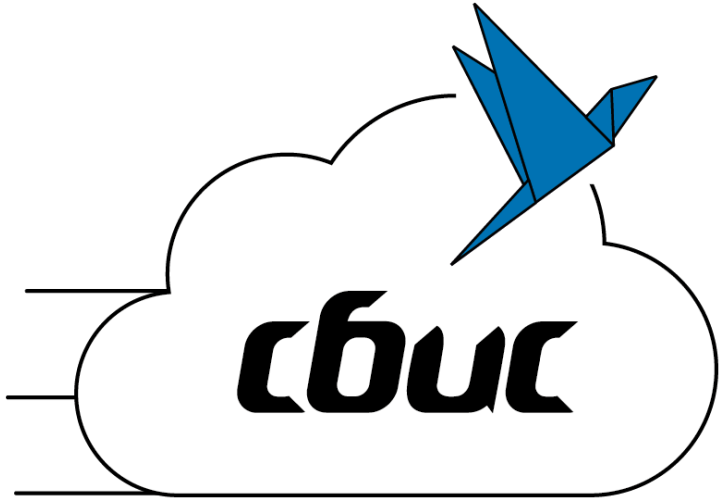


PostgreSQL для начинающих

#5: индексы

Кирилл Боровиков / Компания «Тензор», технический директор / explain.tensor.ru, sbis.ru

Индекс – подсказка для СУБД

Зачем нужны индексы?

```
CREATE TABLE pg_class_copy(
  LIKE pg_class -- «по образу и подобию»
, id serial     -- автоматически создали последовательность
);

INSERT INTO pg_class_copy
  TABLE pg_class; -- скопировали все содержимое
```

Зачем нужны индексы?

```
EXPLAIN (ANALYZE, TIMING off, COSTS off, SUMMARY off)
  SELECT oid FROM pg_class_copy WHERE relkind = 'S'; -- sequence
```

```
Seq Scan on pg_class_copy (actual rows=1 loops=1)
  Filter: (relkind = 'S'::"char")
  Rows Removed by Filter: 413
```

* без индексов приходится читать всю таблицу

Создаем индекс

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

https://postgrespro.ru/docs/postgresql/15/sql-createindex

Создаем индекс

Простая индексация

```
CREATE INDEX ON pg_class_copy(relkind); -- по одному полю
```

```
Index Scan using pg_class_copy_relkind_idx on pg_class_copy (actual rows=1 loops=1)
  Index Cond: (relkind = 'S'::"char")
```

```
DROP INDEX pg_class_copy_relkind_idx; -- удаляем индекс
```

Создаем индекс

Простая индексация

```
CREATE INDEX ON pg_class_copy(relkind, oid); -- по нескольким полям
```

```
Index Only Scan using pg_class_copy_relkind_oid_idx on pg_class_copy (actual rows=1 loops=1)
  Index Cond: (relkind = 'S'::"char")
  Heap Fetches: 1
```

```
DROP INDEX pg_class_copy_relkind_oid_idx;
```

Создаем индекс

Неключевые поля

```
CREATE INDEX ON pg_class_copy(relkind)
  INCLUDE(oid); -- включение неключевых полей
```

```
Index Only Scan using pg_class_copy_relkind_oid_idx on pg_class_copy (actual rows=1 loops=1)
  Index Cond: (relkind = 'S'::"char")
  Heap Fetches: 1
```

```
DROP INDEX pg_class_copy_relkind_oid_idx;
```

Создаем индекс

Неключевые поля (профит – в размере)

```
CREATE TABLE idx_tbl AS
  SELECT i % 1000 a, i % 10000 b FROM generate_series(1, 1e6) i;
CREATE INDEX idx_fld ON idx_tbl(a, b);
CREATE INDEX idx_inc ON idx_tbl(a) INCLUDE(b);
SELECT relname, relpages FROM pg_class WHERE relname LIKE 'idx_%';
```

```
idx_fld | 3858
idx_inc | 3852 -- с INCLUDE на 6 меньше!
idx_tbl | 5406 -- суммарный объем индексов больше таблицы
```

Создаем индекс

Индекс по выражению

```
CREATE INDEX ON pg_class_copy((relkind = 'S')) -- выражение
  INCLUDE(oid);
```

```
Index Scan using pg_class_copy_expr_oid_idx on pg_class_copy (actual rows=1 loops=1)
  Index Cond: ((relkind = 'S'::"char") = true)
```

```
DROP INDEX pg_class_copy_expr_oid_idx;
```

* bool в поле индекса – это плохо!

Создаем индекс

Частичный индекс (условие можно не проверять)

```
CREATE INDEX ON pg_class_copy(oid)
  WHERE relkind = 'S'; -- условие включения записей в индекс
```

```
Index Only Scan using pg_class_copy_oid_idx on pg_class_copy (actual rows=1 loops=1)
  Heap Fetches: 1
```

```
DROP INDEX pg_class_copy_oid_idx;
```

Создаем индекс

Учитываем мутабельность выражений

https://postgrespro.ru/docs/postgresql/15/xfunc-volatility

expr	idx
`ts >= now() + '1 day'::interval`	`(ts)`
`ts - '1 day'::interval >= now()`	`((ts - '1 day'::interval))`
`ts - now() >= '1 day'::interval`	`-- никак`

* иногда запрос стоит переписать!

Создаем индекс

Избегаем блокировок

```
BEGIN;
CREATE INDEX ON pg_class_copy ...;
  ...
  ...
  ...
  ...
COMMIT; / ROLLBACK;
```

```
-- другое подключение
SELECT ... FROM pg_class_copy ...;
  -- выполняется быстро
INSERT INTO pg_class_copy ...;
  -- ждет блокировку
  ...
  ... до конца блокирующей транзакции
```

Создаем индекс

Избегаем блокировок (**CONCURRENTLY**)

```
BEGIN;
CREATE INDEX CONCURRENTLY
  ON pg_class_copy ...;
ERROR:  CREATE INDEX CONCURRENTLY
cannot run inside a transaction block
ROLLBACK;
```

Создаем индекс

Избегаем блокировок (**CONCURRENTLY**)

```
CREATE INDEX CONCURRENTLY
  ON pg_class_copy ...;
  ...
  ...
  ...
  ...
```

```
-- другое подключение
  ...
INSERT INTO pg_class_copy ...;
  -- выполняется быстро
```

Создаем индекс

Избегаем дублей в данных (**UNIQUE**)

```
CREATE UNIQUE INDEX ON pg_class_copy(oid);
INSERT INTO pg_class_copy TABLE pg_class; -- повторно добавляем
ERROR:  duplicate key value violates unique constraint "pg_class_copy_oid_idx"
DETAIL:  Key (oid)=(2690) already exists.
```

```
-- можно объявить PK, используя уже готовый UNIQUE
ALTER TABLE pg_class_copy
  ADD PRIMARY KEY USING INDEX pg_class_copy_oid_idx;
```

Создаем индекс

Отслеживаем ход создания

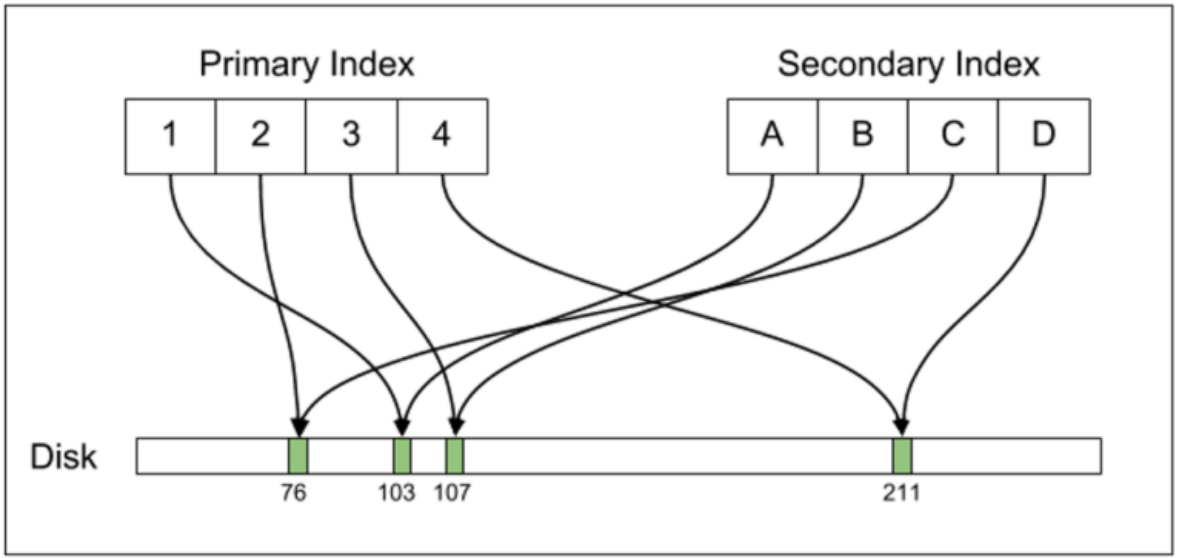
```
SELECT * FROM pg_stat_progress_create_index;
```

https://postgrespro.ru/docs/postgresql/15/progress-reporting#CREATE-INDEX-PROGRESS-REPORTING

Обслуживание индексов

Плохо: неиспользуемые индексы (Write Amplification)

https://habr.com/ru/companies/slurm/articles/322624/



Обслуживание индексов

Плохо: неиспользуемые индексы

решение: `DROP INDEX CONCURRENTLY`

```
-- сбрасываем накопленную статистику
SELECT pg_stat_reset();
-- ждем... копим...
SELECT * FROM pg_stat_user_indexes WHERE idx_scan = 0;
-- находим все индексы, к которым так и не было обращений
DROP INDEX CONCURRENTLY ...;
```

Обслуживание индексов

Плохо: раздувшиеся индексы

решение: REINDEX CONCURRENTLY

```
REINDEX [ ( параметр [, ...] ) ] { INDEX | TABLE | SCHEMA | DATABASE | SYSTEM } [ CONCURRENTLY ] имя

Здесь допускается параметр:

    CONCURRENTLY [ логическое_значение ]
    TABLESPACE новое_табл_пространство
    VERBOSE [ логическое_значение ]
```

https://postgrespro.ru/docs/postgresql/15/sql-reindex

Типы индексов

Встроенные

btree, **hash**, **gist**, **spgist**, **gin**, **brin**

https://postgrespro.ru/docs/postgresql/15/indexes-types

Можно писать свои!

