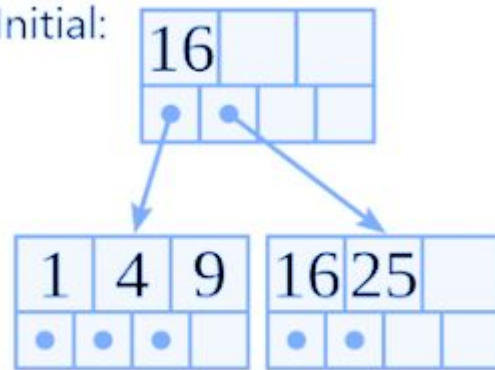
B+tree. Поиск по неравенству < 38



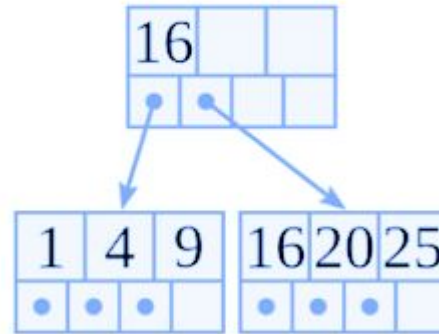
2: $32 < 38 < 44$,
идём в 3 ячейку

Пример работы по добавлению значений в B+ tree

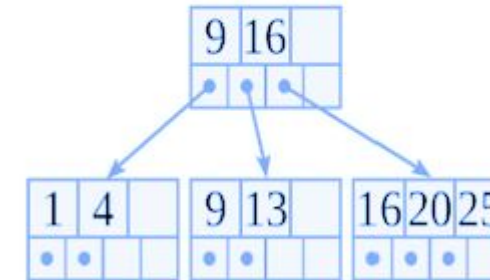
Initial:



Insert 20:



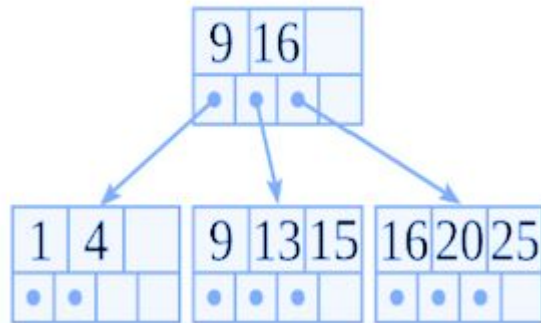
Insert 13:



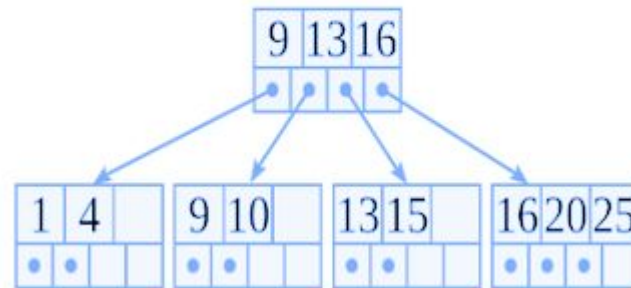
<https://medium.com/@lordmoma/why-using-b-tree-indexing-in-sql-6a3203ed57a5>

Пример работы по добавлению значений в B+ tree

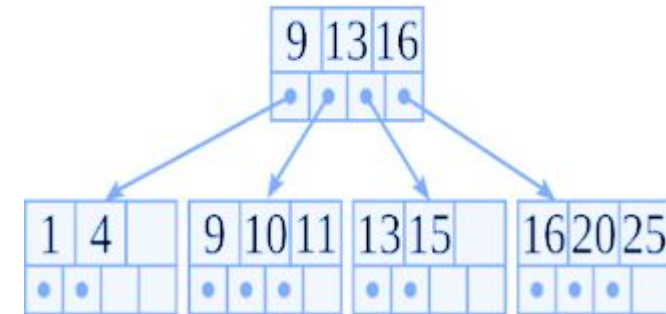
Insert 15:



Insert 10:

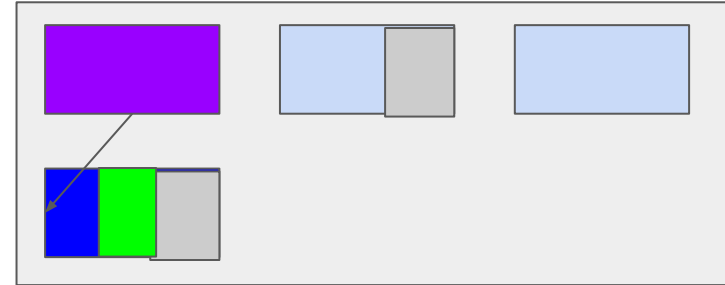
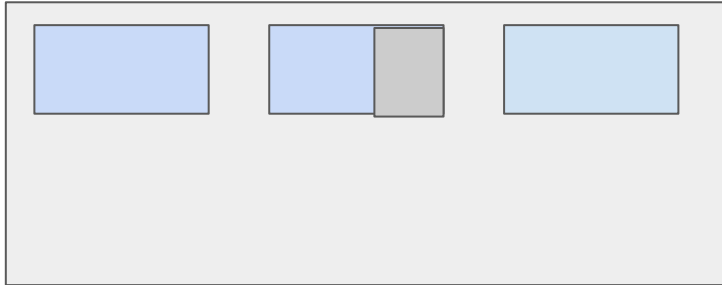


