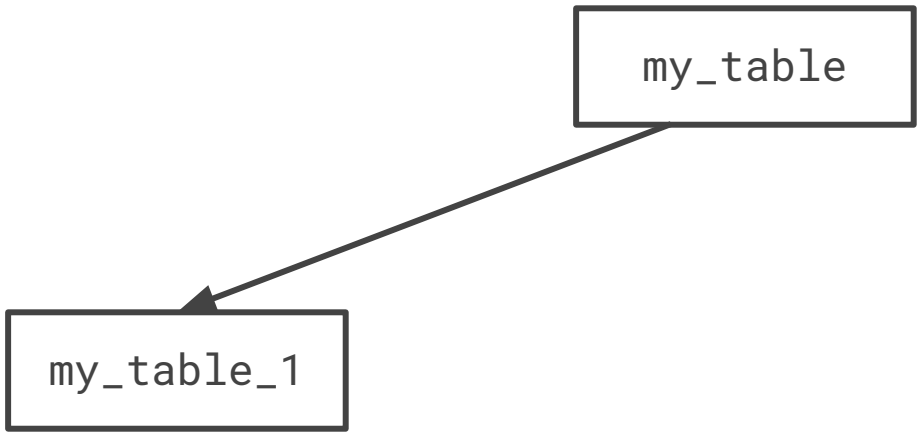
Секционирование при помощи pg_pathman

Дмитрий Иванов, Postgres Professional

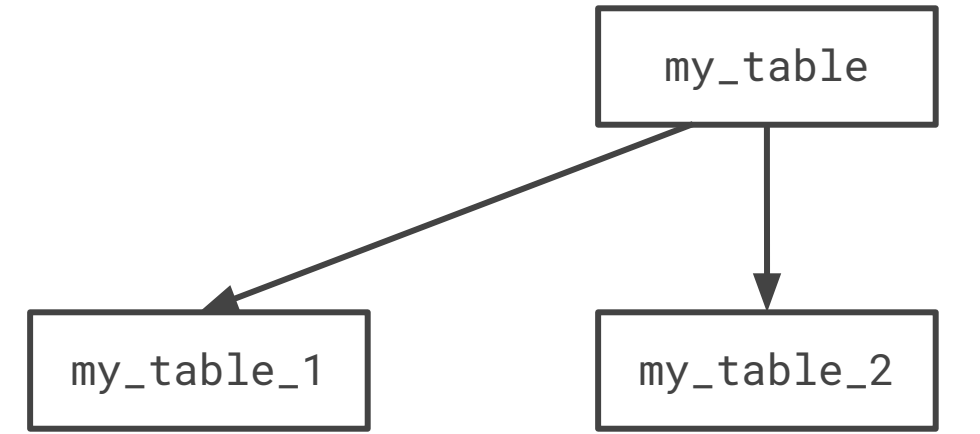
Проверенный метод секционирования

(PostgreSQL <= 9.6)