bloom, **vodka**, **rum** :)

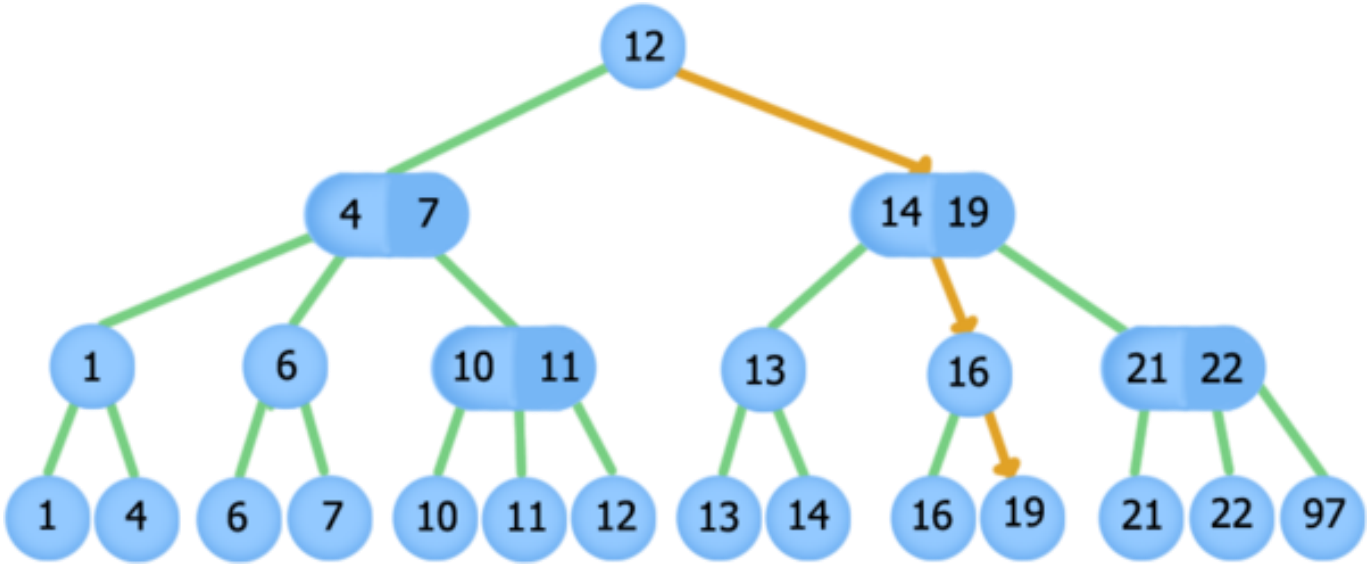
https://postgrespro.ru/docs/postgresql/15/xindex

btree

Balanced Tree

сбалансированное дерево, B-tree

https://postgrespro.ru/docs/postgresql/15/btree



btree

операторы **линейного порядка**

(<, <=, =, >=, >)

числовые и хронологические типы

smallint, integer, bigint, numeric, real, double precision

date, time/timestamp [with/without time zone]

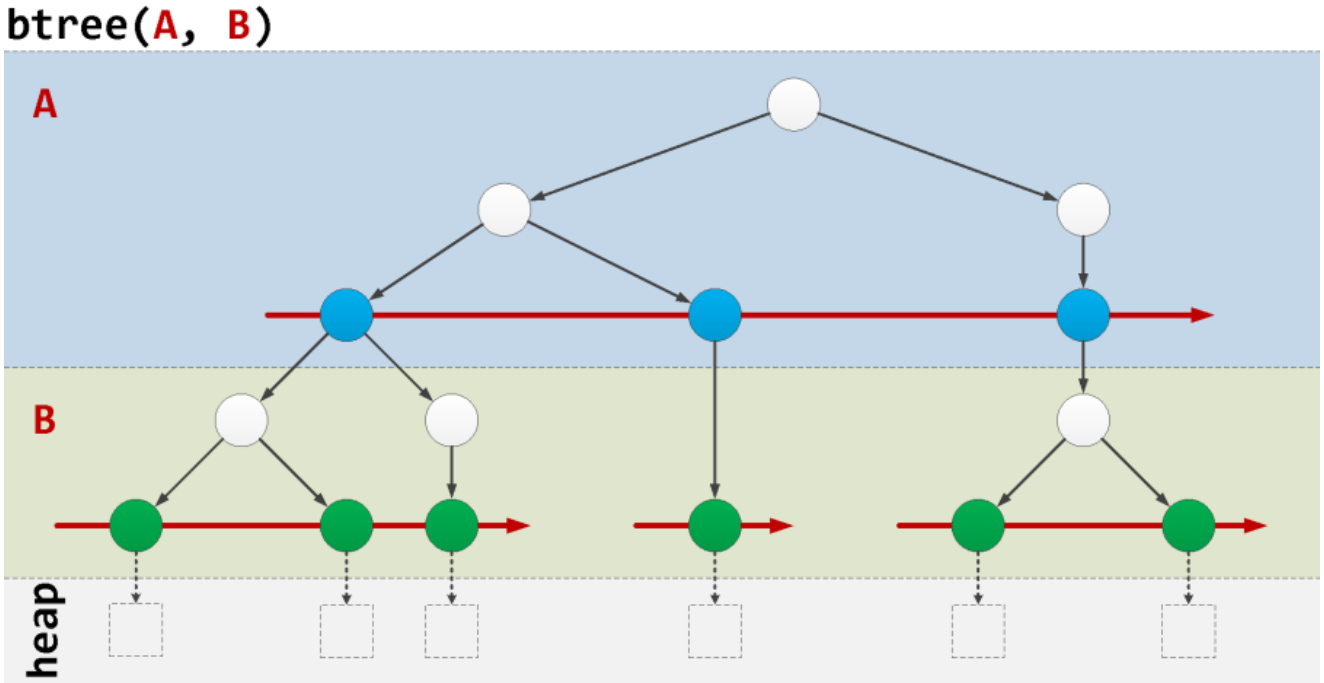
текстовые типы

varchar, text

uuid

btree

https://habr.com/ru/companies/tensor/articles/488104/



btree

Хорошо: префикс-условие

btree(A, B, C) уже включает

btree(A, B) и

btree(A)

btree

Хорошо: все константы (кроме последнего поля)

```
A = constA AND B [op] constB / op : { =, >, >=, <, <= }

A = constA AND B BETWEEN constB1 AND constB2

A = constA AND B = ANY(...)

A = constA AND B IN (...)

A = constA ORDER BY B
```

btree

Хорошо: все константы (кроме последнего поля)

```
A [op] constA / op : { =, >, >=, <, <= }

A BETWEEN constA1 AND constA2

A = ANY(...)

A IN (...)

ORDER BY A, B
```

btree

Плохо: полный перебор «слоя»

```
A <> const / <> ALL / <> ANY / NOT IN
```

неравенство

```
B [op] const / BETWEEN / = ANY / IN
```

```
ORDER BY B
```

пропуск предстоящих ключей

```
ORDER BY B, A
```

другой порядок сортировки

btree

Плохо: интервал/набор не в последнем поле

```
A [op] const / op : { >, >=, <, <= } без «=»

A BETWEEN ...

A = ANY(...)

A IN (...)                          ... AND B ... / ORDER BY B
```

* классика: **start <= const AND const <= finish**

btree

Плохо: выражение вместо поля

```
A - const1 [op] const2
```

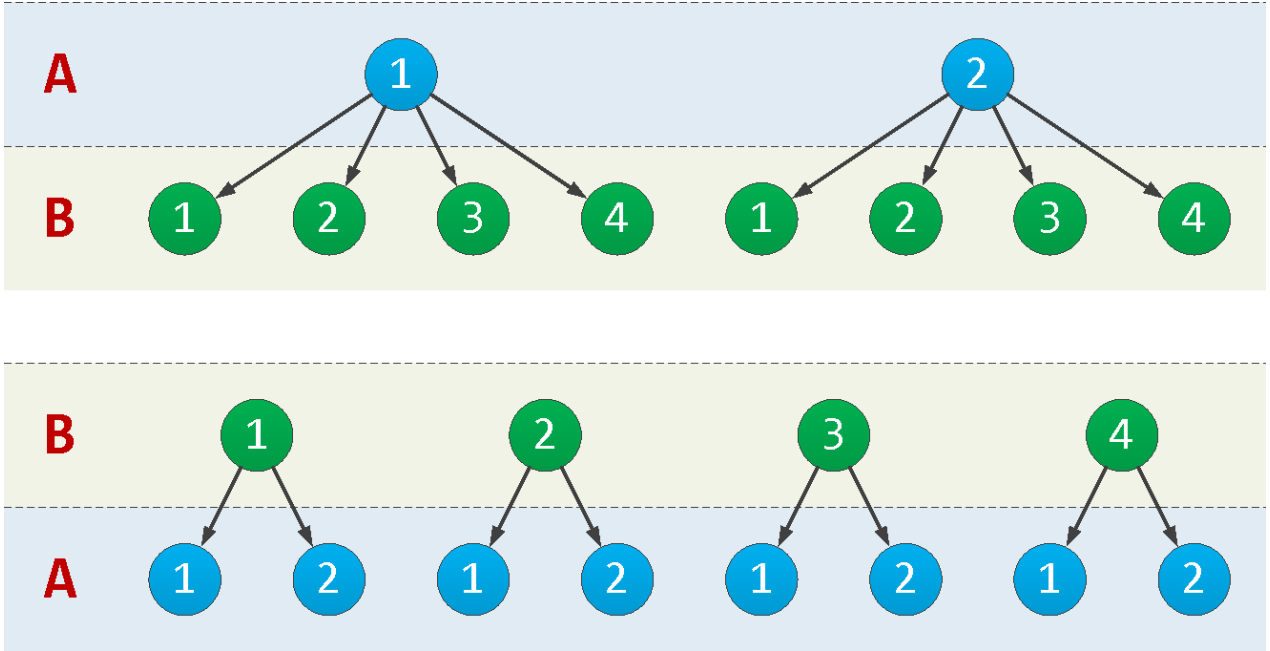
исправляем: `A [op] const1 + const2`

```
A::typeOfConst = const
```