Insert 11:



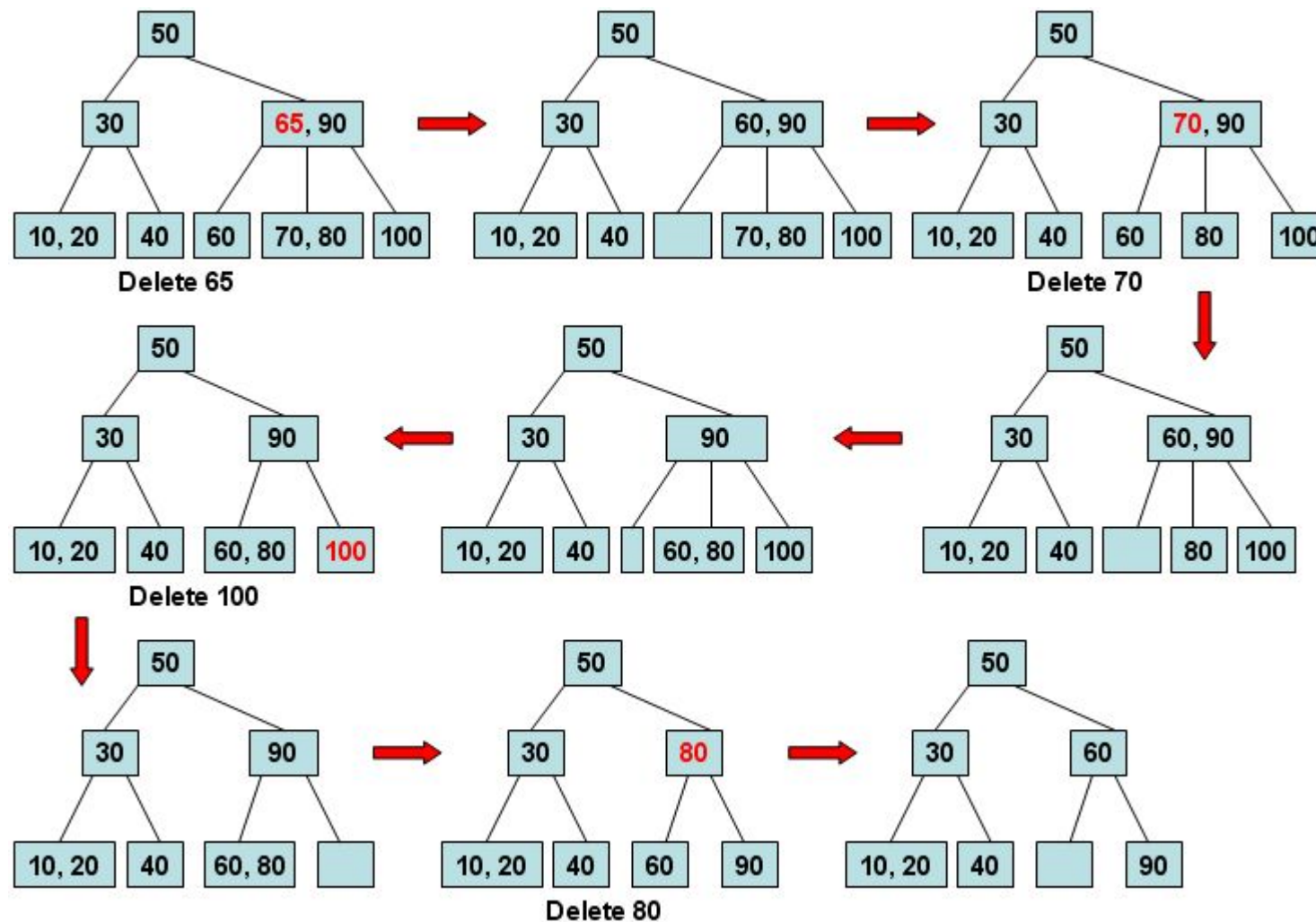
Физически выглядит так

при добавлении в 1 лист, так
как нет места - создается новый



Пример работы по удалению значений в B+ tree

<https://aristov.tech>



<https://aristov.tech>

<https://iq.opengenus.org/b-tree-deletion/>

Типы индексов

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

Хитрость уникального индекса

Объявляем тип NOT NULL, иначе можем добавлять сколько угодно NULL значений, ибо
NULL <> NULL

NULLABLE UNIQUE

<https://www.db-fiddle.com/f/honZUi6C2aQmNd3HZbUTqt/3>

NULLABLE FK -> NULLABLE PK

<https://www.db-fiddle.com/f/honZUi6C2aQmNd3HZbUTqt/4>

ссылка на составной уникальный ПК
добавляем несуществующее значение в FK

<https://www.db-fiddle.com/f/pkkczXgrU6EHUciWdHvY8q/1>

Unique без NOT NULL ошибка!

Хитрость уникального констрейнта

<https://aristov.tech>

Создание уникального индекса аналогично объявлению уникального констрейнта за одним отличием:

DEFERRABLE

NOT DEFERRABLE (default)

Это предложение определяет, может ли ограничение быть отложенным.

Неоткладываемое ограничение будет проверяться немедленно после каждой команды.

Проверка откладываемых ограничений **может быть** отложена до завершения транзакции (обычно с помощью команды SET CONSTRAINTS).