my_table

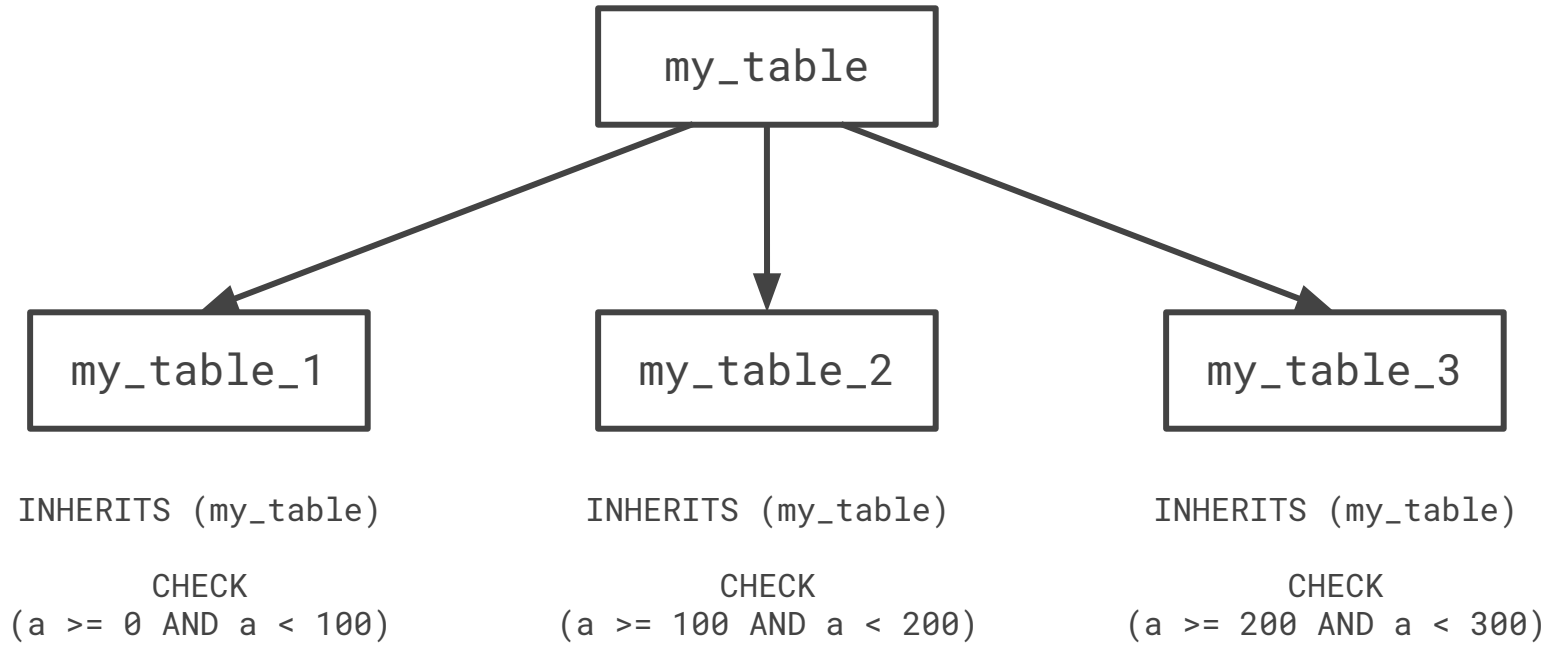


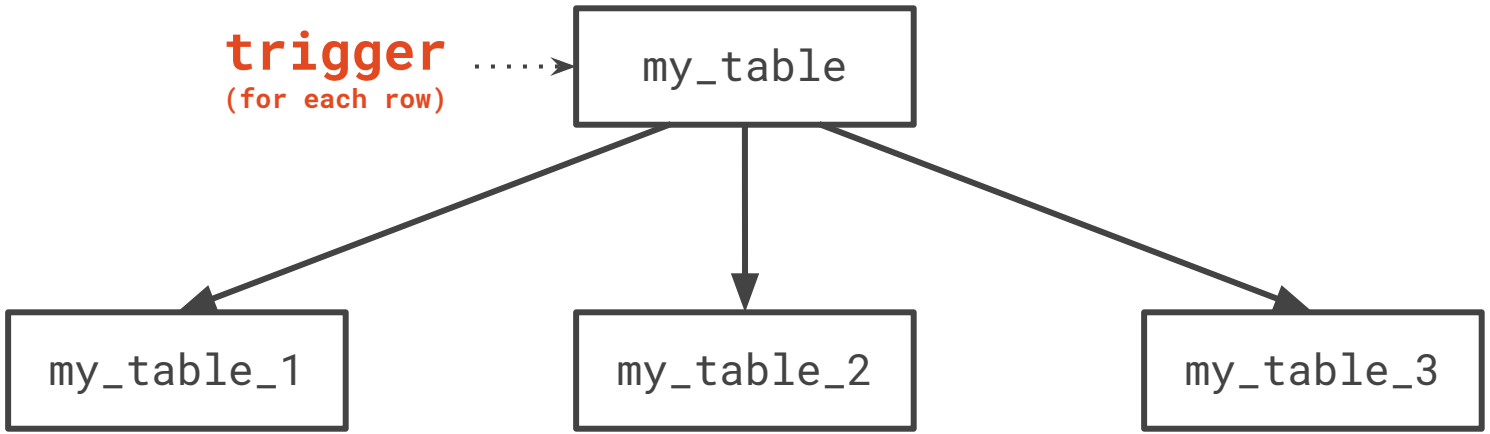

```
INHERITS (my_table)

        CHECK
(a >= 0 AND a < 100)
```