исправляем: `A = const::typeOfA`

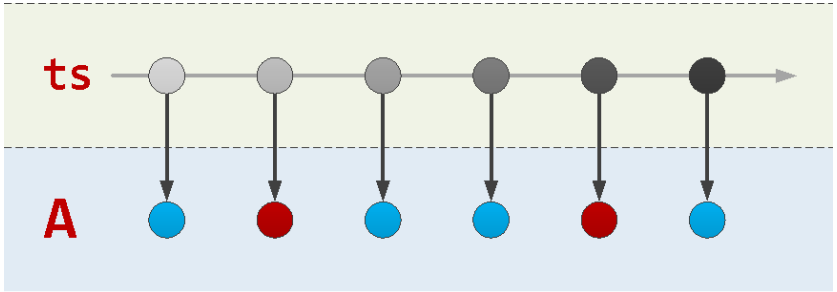
btree

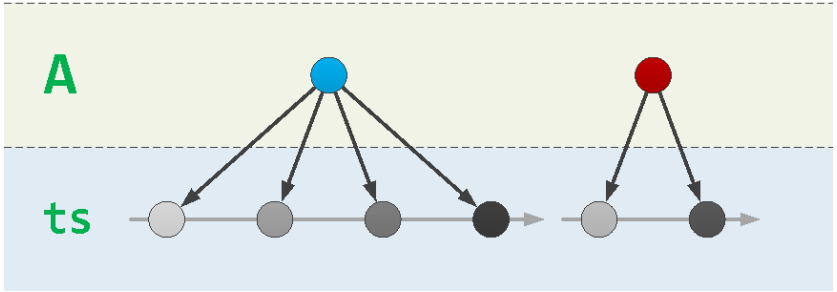
Плохо: неправильный учет кардинальности



btree

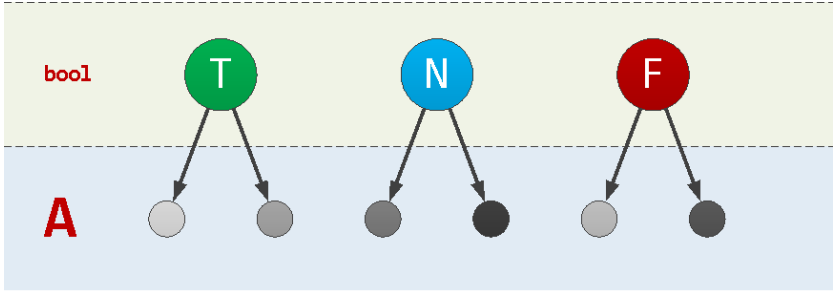
Плохо: `timestamp` «в середине»

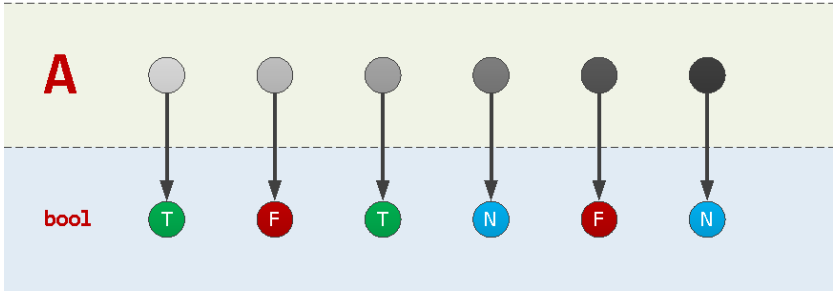




btree

Плохо: boolean в ключевом поле (неселективность)





btree

Плохо: **массив** в ключевом поле

для операторов **`<@, @>, &&`** есть **другие типы индексов**

для доступа к элементу **`arr[i]`** все равно **не работает**

btree

Плохо: NULL-записи в индексе

решение: условные индексы

```
(A, B) WHERE A IS NOT NULL OR B IS NOT NULL

       WHERE (A, B) IS DISTINCT FROM NULL

(A, B) WHERE A IS NOT NULL AND B IS NOT NULL

       WHERE (A, B) IS NOT NULL
```

btree

Подозрительно: BitmapAnd

Подозрительно: Limit/Sort/Scan

https://habr.com/ru/companies/tensor/articles/659889/

btree

Подозрительно: BitmapAnd

```
CREATE TABLE tst_eav AS
SELECT
  (random() * 1e4)::integer e -- 10k объектов
, (random() * 1e2)::integer a -- 100 характеристик
, (random() * 1e2)::integer v -- 100 вариантов значений
FROM
  generate_series(1, 1e6);    -- 1M записей о значениях
```

btree

Подозрительно: BitmapAnd

```
CREATE INDEX ON tst_eav(a);
CREATE INDEX ON tst_eav(v); -- два независимых индекса

-- отбираем по пересечению условий
EXPLAIN (ANALYZE, BUFFERS, TIMING off, COSTS off, SUMMARY off)
  SELECT * FROM tst_eav WHERE a = 1 AND v = 1;
```

btree

Подозрительно: BitmapAnd

```
Bitmap Heap Scan on tst_eav (actual rows=99 loops=1)
  Recheck Cond: ((v = 1) AND (a = 1))
  Heap Blocks: exact=98
  Buffers: shared hit=120
  ->  BitmapAnd (actual rows=0 loops=1)
        Buffers: shared hit=22
        ->  Bitmap Index Scan on tst_eav_v_idx (actual rows=9845 loops=1)
              Index Cond: (v = 1)
              Buffers: shared hit=11
        ->  Bitmap Index Scan on tst_eav_a_idx (actual rows=10139 loops=1)
              Index Cond: (a = 1)
              Buffers: shared hit=11
```

btree

Подозрительно: BitmapAnd

решение: составной индекс

```
CREATE INDEX ON tst_eav(a, v); -- можно попробовать и другие:
-- (a) WHERE v = 1
-- (v) WHERE a = 1
-- (?) WHERE a = 1 AND v = 1
```

```
Index Scan using tst_eav_a_v_idx on tst_eav (actual rows=99 loops=1)
  Index Cond: ((a = 1) AND (v = 1))
  Buffers: shared hit=101 -- сэкономили 19 страниц!
```

btree

Подозрительно: BitmapAnd

```
Index Scan using tst_eav_a_v_idx on tst_eav (actual rows=99 loops=1)
  Index Cond: ((a = 1) AND (v = 1))
  Buffers: shared hit=101 -- сэкономили 19 страниц!
```

btree

Подозрительно: Limit/Sort/Scan

```
DROP INDEX tst_eav_a_idx;
DROP INDEX tst_eav_v_idx;
DROP INDEX tst_eav_a_v_idx;

-- отбираем минимум в рамках ключа
EXPLAIN (ANALYZE, BUFFERS, TIMING off, COSTS off, SUMMARY off)
  SELECT * FROM tst_eav WHERE a = 1 ORDER BY v LIMIT 1;
```

btree

Подозрительно: Limit/Sort/Scan

```
Limit (actual rows=1 loops=1)
  Buffers: shared hit=5406
  ->  Sort (actual rows=1 loops=1)
        Sort Key: v
        Sort Method: top-N heapsort  Memory: 25kB
        Buffers: shared hit=5406
        ->  Seq Scan on tst_eav (actual rows=10139 loops=1)
              Filter: (a = 1)
              Rows Removed by Filter: 989861
              Buffers: shared hit=5406
```

btree

Подозрительно: Limit/Sort/Scan

```
CREATE INDEX ON tst_eav(a);
```

```
Limit (actual rows=1 loops=1)
  Buffers: shared hit=4590
  ->  Sort (actual rows=1 loops=1)
        Sort Key: v
        Sort Method: top-N heapsort  Memory: 25kB
        Buffers: shared hit=4590
        ->  Bitmap Heap Scan on tst_eav (actual rows=10139 loops=1)
              Recheck Cond: (a = 1)
              Heap Blocks: exact=4579
              Buffers: shared hit=4590
              ->  Bitmap Index Scan on tst_eav_a_idx (actual rows=10139 loops=1)
                    Index Cond: (a = 1)
                    Buffers: shared hit=11
```

btree

Подозрительно: Limit/Sort/Scan

решение: ключ сортировки в «хвосте» индекса

```
CREATE INDEX ON tst_eav(a, v);
```

```
Limit (actual rows=1 loops=1)
  Buffers: shared hit=4
  ->  Index Scan using tst_eav_a_v_idx on tst_eav (actual rows=1 loops=1)
        Index Cond: (a = 1)
        Buffers: shared hit=4
```

btree

Индексный **префиксный** `LIKE`-поиск (бинарный)

```
EXPLAIN (ANALYZE, BUFFERS, TIMING off, COSTS off, SUMMARY off)
  SELECT * FROM pg_class_copy
  WHERE
    relname LIKE 'pg\_class\_%'
  ORDER BY
    relname
  LIMIT 1;
```

btree

Индексный **префиксный** `LIKE`-поиск (бинарный)

```
Limit (actual rows=1 loops=1)
  Buffers: shared hit=11
  ->  Sort (actual rows=1 loops=1)
        Sort Key: relname
        Sort Method: top-N heapsort  Memory: 25kB
        Buffers: shared hit=11
        ->  Seq Scan on pg_class_copy (actual rows=8 loops=1)
              Filter: (relname ~~ 'pg\_class\_%'::text)
              Rows Removed by Filter: 409
              Buffers: shared hit=11
```

btree

Индексный **префиксный** LIKE-поиск (бинарный)

```
CREATE INDEX ON pg_class_copy(relname);
```

```
Limit (actual rows=1 loops=1)
  Buffers: shared hit=3
  ->  Index Scan using pg_class_copy_relname_idx on pg_class_copy (actual rows=1 loops=1)
        Index Cond: ((relname >= 'pg_class_'::text) AND (relname < 'pg_class`'::text))
        Filter: (relname ~~ 'pg\_class\_%'::text)
        Buffers: shared hit=3
```

btree

операторы **префиксного поиска**

(`~<~`, `~<=~`, `~`, `~>=~`, `~>~`)

текстовые типы (с использованием нужных **классов операторов**)