В настоящее время это предложение принимают только ограничения **UNIQUE**, PRIMARY KEY, EXCLUDE и REFERENCES (внешний ключ).

Откладываемые ограничения **не могут применяться** в качестве решающих при конфликте в операторе

INSERT с предложением ON CONFLICT DO UPDATE.

Хитрость уникального констрейнта

<https://aristov.tech>

Также есть еще одна особенность ограничений при выборе **DEFERRABLE** - поведение по умолчанию:

INITIALLY IMMEDIATE (default)
INITIALLY DEFERRED

Ограничение с характеристикой **INITIALLY IMMEDIATE** проверяется после каждого оператора, **позволяет включить** в транзакции режим проверки в конце.

Ограничение **INITIALLY DEFERRED**, напротив, **позволяет отключить** режим проверки в конце транзакции **в каждой транзакции**.

[Deferrable Constraints in PostgreSQL | Christian Emmer](#)

<https://aristov.tech>

Хитрость GIN индекса

У GIN-индекса есть параметр хранения `fastupdate`, который можно указать при создании индекса или изменить позже:

```
CREATE INDEX ON ts USING gin(doc_tsv) with (fastupdate = true);
```

При включенном параметре изменения будут накапливаться в виде отдельного неупорядоченного списка (в отдельных связанных страницах). Когда этот список становится достаточно большим, либо при выполнении процесса очистки, все накопленные изменения одномоментно вносятся в индекс. Что считать «достаточно большим» списком, определяется конфигурационным параметром `gin_pending_list_limit` или одноименным параметром хранения самого индекса.

Минусы: во-первых, замедляется поиск (за счет того, что кроме дерева приходится просматривать еще и неупорядоченный список), а во-вторых, очередное изменение может внезапно занять много времени, если неупорядоченный список переполнился.

Отличная идея

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**

Случаи, когда индексы в PostgreSQL могут не работать

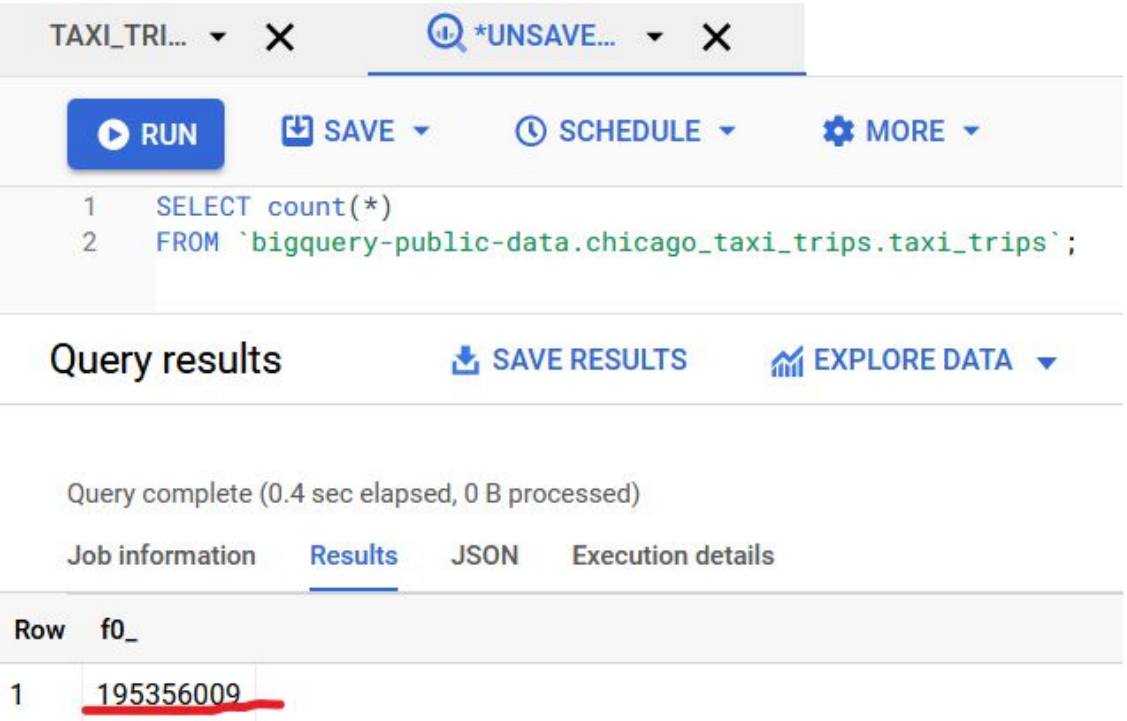
Индексы

Берем небольшую таблицу на ~200 млн. записей Чикагское такси в **BigQuery**

Посчитаем, сколько всего данных в наборе:

```
SELECT count(*)
```

```
FROM `bigquery-public-data.chicago_taxi_trips.taxi_trips`;
```



The screenshot shows the BigQuery interface. At the top, there's a tab labeled 'TAXI_TRI...' and a button to 'UNSAVE...'. Below this is a toolbar with 'RUN', 'SAVE', 'SCHEDULE', and 'MORE' buttons. The query editor contains the following SQL code:







```
1 SELECT count(*)
2 FROM `bigquery-public-data.chicago_taxi_trips.taxi_trips`;
```