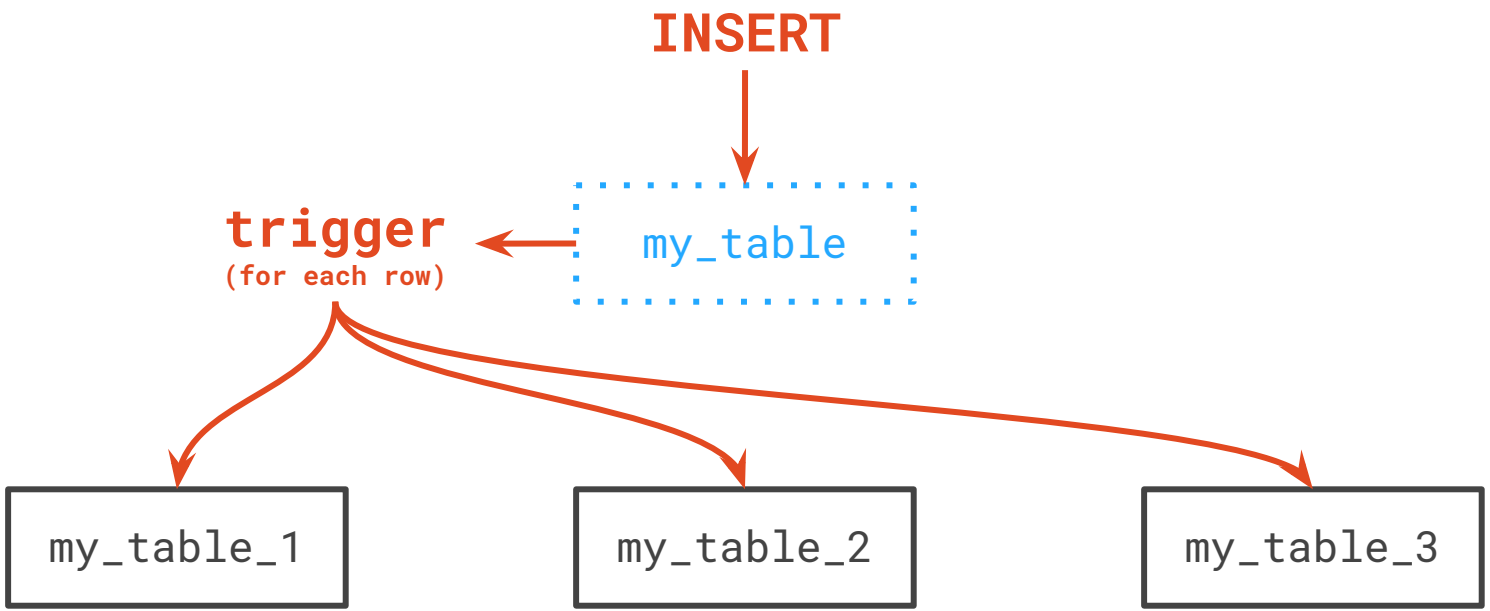



```
INHERITS (my_table)

CHECK
(a >= 0 AND a < 100)
```

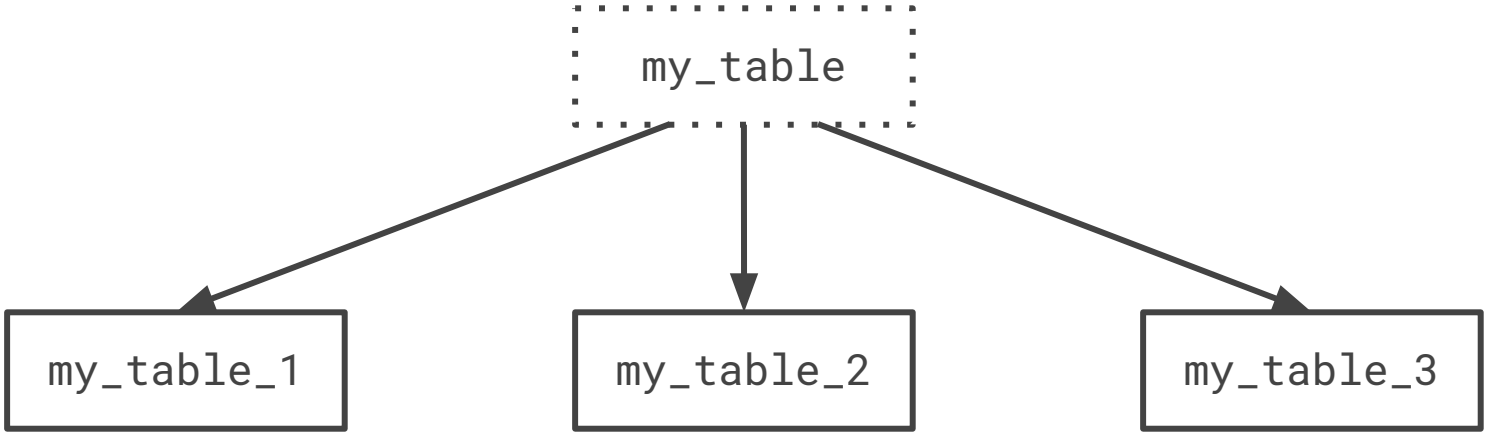
```
INHERITS (my_table)

CHECK
(a >= 100 AND a < 200)
```