`varchar`, `text`

https://postgrespro.ru/docs/postgresql/15/indexes-opclass

btree

Индексный **префиксный** LIKE-поиск (текстовый)

https://habr.com/ru/companies/tensor/articles/491162/

```
CREATE TABLE prefix AS
  SELECT md5(i::text) FROM generate_series(1, 1e6) i;

EXPLAIN (ANALYZE, BUFFERS, TIMING off, COSTS off, SUMMARY off)
  SELECT * FROM prefix WHERE md5 LIKE '0000%';
```

btree

Индексный **префиксный** `LIKE`-поиск (текстовый)

```
CREATE INDEX ON prefix(md5);
```

```
Seq Scan on prefix (actual rows=13 loops=1)
  Filter: (md5 ~~ '0000%'::text)
  Rows Removed by Filter: 999987
  Buffers: shared hit=8334
```

btree

Индексный **префиксный** LIKE-поиск (текстовый)

```
CREATE INDEX ON prefix(md5 text_pattern_ops);
```

```
Index Only Scan using prefix_md5_idx1 on prefix (actual rows=13 loops=1)
  Index Cond: ((md5 ~>=~ '0000'::text) AND (md5 ~<~ '0001'::text))
  Filter: (md5 ~~ '0000%'::text)
  Heap Fetches: 0
  Buffers: shared hit=4
```

* **text_pattern_ops** – это класс операторов

btree

ОПЦИИ

```
CREATE INDEX ... WITH (
  deduplicate_items = 'on'
);
```

https://postgrespro.ru/docs/postgresql/15/sql-createindex#INDEX-RELOPTION-DEDUPLICATE-ITEMS

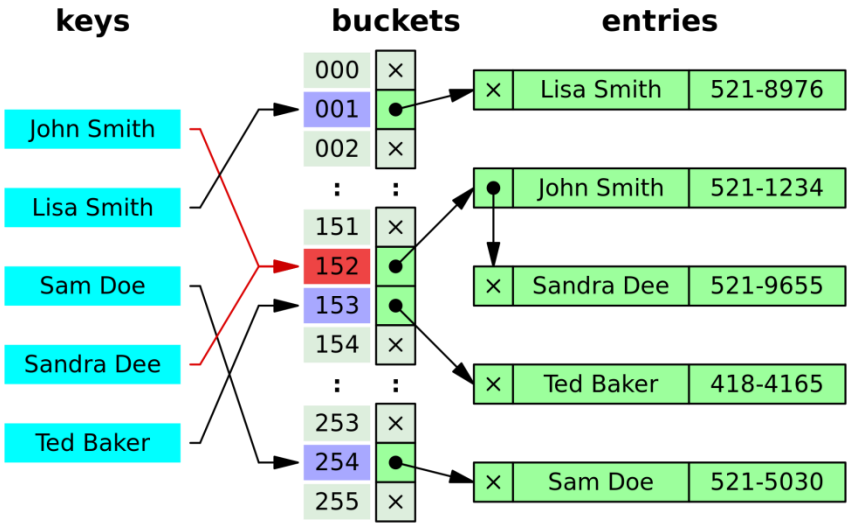
https://postgrespro.ru/docs/postgresql/15/btree-implementation#BTREE-DEDUPLICATION

* PostgreSQL 13+

hash

4-байтовый хэш

https://postgrespro.ru/docs/postgresql/15/hash-index



hash

только оператор **равенства** (=)

типы аналогичны **btree**

только одно поле

```
CREATE INDEX ON tst_eav USING hash(a, v);
ERROR:  access method "hash" does not support multicolumn indexes
```

hash

Профит – в размере!

```
CREATE INDEX prefix_btree ON prefix(md5); -- USING btree
CREATE INDEX prefix_hash ON prefix USING hash(md5);
```

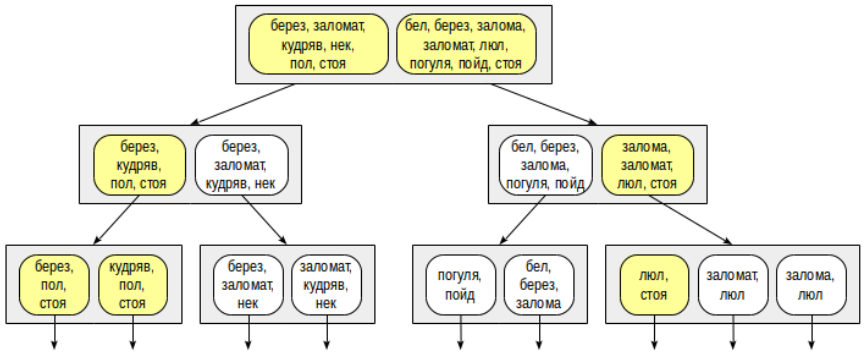
```
prefix       | 8334
prefix_btree | 7210
prefix_hash  | 4098 -- почти в 2 раза компактнее!
```

gist

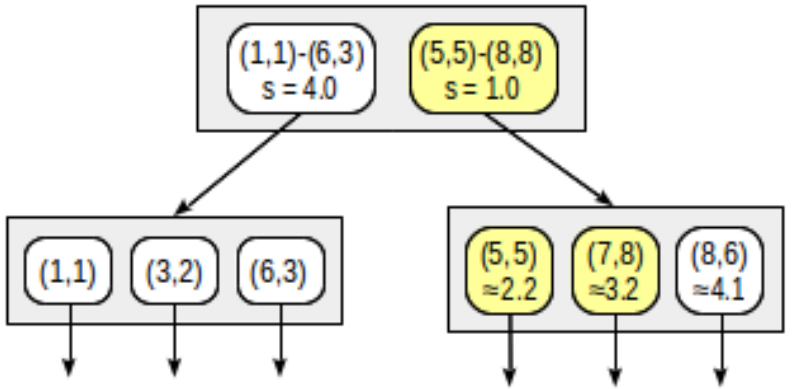
Generalized Search Tree

обобщенное поисковое дерево

универсальная инфраструктура (B-Tree, R-Tree, RD-Tree)



https://postgrespro.ru/docs/postgresql/15/gist



gist

операторы **геометрических отношений**

(`<<`, `&<`, `&>`, `>>`, `<@`, `@>`, `~=`, `&&`, `<<|`, `&<|`, `|&>`, `|>>`, `~`, `@`)

оператор **упорядочивания** (`<->`) для kNN-search

геометрические типы

`box`, `circle`, `poly`, `point`

https://postgrespro.ru/docs/postgresql/15/functions-geometry

gist

Принадлежность точки объекту

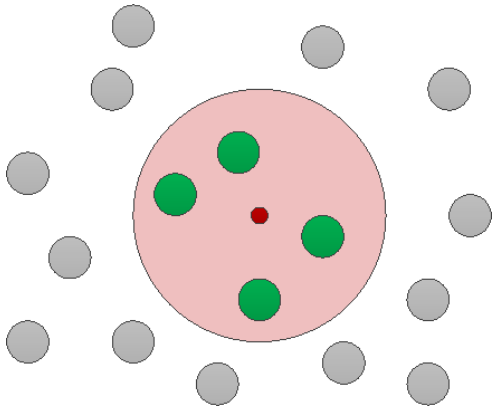
```
CREATE TABLE points AS -- миллион случайных точек
  SELECT point(random(), random()) p FROM generate_series(1, 1e6);

CREATE INDEX ON points USING gist(p);
```

gist

Принадлежность точки объекту

```
EXPLAIN (ANALYZE, BUFFERS, TIMING off, COSTS off, SUMMARY off)
  SELECT * FROM points WHERE p <@ circle(point(0.5, 0.5), 0.01);
-- точка принадлежит кругу радиуса 0.01 с центром (0.5;0.5)
```



gist

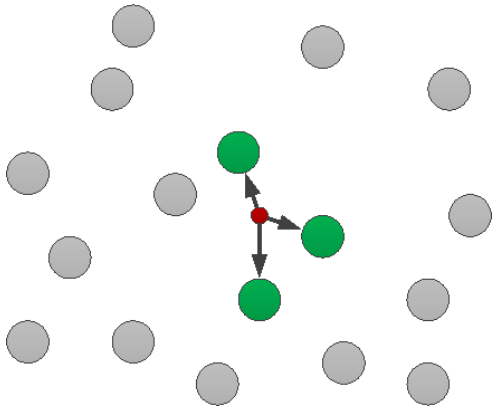
Принадлежность точки объекту

```
Index Only Scan using points_p_idx on points (actual rows=324 loops=1)
  Index Cond: (p <@ '<(0.5,0.5),0.01>'::circle)
  Heap Fetches: 0
  Buffers: shared hit=17
```

gist

kNN-search (ближайшие объекты)

```
EXPLAIN (ANALYZE, BUFFERS, TIMING off, COSTS off, SUMMARY off)
  SELECT * FROM points ORDER BY p <-> point(0.5, 0.5) LIMIT 5;
-- 5 ближайших к (0.5;0.5) точек
```



gist

kNN-search (ближайшие объекты)

```
Limit (actual rows=5 loops=1)
  Buffers: shared hit=13
  ->  Index Only Scan using points_p_idx on points (actual rows=5 loops=1)
        Order By: (p <-> '(0.5,0.5)'::point)
        Heap Fetches: 0
        Buffers: shared hit=13
```

gist

операторы **сетевых адресов**

(<<, <<=, >>, >>=, =, <>, <, <=, >, >=, &&)

тип сетевых адресов

inet

https://postgrespro.ru/docs/postgresql/15/functions-net

gist

Принадлежность IP-адреса подсети

```
CREATE TABLE ip AS -- миллион случайных IP
  SELECT
    '0.0.0.0'::inet + (random() * (1::bigint << 32))::bigint addr
  FROM generate_series(1, 1e6);

CREATE INDEX ON ip USING gist(addr);
ERROR:  data type inet has no default operator class for access method "gist"
HINT:  You must specify an operator class for the index or define a default operator class for the data type.
```

gist

Принадлежность IP-адреса подсети

```
CREATE INDEX ON ip USING gist(addr inet_ops);
EXPLAIN (ANALYZE, BUFFERS, TIMING off, COSTS off, SUMMARY off)
  SELECT * FROM ip WHERE addr << '192.168/16'::inet;
-- ищем все 192.168.xxx.yyy
```

```
Index Only Scan using ip_addr_idx on ip (actual rows=8 loops=1)
  Index Cond: (addr << '192.168.0.0/16'::inet)
  Heap Fetches: 0
  Buffers: shared hit=4
```

gist

операторы **диапазонов**

(=, &&, @>, <@, <<, >>, &<, &>, -|-)