Below the query editor, there's a section for 'Query results' with 'SAVE RESULTS' and 'EXPLORE DATA' buttons. The status bar indicates 'Query complete (0.4 sec elapsed, 0 B processed)'. There are tabs for 'Job information', 'Results', 'JSON', and 'Execution details'. The 'Results' tab is active, showing a table with one row and one column:

Row	f0_
1	195356009

Индексы

Попробуем посчитать агрегирующий отчёт - сумма чаевых по типам чаевых - итого нам нужно будет пробежать по всем записям и выбрать нужные данные, затратим при этом всего 0.9 секунды:

<div><div> RUN</div><div> SAVE ▾</div><div> SCHEDULE ▾</div><div> MORE ▾</div></div>				
<pre>1 SELECT payment_type, round(sum(tips)/sum(trip_total)*100, 0) + 0 as tips_percent, count(*) as c 2 FROM `bigquery-public-data.chicago_taxi_trips.taxi_trips` 3 group by payment_type 4 order by 3 desc;</pre>				
<div>Query results<div> SAVE RESULTS</div><div> EXPLORE DATA ▾</div></div>				
Query complete (0.9 sec elapsed, 4.5 GB processed)				
<div>Job information</div> <div>Results</div> <div>JSON</div> <div>Execution details</div>				
Row	payment_type	tips_percent	c	
1	Cash	0.0	113471003	
2	Credit Card	17.0	79238471	
3	No Charge	5.0	815483	
4	Unknown	1.0	671005	
5	Prcard	1.0	603078	
6	Mobile	15.0	433838	
7	Dispute	0.0	80872	

Индексы

Попробуем посчитать аналогичный запрос на нашем Постгресе:

```
taxi=# SELECT payment_type, round(sum(tips)/sum(tips+fare)*100) tips_persent, count(*)
taxi=# FROM taxi_trips
taxi=# group by payment_type
taxi=# order by 3 desc;
 payment_type | tips_persent | count
-----+-----+-----
Cash          |          0   | 113471003
Credit Card  |         18   |  79238471
No Charge     |          5   |   815483
Unknown       |          1   |   671005
Prcard        |          1   |   603078
Mobile        |         16   |   433838
Dispute       |          0   |    80872
Pcard         |          2   |    36874
Split         |         18   |     3442
Prepaid       |          0   |     1801
Way2ride      |         16   |      142
(11 rows)

Time: 1699838.826 ms (28:19.839)
```

Индексы

Индексы по отдельным полям как и тюнинг Постгреса нам не помогут - мы просто упираемся в дисковую подсистему.

Решение?

Индексы

Индексы по отдельным полям как и тюнинг Постгреса нам не помогут - мы просто упираемся в дисковую подсистему.

Решение?

Конечно **составной** индекс.

```
create index idx_taxi on taxi_trips(payment_type, tips, fare);
```

Время создания:

Time: 1370972.471 ms (22:50.972)

Также с 11 версии Постгреса можно использовать покрывающий индекс

```
create index idx_taxi2 on taxi_trips(payment_type) include (tips, fare);
```

Time: 932280.011 ms (15:32.280)

Индексы

Посмотрим план выполнения запроса после создания индекса:

```
taxi=# EXPLAIN SELECT payment_type, round(sum(tips)/sum(tips+fare)*100) tips_persent, count(*)
taxi=# FROM taxi_trips
taxi=# group by payment_type
taxi=# order by 3 desc;

                                QUERY PLAN
-----
Sort  (cost=12715579.86..12715579.88 rows=8 width=23)
  Sort Key: (count(*)) DESC
  -> Finalize GroupAggregate (cost=12715577.57..12715579.74 rows=8 width=23)
    Group Key: payment_type
    -> Gather Merge (cost=12715577.57..12715579.44 rows=16 width=31)
      Workers Planned: 2
      -> Sort (cost=12714577.55..12714577.57 rows=8 width=31)
        Sort Key: payment_type
        -> Partial HashAggregate (cost=12714577.35..12714577.43 rows=8 width=31)
          Group Key: payment_type
          -> Parallel Seq Scan on taxi_trips (cost=0.00..11667169.93 rows=83792593 width=23)

JIT:
  Functions: 7
  Options: Inlining true, Optimization true, Expressions true, Deforming true
(14 rows)
```

Индексы

Почему не используется индекс???

Ок, соберем статистику

ANALYZE taxi_trips;

Time: 3057.809 ms (00:03.058)

Индексы

<https://aristov.tech>

Не помогло...

отключим SEQ SCAN

<https://aristov.tech>

Индексы

Отключив SEQ SCAN мы наконец то увидели использование индекса, но!!! увидим выросшую сложность запроса и он будет выполняться значительно дольше простого SEQ SCAN !!!

```
taxi=# SET enable_seqscan = OFF;
SET
Time: 1.875 ms
taxi=# EXPLAIN SELECT payment_type, round(sum(tips)/sum(tips+fare)*100) tips_persent, count(*)
taxi=# FROM taxi_trips
taxi=# group by payment_type
taxi=# order by 3 desc;

                                QUERY PLAN
-----
Sort  (cost=399787847.34..399787847.36 rows=8 width=23)
  Sort Key: (count(*)) DESC
  -> GroupAggregate (cost=0.57..399787847.22 rows=8 width=23)
      Group Key: payment_type
      -> Index Only Scan using idx_taxi2 on taxi_trips (cost=0.57..397275169.08 rows=201014240 width=23)

JIT:
  Functions: 3
  Options: Inlining true, Optimization true, Expressions true, Deforming true
(8 rows)
```

Индексы

Что же делать?

Отключение JIT и куча других вариантов не помогла...

В предыдущих версиях работало из коробки...

Ответ таит документация:

*After adding or deleting a large number of rows, it might be a good idea to issue a **VACUUM ANALYZE** command for the affected table. This will update the system catalogs with the results of all recent changes, and allow the PostgreSQL query planner to make better choices in planning queries.*

Индексы

Наконец то видим параллельный индекс скан:

```
QUERY PLAN
-----
Sort (cost=3877164.38..3877164.40 rows=9 width=23)
  Sort Key: (count(*))
    -> Finalize GroupAggregate (cost=1000.59..3877164.24 rows=9 width=23)
      Group Key: payment_type
        -> Gather Merge (cost=1000.59..3877163.90 rows=18 width=31)
          Workers Planned: 2
            -> Partial GroupAggregate (cost=0.57..3876161.80 rows=9 width=31)
              Group Key: payment_type
                -> Parallel Index Only Scan using idx_payment_type_incl on taxi_trips (cost=0.57..3038455.04 rows=83770667 width=23)
JIT:
  Functions: 6
  Options: Inlining true, Optimization true, Expressions true, Deforming true
(12 rows)
```

Индексы

```
taxi=# SELECT payment_type, round(sum(tips)/sum(tips+fare)*100) tips_persent, count(*)
taxi=# FROM taxi_trips
taxi=# group by payment_type
taxi=# order by 3 desc;
 payment_type | tips_persent | count
-----+-----+-----
Cash          |          0 | 115623879
Credit Card  |         18 | 81251263
Prcard        |          1 | 1189109
Unknown       |          1 | 1072914
Mobile        |         16 | 1019082
No Charge     |          5 | 818879
Dispute       |          0 | 84820
Pcard         |          2 | 36874
Split         |         18 | 3442
Prepaid       |          0 | 1813
Way2ride      |         16 | 142
(11 rows)

Time: 56838.213 ms (00:56.838)
```

Индексы

Если интересно потренироваться есть бесплатная выгрузка на 10Гб в CSV:

Cloud Console URL - <https://console.cloud.google.com/storage/browser/chicago10>

[gsutil](#) URI - gs://chicago10

Индексы

Так что для нашего варианта:

- ❖ использовать [cstore_fdw](#)
- ❖ преагрегаты
- ❖ мат.представления
- ❖ триггеры
- ❖ ну или clickhouse %)

Индексы

Также индексы могут не работать:

- ❖ слишком мало данных в таблице (например справочник) и Постгресу быстрее извлечь все данные в память, чем бегать туда сюда
- ❖ используется функциональный индекс, но не совпадает количество или тип аргументов у функции
- ❖ используется составной индекс, но поиск (сортировка) ведется только по второму и дальнейшим ключам. Но иногда может и отработать - если данных хватит в индексе и он намного меньше размеров - полный index scan
- ❖ используется покрывающий индекс при наличие предыдущих ключей в запросе при сортировке 2 и дальнейших полей
- ❖ несоответствие кодировок
- ❖ индекс НЕвалидный (поломался или недостроился)
- ❖ по этому полю/полям уже есть аналогичный индекс

На что стоит обратить внимание:

<https://www.postgresql.org/docs/current/indexes-examine.html>

Индексы

Не применяйте частичные индексы в качестве замены секционированию!!!

У вас может возникнуть желание создать множество неперекрывающихся частичных индексов, например:

```
CREATE INDEX mytable_cat_1 ON mytable (data) WHERE category = 1;
```

```
CREATE INDEX mytable_cat_2 ON mytable (data) WHERE category = 2;
```

```
CREATE INDEX mytable_cat_3 ON mytable (data) WHERE category = 3;
```

...

```
CREATE INDEX mytable_cat_N ON mytable (data) WHERE category = N;
```

Но так делать не следует! Почти наверняка вам лучше использовать один не частичный индекс, объявленный так:

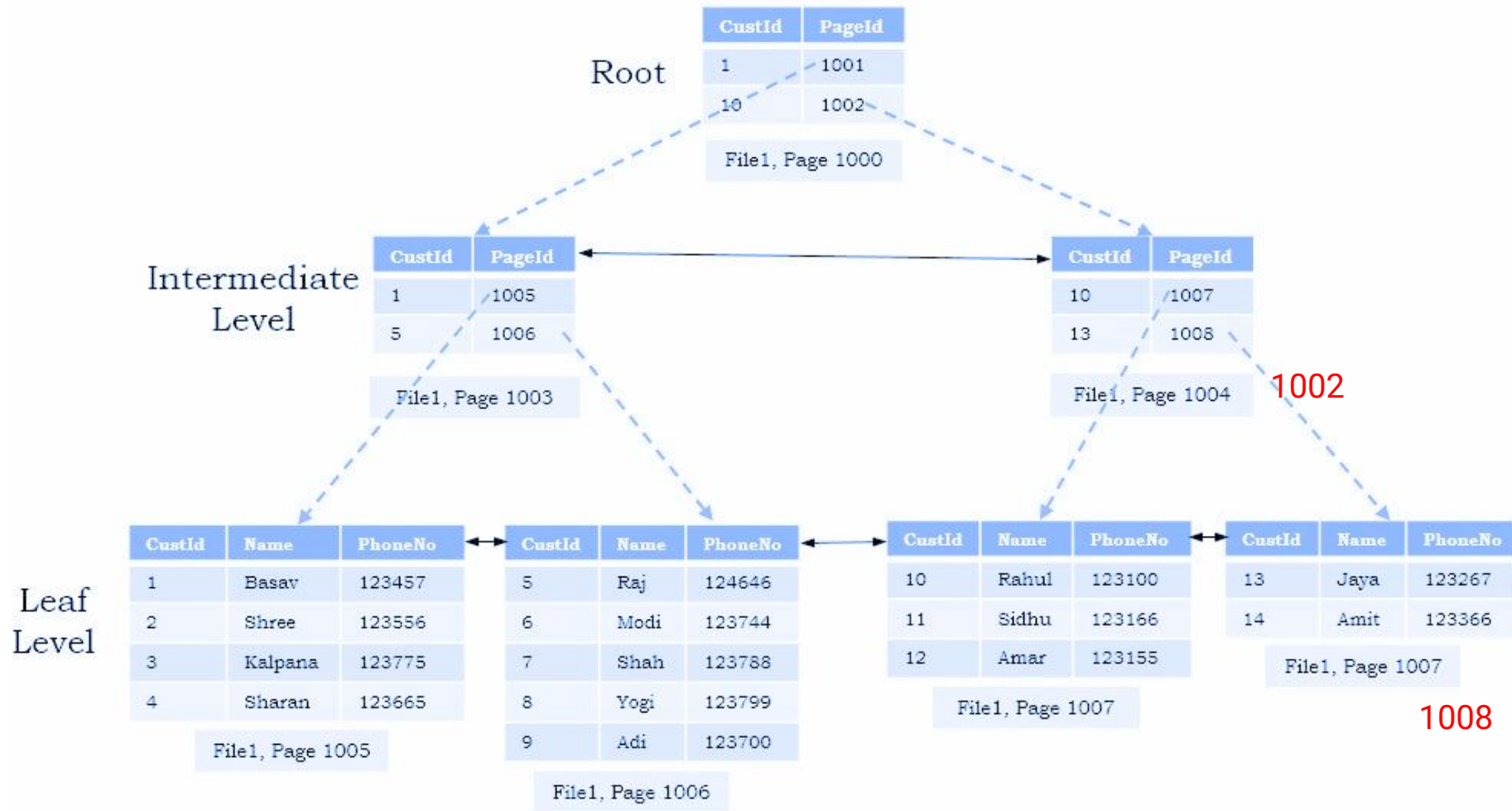
```
CREATE INDEX mytable_cat_data ON mytable (category, data);
```

Кластерный индекс

Пример кластерного btree+ индекса

B+ Tree Structure of a Clustered Index

<https://aristov.tech>



<https://aristov.tech>

<https://sqlhints.com/tag/clustered-index-b-tree-structure/>

Кластерный индекс

В результате кластеризации таблицы её содержимое физически переупорядочивается в зависимости от индекса.

Кластеризация является одноразовой операцией из-за работы механизма MVCC: последующие изменения в таблице нарушают порядок кластеризации

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

Также требует эксклюзивной блокировки!!

Что с этим делать?

Конкурентная постройка и перестройка индексов

Не забываем использовать!

Иначе построение/перестройка индекса требует **эксклюзивной** блокировки

<https://www.postgresql.org/docs/current/sql-reindex.html#SQL-REINDEX-CONCURRENTLY>

Конкурентная постройка секционированного индекса

Не работает(

```
-- При создании индекса на секционированной таблице нельзя указать CONCURRENTLY.
CREATE INDEX CONCURRENTLY idx_tickets_part_startdate2 ON tickets_part(startdate);
-- ERROR:  cannot create index on partitioned table "tickets_part" concurrently

-- Но можно поступить следующим образом. Сначала создаем индекс только на основной таблице, он получит статус invalid:

CREATE INDEX idx_tickets_part_startdate2 ON ONLY tickets_part(startdate);

SELECT indisvalid FROM pg_index WHERE indexrelid::regclass::text = 'idx_tickets_part_startdate2';
 indisvalid
-----
 f
(1 row)
```

Конкурентная постройка секционированного индекса

Далее создаем отдельные индексы конкурентно на каждую партицию:

```
-- Затем создаем индексы на всех секциях с опцией CONCURRENTLY:  
=> CREATE INDEX CONCURRENTLY tickets_part_2000_01_startdate_idx ON tickets_part_2000_01(startdate);  
..  
=> CREATE INDEX CONCURRENTLY tickets_part_2002_09_startdate_idx ON tickets_part_2002_09(startdate);
```


Конкурентная постройка секционированного индекса

Подключаем:

```
-- Теперь подключаем локальные индексы к глобальному:
=> ALTER INDEX idx_tickets_part_startdate2 ATTACH PARTITION tickets_part_2000_01_startdate_idx;
..
=> ALTER INDEX idx_tickets_part_startdate2 ATTACH PARTITION tickets_part_2002_09_startdate_idx;

-- Как только все индексные секции будут подключены, основной индекс изменит свой статус:
SELECT indisvalid FROM pg_index WHERE indexrelid::regclass::text = 'idx_tickets_part_startdate2';
 indisvalid
-----
t
(1 row)
```