типы числовых/хронологических диапазонов

int4range, int8range, numrange

daterange, tsrange, tstzrange

https://postgrespro.ru/docs/postgresql/15/functions-range

gist

Работа с отрезками

https://habr.com/ru/companies/tensor/articles/526624/

```
CREATE TABLE ranges AS
SELECT
  (random() * 1e2)::integer segment   -- 100 отделов
, (random() * 1e3)::integer employee  -- 1000 сотрудников
, dtb
, dtb + (random() * 1e2)::integer dte -- 100000 интервалов
FROM
  (SELECT now()::date - (random() * 1e3)::integer dtb FROM generate_series(1, 1e5)) T;
```

gist

Работа с отрезками

кто был в отпуске в новогоднюю ночь?

```
EXPLAIN (ANALYZE, BUFFERS, TIMING off, COSTS off, SUMMARY off)
  SELECT employee FROM ranges WHERE '2023-01-01'::date BETWEEN dtb AND dte;
  -- dtb <= const AND dte >= const
```

```
CREATE INDEX ON ranges(dtb, dte);
```

gist

Работа с отрезками

```
Index Scan using ranges_dtb_dte_idx on ranges (actual rows=5088 loops=1)
  Index Cond: ((dtb <= '2023-01-01'::date) AND (dte >= '2023-01-01'::date))
  Buffers: shared hit=5251
```

gist

Работа с отрезками

```
CREATE INDEX ON ranges USING gist(daterange(dtb, dte, '[]'));
```

```
EXPLAIN (ANALYZE, BUFFERS, TIMING off, COSTS off, SUMMARY off)
  SELECT employee FROM ranges
  WHERE daterange(dtb, dte, '[]') @> '2023-01-01'::date;
  -- принадлежность значения диапазону
```

gist

Работа с отрезками

```
Bitmap Heap Scan on ranges (actual rows=5088 loops=1)
  Recheck Cond: (daterange(dtb, dte, '[]'::text) @> '2023-01-01'::date)
  Heap Blocks: exact=541
  Buffers: shared hit=638 -- в 8 раз меньше!
  ->  Bitmap Index Scan on ranges_daterange_idx (actual rows=5088 loops=1)
        Index Cond: (daterange(dtb, dte, '[]'::text) @> '2023-01-01'::date)
        Buffers: shared hit=97
```

gist

Работа с отрезками

кто был в отпуске в новогоднюю ночь в конкретном отделе?

```
EXPLAIN (ANALYZE, BUFFERS, TIMING off, COSTS off, SUMMARY off)
  SELECT employee FROM ranges
  WHERE daterange(dtb, dte, '[]') @> '2023-01-01'::date AND segment = 1;
```

gist

Работа с отрезками

```
Bitmap Heap Scan on ranges (actual rows=62 loops=1)
  Recheck Cond: (daterange(dtb, dte, '[]'::text) @> '2023-01-01'::date)
  Filter: (segment = 1)
  Rows Removed by Filter: 5026
  Heap Blocks: exact=541
  Buffers: shared hit=638
  ->  Bitmap Index Scan on ranges_daterange_idx (actual rows=5088 loops=1)
        Index Cond: (daterange(dtb, dte, '[]'::text) @> '2023-01-01'::date)
        Buffers: shared hit=97
```

gist

Работа с отрезками

скалярная сегментация

```
CREATE INDEX ON ranges USING gist(segment, daterange(dtb, dte, '[]'));
ERROR:  data type integer has no default operator class for access method "gist"
HINT:  You must specify an operator class for the index or define a default operator class for the data type.
```

https://postgrespro.ru/docs/postgresql/15/btree-gist

```
CREATE EXTENSION btree_gist;
CREATE INDEX ON ranges USING gist(segment, daterange(dtb, dte, '[]'));
```

gist

Работа с отрезками

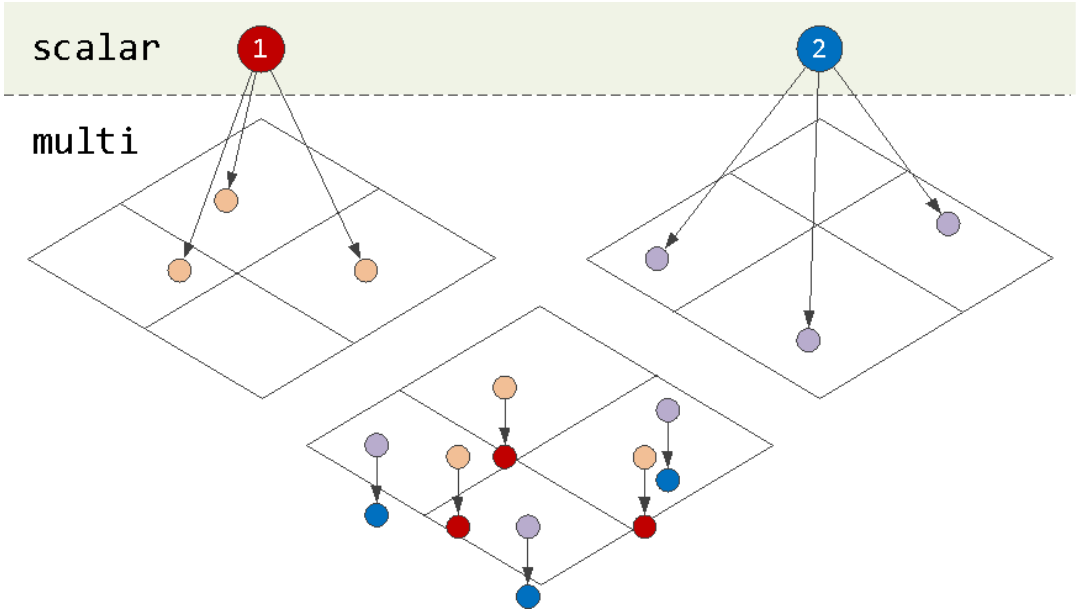
скалярная сегментация

```
Index Scan using ranges_segment_daterange_idx on ranges (actual rows=62 loops=1)
  Index Cond: ((segment = 1) AND (daterange(dtb, dte, '[]'::text) @> '2023-01-01'::date))
  Buffers: shared hit=68 -- в 10 раз меньше!
```

gist

Плохо: скаляр после «множества»

https://habr.com/ru/companies/tensor/articles/679834/



gist

Работа с отрезками

исключение пересечений

```
ALTER TABLE ranges
  ADD EXCLUDE USING gist(
    segment WITH =
  , daterange(dtb, dte, '[]'::text) WITH &&
  );
ERROR:  could not create exclusion constraint "ranges_segment_daterange_excl"
DETAIL:  Key (segment, daterange(dtb, dte, '[]'::text))=(62, [2022-11-30,2022-12-13)) conflicts
    with key (segment, daterange(dtb, dte, '[]'::text))=(62, [2022-09-23,2022-12-07)).
```

gist

операторы **полнотекстового поиска**

(<@, @>, @@)

FTS-типы

tsquery, tsvector

https://postgrespro.ru/docs/postgresql/15/functions-textsearch

gist

Полнотекстовый поиск

```
CREATE TABLE news AS
  SELECT
    id
  , string_agg(ch, '' ORDER BY i) title -- собираем «название»
  FROM (
    SELECT
      i
    , (random() * 1e4)::integer id -- 10000 разных «новостей»
    , CASE WHEN random() < 0.1 THEN ' ' ELSE chr(ascii('a') + (random() * 26)::integer) END ch -- [\s, a..z]
    FROM generate_series(1, 1e6) i
  ) T
  GROUP BY 1;
```

gist

Полнотекстовый поиск

```
CREATE INDEX ON news USING gist(to_tsvector('simple', title));

EXPLAIN (ANALYZE, BUFFERS, TIMING off, COSTS off, SUMMARY off)
  SELECT title FROM news
  WHERE to_tsvector('simple', title) @@ to_tsquery('ab:*');
  -- какое-то из «слов» названия начинается с 'ab'
```

gist

Полнотекстовый поиск

```
Index Scan using news_to_tsvector_idx on news (actual rows=78 loops=1)
  Index Cond: (to_tsvector('simple'::regconfig, title) @@ to_tsquery('ab:*'::text))
  Rows Removed by Index Recheck: 9923
  Buffers: shared hit=4054
```

* gist-преобразование деструктивно, поэтому не получили **Index Only Scan**

gist

Полнотекстовый поиск

```
CREATE INDEX ON news USING gist(to_tsvector('simple', title))
  INCLUDE(title);
```

```
Index Only Scan using news_to_tsvector_title_idx on news (actual rows=78 loops=1)
  Index Cond: ((to_tsvector('simple'::regconfig, title)) @@ to_tsquery('ab:*'::text))
  Rows Removed by Index Recheck: 9923
  Heap Fetches: 0
  Buffers: shared hit=250 -- в 16 раз меньше!
```

gist

расширения

btree_gist, pg_trgm, cube, hstore, intarray, ltree, seg

https://postgrespro.ru/docs/postgresql/15/gist-examples

gist

Индексный **не-префиксный** `LIKE`-поиск (+ RegExp)

триграммы спешат на помощь!

https://postgrespro.ru/docs/postgresql/15/pgtrgm

```
CREATE EXTENSION pg_trgm;
CREATE INDEX ON news USING gist(title gist_trgm_ops);
```

gist

Индексный не-префиксный LIKE-поиск (+ RegExp)

```
EXPLAIN (ANALYZE, BUFFERS, TIMING off, COSTS off, SUMMARY off)
  SELECT title FROM news
  WHERE title LIKE '%abc%'; -- title ~ 'abc'
  -- где-то в строке есть 'abc'
```

```
Bitmap Heap Scan on news (actual rows=15 loops=1)
  Recheck Cond: (title ~~ '%abc%'::text) -- (title ~ 'abc'::text)
  Heap Blocks: exact=12
  Buffers: shared hit=844
  ->  Bitmap Index Scan on news_title_idx (actual rows=15 loops=1)
        Index Cond: (title ~~ '%abc%'::text)
        Buffers: shared hit=832
```

gist

Индексный **не-префиксный** LIKE-поиск (+ RegExp)

```
CREATE INDEX ON news USING gist(title gist_trgm_ops)
  INCLUDE(title);
```

```
Index Only Scan using news_title_title1_idx on news (actual rows=15 loops=1)
  Index Cond: (title ~~ '%abc%'::text)
  Heap Fetches: 0
  Buffers: shared hit=1097 -- на 30% больше :(
```