Reindex

Reindex

Vacuum не помогает(

Как диагностировать распухание индекса:

❖ соотношение метрик -> 2 update

```
-[ RECORD 1 ]-----+-----  
version      | 4  
tree_level   | 2  
index_size   | 22487040  
root_block_no | 290  
internal_pages | 11  
leaf_pages   | 2733  
empty_pages  | 0  
deleted_pages | 0  
avg_leaf_density | 90.06  
leaf_fragmentation | 0
```

```
-[ RECORD 1 ]-----+-----  
version      | 4  
tree_level   | 2  
index_size   | 67403776  
root_block_no | 290  
internal_pages | 30  
leaf_pages   | 8197  
empty_pages  | 0  
deleted_pages | 0  
avg_leaf_density | 60.15  
leaf_fragmentation | 33.33
```

Обслуживание индексов

Обязательно смотрим состав индексов и перестраиваем:

- ❖ при большой фрагментарности листьев индекса
- ❖ при низкой средней плотности
- ❖ при большом количестве пустых или удаленных листьев

Обязательно мониторим использование индексов!!

- ❖ ненужные удаляем
- ❖ отсутствующие добавляем (seqscan)
 - если таблица маленькая - seqscan быстрее
 - если большая селективность/низкая кардинальность (м/ж) - смысла нет

На время создания индекса влияет **параметр maintenance_work_mem**

Reindex

- ❖ когда запускать обслуживание
 - желательно в момент наименьшей нагрузки
- ❖ Не забываем про **CONCURRENTLY**
 - иначе требуется эксклюзивная блокировка
 - возможно есть смысл периодического CLUSTER - тяжело добиться
 - дорогая реализация аналога кластерного индекса - делаем покрывающий индекс по нужному полю, который включает в себя остальные поля
 - можно делать несколько таких индексов %)
- ❖ *Интересный вопрос: в БД 1 млн индексов, из них 10к имеют высокий bloat, и нужно эти индексы перестроить в N потоков и так, чтобы не положить базу*

Reindex

<https://aristov.tech>

❖ *А мы ничего не забыли про реиндекс?*

Может какие-то объекты, имеющие свои индексы?)

<https://aristov.tech>

Reindex

<https://aristov.tech>

❖ *А мы ничего не забыли про реиндекс?*

А TOAST сегменты то имеют свой индекс, который также пухнет!

<https://aristov.tech>

Обслуживание GIN индекса

В отличие от Btree индексов, элементы никогда не удаляются из GIN-индекса.

<https://habr.com/ru/companies/postgrespro/articles/340978/>

Считается, что значения, содержащие элементы, могут пропадать, появляться, изменяться, но набор элементов, из которых они состоят — довольно статичен. Такое решение существенно упрощает алгоритмы, обеспечивающие параллельную работу с индексом нескольких процессов.

Но в целом рекомендуется REINDEX INDEX CONCURRENTLY в моменты наименьшей нагрузки

Воркшопы:

Многие моменты из лекции в примерах рассматриваются в

[Воркшопы от aristov.tech - Блог по архитектуре и оптимизации PostgreSQL от Аристова Евгения](#)

Индексы и текстовый поиск

- ❖ замена ILIKE - index on lower(text)
- ❖ to_tsvector - полнотекст (GIN index) - при этом делаем функциональный индекс, а НЕ ДОБАВЛЯЕМ поле
- ❖ Ищем имена с опечатками в PostgreSQL - trigrams - <https://habr.com/ru/articles/341142/>

Холиварный вопрос. Индексы и внешние ключи

<https://aristov.tech>

Плюсы:

- ❖ **Целостность данных:** Индексы на внешние ключи обеспечивают целостность данных, контролируя, что значения в столбце ссылающегося ключа существуют в связанной таблице. Это предотвращает нарушение ссылочной целостности.
- ❖ **Ускорение JOIN-операций:** Индексы на внешние ключи улучшают производительность операций объединения (JOIN) между таблицами, так как PostgreSQL может использовать индексы для эффективного объединения данных.
- ❖ **Улучшенная оптимизация запросов:** Оптимизатор запросов может использовать индексы на внешние ключи для выбора оптимальных планов выполнения запросов, что может сократить время выполнения запросов.
- ❖ **Улучшение производительности DELETE и UPDATE:** Индексы на внешние ключи могут ускорить операции удаления и обновления записей, так как они помогают быстро определить, какие записи должны быть удалены или обновлены.

Холиварный вопрос. Индексы и внешние ключи

Минусы:

- ❖ **Затраты на обслуживание:** Индексы требуют дополнительного пространства на диске и добавляют небольшие накладные расходы на обслуживание вставок, обновлений и удалений данных. Это может повлиять на производительность в некоторых сценариях.
- ❖ **Обновление индексов при изменении данных:** Индексы на внешние ключи должны обновляться при изменении данных в связанных таблицах. Это может повлечь за собой небольшие накладные расходы на производительность при выполнении больших операций обновления или удаления данных.
- ❖ **Выбор правильных индексов:** Неверный выбор индексов на внешние ключи или их избыточное создание может привести к излишнему использованию ресурсов и ухудшению производительности.

Холиварный вопрос. Индексы и внешние ключи

Случай из жизни:

Все было отлично... База 10 гигабайт...

Внезапно приложение начинает тормозить, обычные операции выполняются вместо нескольких секунд минуты, в журнале приложения ошибки Unable to acquire Sql connection

Проц в полку...

Смотрим на топ запросов и видим:

Запрос простой **select f1, f2...f7 from table where fk_id in (c1, c2, c3 ... c53);**

План тоже простой - sec_scan, rows 20, cost 250

Что пошло не так?)

строк в таблице всего 500 000...

Холиварный вопрос. Индексы и внешние ключи

Случай из жизни:

Все было отлично... База 10 гигабайт...

Внезапно приложение начинает тормозить, обычные операции выполняются вместо нескольких секунд минуты, в журнале приложения ошибки Unable to acquire Sql connection

Проц в полку...

Смотрим на топ запросов и видим:

Запрос простой **select f1, f2...f7 from table where fk_id in (c1, c2, c3);**

План тоже простой - sec_scan, rows 20, cost 250

Что пошло не так?)

Все данные в памяти и проц просто их молотит каждый раз все во вложенных циклах...

Холиварный вопрос. Индексы и внешние ключи

Рекомендация:

в 90% случаев строить - но лучше все тестировать на реальных кейсах

Найти FK без индексов:

https://github.com/pgexperts/pgx_scripts/blob/master/indexes/fk_no_index.sql

Доп.материал:

[Postgres и индексы на внешних ключах и первичных ключах](#)

[Индексы на поля с foreign key / PostgreSQL / Sql.ru](#)

В PostgreSQL 17 всё неоднозначно!

Индексы работать стали быстрее и индексы FK в тайских перевозках не помогли...

А где смотреть что как? pg_stat*

Также существует ЕЩЕ ряд системных представлений для сбора статистики:

- ❖ pg_stat_io
- ❖ pg_statio_all_tables
- ❖ pg_statio_all_indexes
- ❖ pg_store_plans
- ❖ pg_stat_all_tables
- ❖ pg_stat_all_indexes
- ❖ pg_stat_kcache
- ❖ pg_wait_sampling
- ❖ pg_buffercache
- ❖ pgstattuple
- ❖ pg_stat_archiver

на самом деле 33-50 штук)

HypoPG

НуроPG

Не всегда мы можем со 100% гарантией сказать - поможет ли нам добавление индекса. Физически проверять гипотезу может быть дорого. Для этого создан инструмент виртуальных индексов.

<https://github.com/HypoPG/hypopg>

Итоговый чеклист

Checklist

- ❖ используем индексы исходя из запросов
- ❖ убираем неиспользуемые индексы
- ❖ добавляем новые по потребности
- ❖ помним про HуроPG
- ❖ учитываем особенности массовой вставки данных
- ❖ используем индексы на внешние ключи исходя из анализа запросов
- ❖ не забываем обслуживать индексы

Проект aristov.tech

aristov.tech

Книги по 14 Постгресу (Архитектуре/16 Оптимизации) <https://aristov.tech/#orderbook>

Эксклюзивный **курс** по Оптимизации Постгреса - 5 поток - постоянно дорабатывается исходя из бизнес-кейсов <https://aristov.tech/blog/kurs-po-optimizaczii-postgresql/>

Со всеми **отзывами** без цензуры можно ознакомиться по ссылке <https://aristov.tech/blog/otzivi-kurs/>

Блог с популярными темами <https://aristov.tech/blog>

ТГ **канал** с новостями блога и проекта https://t.me/aristov_tech

Ютуб канал с интересными видео <https://www.youtube.com/@aristovtech>

Рутуб - <https://rutube.ru/u/aristovtech/>

ВКВидео - <https://vkvideo.ru/@aristov.tech>

Моя **группа** ДБА <https://t.me/dbaristov> - уже ~350 экспертов

Открытие вебинары (5 штук + этот) - <https://aristov.tech/blog/otkrytye-lekczii-ot-aristov-tech/>

Воркшопы (пока 5) - <https://aristov.tech/blog/vorkshopy-ot-aristov-tech/>

aristov.tech

Также за прошедшее время был запущен проект **менторинга** в котором уже участвует 17 лучших экспертов в отрасли ДБА/DevOps и разработки.

Ознакомится с преподавателями и проектом <https://aristov.tech/mentorship>

В направлении крутых **вакансий** уже 5 предложений от партнёров

<https://aristov.tech/blog/vakansii-ot-partnerov-aristov-tech/>

Всегда можно со мной связаться через сайт для консультации, аудита вашего проекта, менторинга, обучения и многого другого

aristov.tech

Розыгрыш книг и скидки 30% на курс (только очные варианты)
регистрируемся по форме:

<https://forms.gle/9i8eRcSHGLnMXi3B7>

Сам розыгрыш в db-fiddle

<https://www.db-fiddle.com/f/43wbuUzv7YUAcS3aypkvvC/1>

ДЗ

ДЗ

1. Подписаться на канал
2. Подписаться на Ютуб
3. Вступить в мою группу ДБА
4. Посещать открытые уроки
5. Посещать воркшопы или посмотреть их в записи
6. Прийти на курс %)
7. Прокачаться с помощью менторинга
8. Устроиться на одну из вакансий 450+

П.С. Книги просто бомбические

Спасибо за внимание!

Следующий ОУ в апреле 2025

Аристов Евгений