* **Index Only Scan** не всегда оптимален

gist

Индексный поиск в массиве

```
CREATE TABLE refs AS
  SELECT
    id
  , array_agg(ref ORDER BY i) ref_ids -- собираем массив ссылок
  FROM (
    SELECT
      i
    , (random() * 1e4)::integer id  -- 10000 «статей»
    , (random() * 1e4)::integer ref
    FROM generate_series(1, 1e5) i  -- 100000 ссылок
  ) T
  GROUP BY 1;
```

gist

Индексный поиск в **массиве**

целочисленные массивы

https://postgrespro.ru/docs/postgresql/15/intarray

```
CREATE EXTENSION intarray;
CREATE INDEX ON refs USING gist(ref_ids); -- очень долго!
```

gist

Индексный поиск в массиве

```
EXPLAIN (ANALYZE, BUFFERS, TIMING off, COSTS off, SUMMARY off)
  SELECT * FROM refs WHERE ref_ids @> ARRAY[1::integer];
  -- в массиве ссылок есть id=1
```

```
Bitmap Heap Scan on refs (actual rows=9 loops=1)
  Recheck Cond: (ref_ids @> '{1}'::integer[])
  Heap Blocks: exact=8
  Buffers: shared hit=112
  ->  Bitmap Index Scan on refs_ref_ids_idx (actual rows=9 loops=1)
        Index Cond: (ref_ids @> '{1}'::integer[])
        Buffers: shared hit=104
```

gist

Индексный поиск в **массиве**

```
ids @> ARRAY[1, 2]::integer[];          -- присутствуют 1 И 2
ids && ARRAY[1, 2]::integer[];          -- присутствуют 1 ИЛИ 2
ids @@ '(1 & !2)|(!1 & 2)'::query_int -- 1 без 2 или 2 без 1
```

gist

Индексный поиск в массиве

```
CREATE INDEX ON refs USING gist(ref_ids gist__intbig_ops(siglen = 16));
```

```
Bitmap Heap Scan on refs (actual rows=21 loops=1)
  Recheck Cond: (ref_ids @@ '1 & !2 | !1 & 2'::query_int)
  Rows Removed by Index Recheck: 1429
  Heap Blocks: exact=121
  Buffers: shared hit=185
  ->  Bitmap Index Scan on refs_ref_ids_idx (actual rows=1450 loops=1)
        Index Cond: (ref_ids @@ '1 & !2 | !1 & 2'::query_int)
        Buffers: shared hit=64
```

* **siglen** – это опция класса операторов

gist

Индексный поиск в массиве

```
siglen | RRbIR | Buffers | relpages
     4    4651       180         59
     8    2677       181         58
    16    1429       185         66 -- default
    32     741       201         83
    64    1492       185        124
```

gist

ОПЦИИ

```
CREATE INDEX ... WITH (
  buffering = 'auto'
);
```

https://postgrespro.ru/docs/postgresql/15/sql-createindex#INDEX-RELOPTION-BUFFERING

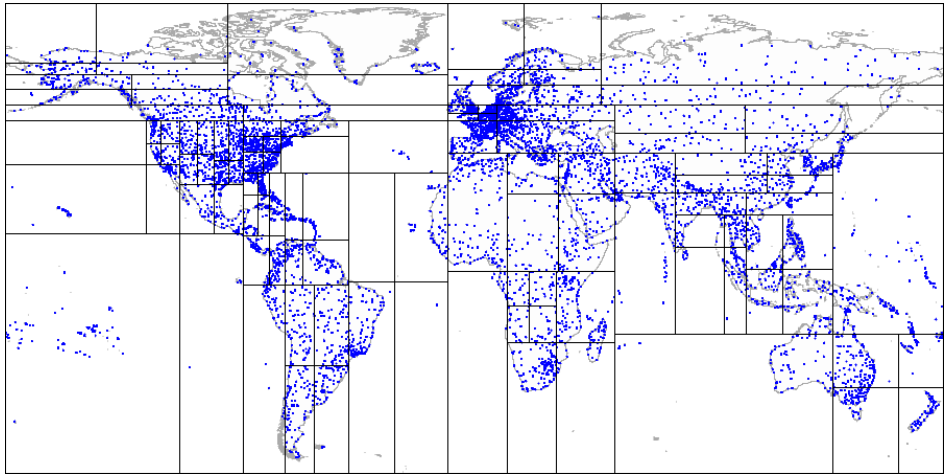
https://postgrespro.ru/docs/postgresql/15/gist-implementation#GIST-BUFFERING-BUILD

spgist

Space-Partitioned GiST

GiST для разбиения пространства

https://postgrespro.ru/docs/postgresql/15/spgist



spgist

все аналогично **gist**

оптимизации «под геометрию», включая kD-tree

но только одно поле (как у **hash**)

\+ операторы **линейного порядка** для **text_ops**

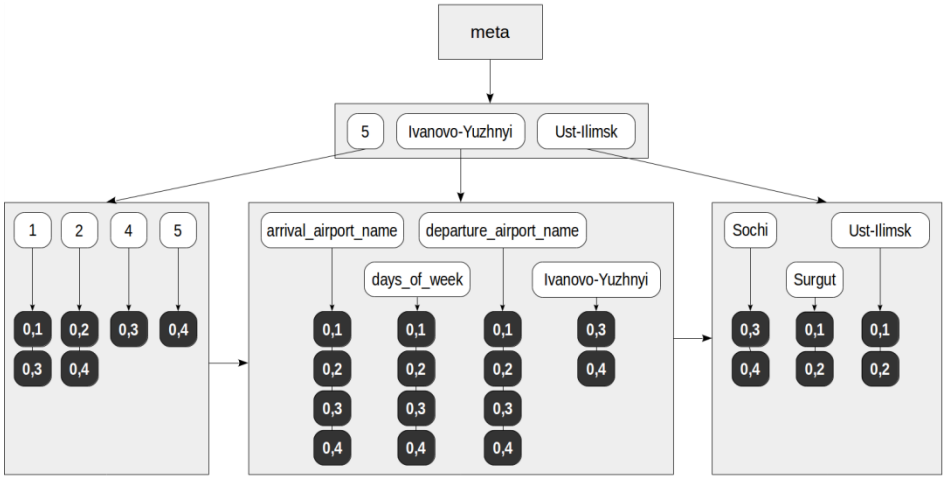
gin

Generalized Inverted Index

обобщенный инвертированный индекс

поиск составных объектов по элементу(ам)

https://postgrespro.ru/docs/postgresql/15/gin



gin

операторы **принадлежности массиву**

(`<@`, `@>`, `=`, `&&`)

любые массивы

`anyarray`

https://postgrespro.ru/docs/postgresql/15/functions-array

gin

Индексный поиск в массиве

```
CREATE INDEX ON refs USING gist(ref_ids gist__intbig_ops);
```

```
Bitmap Heap Scan on refs (actual rows=9 loops=1)
  Recheck Cond: (ref_ids @> '{1}'::integer[])
  Rows Removed by Index Recheck: 39
  Heap Blocks: exact=38
  Buffers: shared hit=98
  ->  Bitmap Index Scan on refs_ref_ids_idx (actual rows=48 loops=1)
        Index Cond: (ref_ids @> '{1}'::integer[])
        Buffers: shared hit=60
```

gin

Индексный поиск в массиве

```
CREATE INDEX ON refs USING gin(ref_ids gin__int_ops);
```

```
Bitmap Heap Scan on refs (actual rows=9 loops=1)
  Recheck Cond: (ref_ids @> '{1}'::integer[])
  Heap Blocks: exact=8
  Buffers: shared hit=11 -- в 9 раз меньше!
  ->  Bitmap Index Scan on refs_ref_ids_idx (actual rows=9 loops=1)
        Index Cond: (ref_ids @> '{1}'::integer[])
        Buffers: shared hit=3
```

gin

операторы **поиска JSON-ключей**

(`<@`, `@?`, `@@`, `?`, `?|`, `?&`)

«двоичные» JSON-объекты

`jsonb`, `jsonb_path`

https://postgrespro.ru/docs/postgresql/15/functions-json

gin

Индексный поиск в JSONb

```
CREATE INDEX ON refs USING gist(ref_ids gist__intbig_ops);
EXPLAIN (ANALYZE, BUFFERS, TIMING off, COSTS off, SUMMARY off)
  SELECT * FROM refs WHERE ref_ids @@ '1 & !2 | !1 & 2';
```

```
Bitmap Heap Scan on refs (actual rows=21 loops=1)
  Recheck Cond: (ref_ids @@ '1 & !2 | !1 & 2'::query_int)
  Rows Removed by Index Recheck: 75
  Heap Blocks: exact=67
  Buffers: shared hit=168
  ->  Bitmap Index Scan on refs_ref_ids_idx (actual rows=96 loops=1)
        Index Cond: (ref_ids @@ '1 & !2 | !1 & 2'::query_int)
        Buffers: shared hit=101
```

gin

Индексный поиск в JSONb

```
CREATE TABLE refs_jsonb AS
  SELECT to_jsonb(T) obj FROM refs T;

EXPLAIN (ANALYZE, BUFFERS, TIMING off, COSTS off, SUMMARY off)
  SELECT * FROM refs_jsonb WHERE obj @@ 'strict $.ref_ids ? (@[*] == 1 &&
!(@[*] == 2) || !(@[*] == 1) && @[*] == 2)';
```

https://postgrespro.ru/docs/postgresql/15/functions-json#FUNCTIONS-SQLJSON-PATH

gin

Индексный поиск в JSONb

```
CREATE INDEX ON refs_jsonb USING gin(obj jsonb_path_ops);
```

```
Bitmap Heap Scan on refs_jsonb (actual rows=21 loops=1)
  Recheck Cond: (obj @? 'strict $."ref_ids"?(@[*] == 1 && !(@[*] == 2) || !(@[*] == 1) && @[*] == 2)'::jsonpath)
  Heap Blocks: exact=20
  Buffers: shared hit=25 -- в 7 раз меньше!
  ->  Bitmap Index Scan on refs_jsonb_obj_idx (actual rows=21 loops=1)
        Index Cond: (obj @? 'strict $."ref_ids"?(@[*] == 1 && !(@[*] == 2) || !(@[*] == 1) && @[*] == 2)'::jsonpath)
        Buffers: shared hit=5
```

gin

операторы **полнотекстового поиска**

(@@)

FTS-типы

`tsquery`, `tsvector`

https://postgrespro.ru/docs/postgresql/15/functions-textsearch

gin

расширения

btree_gin, **pg_trgm**, **hstore**, **intarray**

https://postgrespro.ru/docs/postgresql/15/gin-examples

gin

ОПЦИИ

```
CREATE INDEX ... WITH (
  fastupdate = 'on'
, gin_pending_list_limit = '4MB'
);
```

https://postgrespro.ru/docs/postgresql/15/sql-createindex#INDEX-RELOPTION-FASTUPDATE

https://postgrespro.ru/docs/postgresql/15/sql-createindex#INDEX-RELOPTION-GIN-PENDING-LIST-LIMIT

https://postgrespro.ru/docs/postgresql/15/gin-implementation

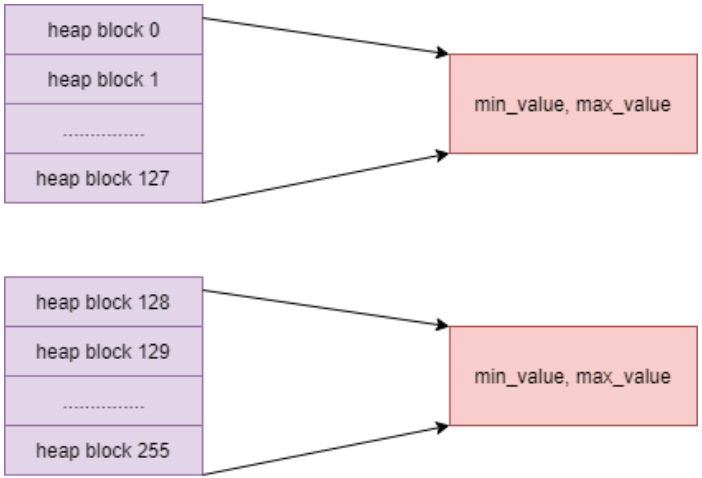
brin

Block Range Index

индекс зон (несколько подряд идущих) страниц данных

храним только min/max или bloom-маску

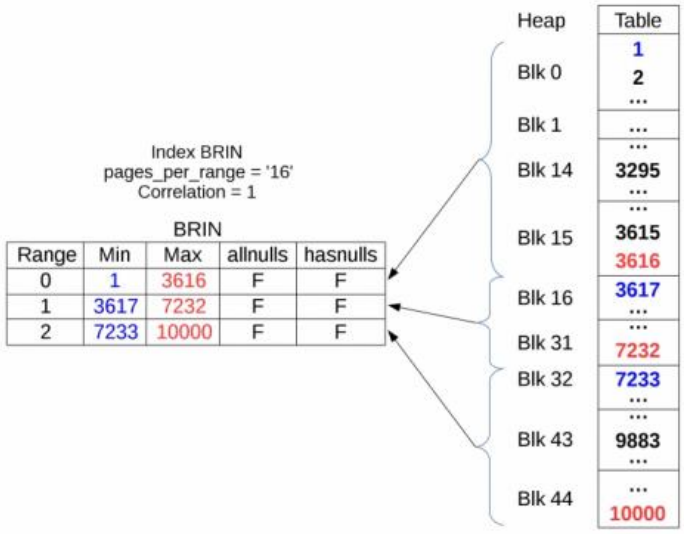
https://postgrespro.ru/docs/postgresql/15/brin



brin

для minmax – операторы **линейного порядка**

все аналогично **btree**



brin

Большие блоки возрастающих последовательностей

логи «за сутки» и т.п.

```
CREATE TABLE logs AS
  SELECT
    '2023-01-01 00:00:00'::timestamp +
      (i::double precision - 0.5 + random()) * '1 sec'::interval ts
  , md5(i::text)::uuid id
  FROM generate_series(1, 1e6) i;
```

brin

Большие блоки возрастающих последовательностей

```
EXPLAIN (ANALYZE, BUFFERS, TIMING off, COSTS off, SUMMARY off)
  SELECT * FROM logs WHERE
    ts >= '2023-01-02' AND ts < '2023-01-03'; -- вторые сутки
```

brin

Большие блоки возрастающих последовательностей

```
CREATE INDEX logs_btree ON logs USING btree(ts);
```

```
Index Scan using logs_btree on logs (actual rows=86400 loops=1)
  Index Cond: ((ts >= '2023-01-02 00:00:00'::timestamp without time zone)
               AND (ts < '2023-01-03 00:00:00'::timestamp without time zone))
  Buffers: shared hit=790
```

brin

Большие блоки возрастающих последовательностей

```
CREATE INDEX logs_brin ON logs USING brin(ts);
```

```
Bitmap Heap Scan on logs (actual rows=86400 loops=1)
  Recheck Cond: ((ts >= '2023-01-02 00:00:00'::timestamp without time zone)
              AND (ts < '2023-01-03 00:00:00'::timestamp without time zone))
  Rows Removed by Index Recheck: 14080
  Heap Blocks: lossy=640
  Buffers: shared hit=642 -- на 20% меньше!
  ->  Bitmap Index Scan on logs_brin (actual rows=6400 loops=1)
        Index Cond: ((ts >= '2023-01-02 00:00:00'::timestamp without time zone)
                  AND (ts < '2023-01-03 00:00:00'::timestamp without time zone))
        Buffers: shared hit=2
```

brin

Большие блоки возрастающих последовательностей

для небольших блоков – неэффективно

```
EXPLAIN (ANALYZE, BUFFERS, TIMING off, COSTS off, SUMMARY off)
  SELECT * FROM logs WHERE
    ts >= '2023-01-02 00:00:00' AND ts < '2023-01-02 01:00:00'; -- 1 час
```

brin

Большие блоки возрастающих последовательностей

для небольших блоков – неэффективно

```
Bitmap Heap Scan on logs (actual rows=3601 loops=1)
  Recheck Cond: ((ts >= '2023-01-02 00:00:00'::timestamp without time zone)
              AND (ts < '2023-01-02 01:00:00'::timestamp without time zone))
  Rows Removed by Index Recheck: 16495
  Heap Blocks: lossy=128
  Buffers: shared hit=130
  ->  Bitmap Index Scan on logs_brin (actual rows=1280 loops=1)
        Index Cond: ((ts >= '2023-01-02 00:00:00'::timestamp without time zone)
                  AND (ts < '2023-01-02 01:00:00'::timestamp without time zone))
        Buffers: shared hit=2
```

brin

Большие блоки возрастающих последовательностей

для небольших блоков – неэффективно (лучше **btree**)

```
Index Scan using logs_btree on logs (actual rows=3601 loops=1)
  Index Cond: ((ts >= '2023-01-02 00:00:00'::timestamp without time zone)
               AND (ts < '2023-01-02 01:00:00'::timestamp without time zone))
  Buffers: shared hit=36 -- в 4 раза меньше!
```

brin

Большие блоки возрастающих последовательностей

для небольших блоков – неэффективно (но **btree** больше)

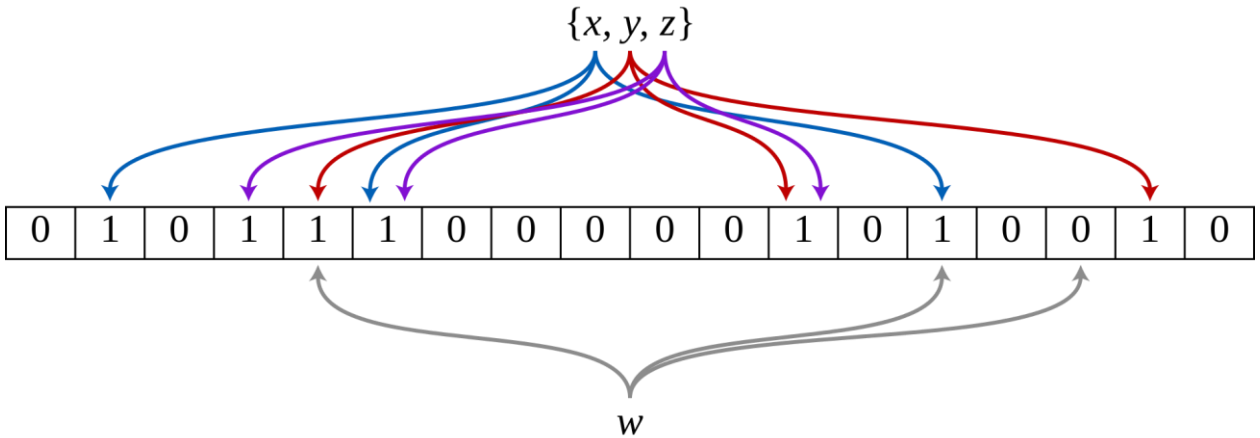
```
relname    | relpages
logs             6370
logs_btree       2745
logs_brin          28 -- в 100 раз меньше даже при PPR=1!
```

brin

для bloom – только оператор **равенства** (=)

типы аналогичны **btree**

https://ru.wikipedia.org/wiki/Фильтр_Блума



brin

ОПЦИИ

```
CREATE INDEX ... WITH (
  pages_per_range = 128
, autosummarize = 'off'
);
```

https://postgrespro.ru/docs/postgresql/15/sql-createindex#INDEX-RELOPTION-PAGES-PER-RANGE

https://postgrespro.ru/docs/postgresql/15/sql-createindex#INDEX-RELOPTION-AUTOSUMMARIZE

```
VACUUM / autovacuum
brin_summarize_new_values(regclass)     -- обновление новых зон
brin_summarize_range(regclass, bigint) -- пересчет зоны страницы
```

https://postgrespro.ru/docs/postgresql/15/brin-intro#BRIN-OPERATION

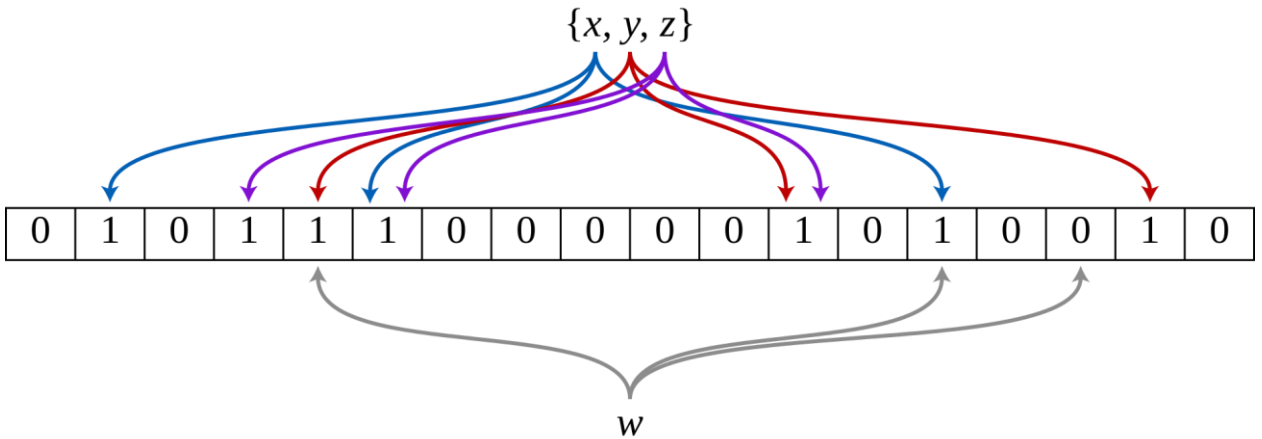
bloom

фильтр Блума

индекс зон (несколько подряд идущих) страниц данных

храним только min/max или bloom-маску

https://postgrespro.ru/docs/postgresql/15/bloom



bloom

только оператор **равенства** (=)

типы аналогичны **btree**

любая комбинация полей в условии

мечта ленивого разработчика

bloom

ОПЦИИ

Любая комбинация полей

```
CREATE EXTENSION bloom;
CREATE INDEX ... WITH (
  length = 80 -- бит на всю маску
, col1 = 2    -- бит на маску поля
, ...
, colN = 2
);
```

https://postgrespro.ru/docs/postgresql/15/bloom#id-1.11.7.16.7

bloom

Любая комбинация полей

```
CREATE TABLE mix AS
  SELECT
     (random() * 1e3)::integer A
   , (random() * 1e3)::integer B
   , (random() * 1e6)::integer C
   FROM generate_series(1, 1e6);
```

bloom

Любая комбинация полей (не для **btree**)

```
CREATE INDEX mix_btree ON mix USING btree(A, B, C);
```

```
EXPLAIN (ANALYZE, BUFFERS, TIMING off, COSTS off, SUMMARY off)
  SELECT * FROM mix WHERE C = 1;
```

```
Index Only Scan using mix_btree on mix (actual rows=1 loops=1)
  Index Cond: (c = 1)
  Heap Fetches: 0
  Buffers: shared hit=3835
```

bloom

Любая комбинация полей (но для **bloom**)

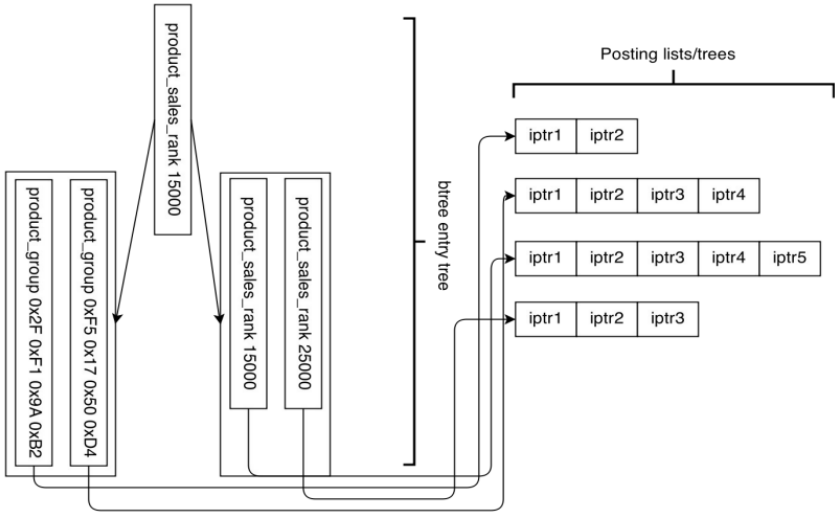
```
CREATE EXTENSION bloom;
CREATE INDEX mix_bloom ON mix USING bloom(A, B, C)
  WITH(col1 = 5, col2 = 5, col3 = 10);
```

```
Bitmap Heap Scan on mix (actual rows=1 loops=1)
  Recheck Cond: (c = 1)
  Heap Blocks: exact=1
  Buffers: shared hit=1962 -- в 2 раза меньше!
  ->  Bitmap Index Scan on mix_bloom (actual rows=1 loops=1)
        Index Cond: (c = 1)
        Buffers: shared hit=1961
```

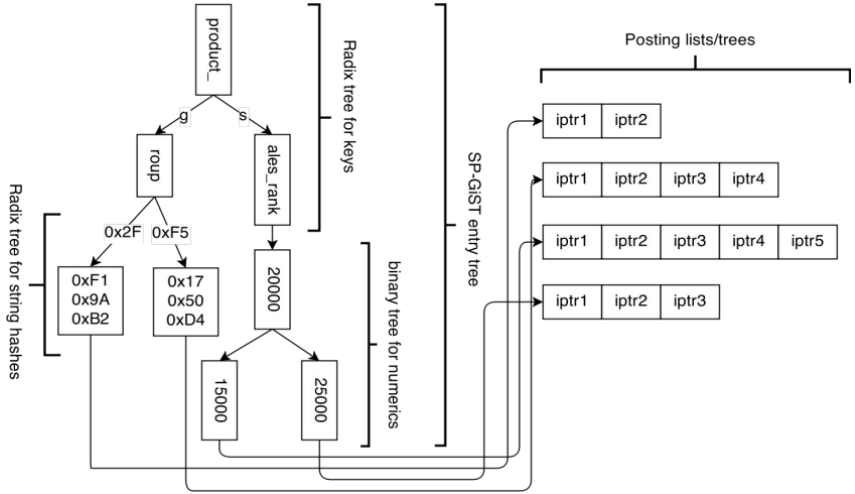
vodka VODKA Optimized Dendriform Keys Array

Proof-of-Concept («древоизация» **gin**-ключей)

https://www.pgcon.org/2014/schedule/attachments/318_pgcon-2014-vodka.pdf



gin keys



vodka keys

rum

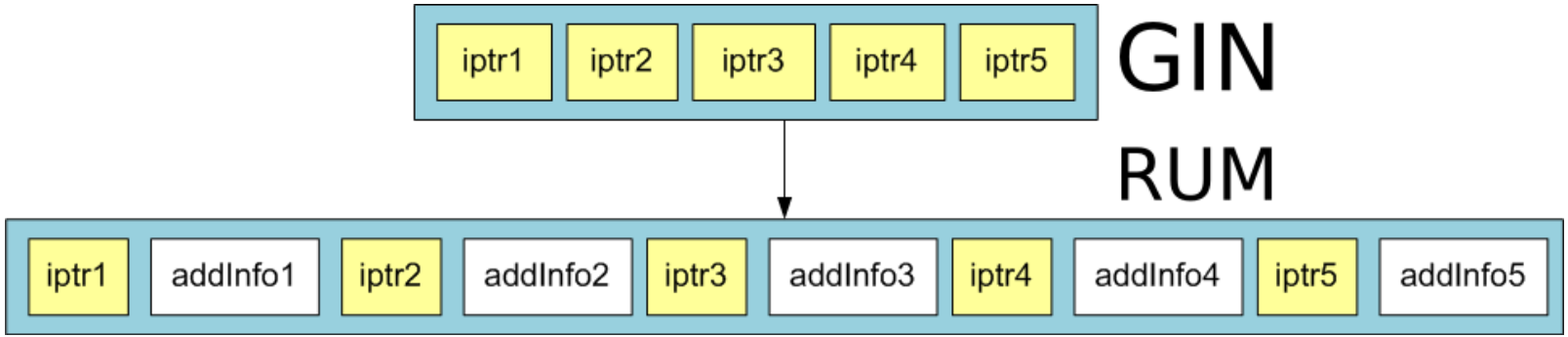
«gin с добавками»

вспомогательная информация в дереве индекса

ранкинг лексем, фразовый поиск, ордеринг **timestamp**

https://github.com/postgrespro/rum

https://pgconf.ru/2017/94101



rum

фразовый поиск

оператор «расстояния между словами» (<N>)

http://www.sai.msu.su/~megera/postgres/talks/pgopen-2016-rum.pdf

```
SELECT phraseto_tsquery('PostgreSQL can be extended by the user in many ways');
```

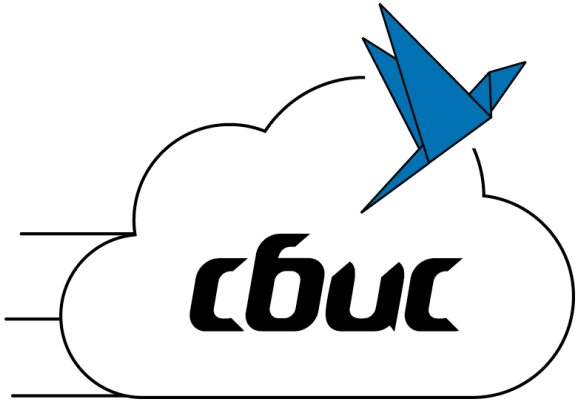
```
'postgresql' <3> 'extend' <3> 'user' <2> 'mani' <-> 'way'
```

Встроенные классы операторов

btree	https://postgrespro.ru/docs/postgresql/15/btree-behavior
gist	https://postgrespro.ru/docs/postgresql/15/gist-builtin-opclasses
spgist	https://postgrespro.ru/docs/postgresql/15/spgist-builtin-opclasses
gin	https://postgrespro.ru/docs/postgresql/15/gin-builtin-opclasses
brin	https://postgrespro.ru/docs/postgresql/15/brin-builtin-opclasses

Егор Рогов, серия «Индексы в PostgreSQL»

hash	https://habr.com/ru/companies/postgrespro/articles/328280/
btree	https://habr.com/ru/companies/postgrespro/articles/330544/
gist	https://habr.com/ru/companies/postgrespro/articles/333878/
spgist	https://habr.com/ru/companies/postgrespro/articles/337502/
gin	https://habr.com/ru/companies/postgrespro/articles/340978/
brin	https://habr.com/ru/companies/postgrespro/articles/346460/
bloom	https://habr.com/ru/companies/postgrespro/articles/349224/
rum	https://habr.com/ru/companies/postgrespro/articles/343488/



Спасибо за внимание!

Боровиков Кирилл

kilor@tensor.ru / https://n.sbis.ru/explain

sbis.ru / tensor.ru