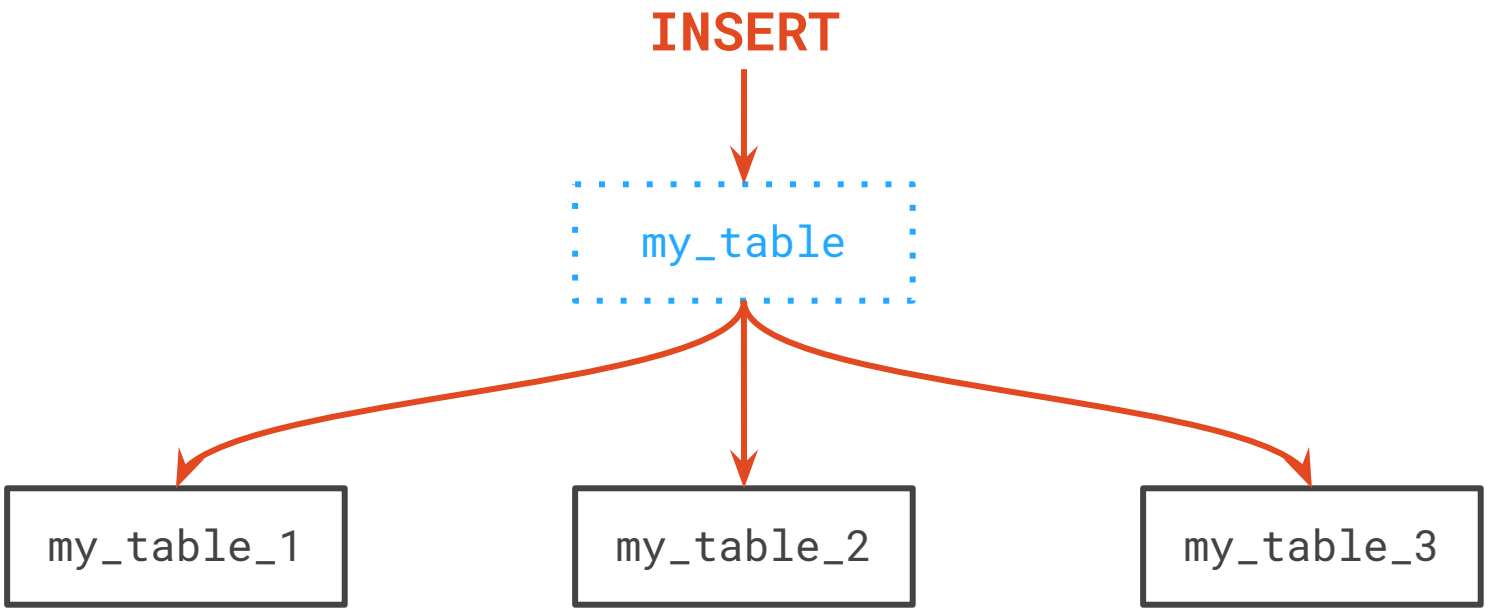

my_table
my_table_1
my_table_2
my_table_3
INHERITS (my_table)
CHECK
(a >= 0 AND a < 100)
INHERITS (my_table)
CHECK
(a >= 100 AND a < 200)
INHERITS (my_table)
CHECK
(a >= 200 AND a < 300)


trigger
(for each row)
my_table
my_table_1
my_table_2
my_table_3


INSERT
my_table
trigger
(for each row)
my_table_1
my_table_2
my_table_3

Возможности PostgreSQL 10


my_table
my_table_1
my_table_2
my_table_3


INSERT
my_table
my_table_1
my_table_2
my_table_3

```
create table test (id int not null, value float8)
partition by range (id);

create table test_1 partition of test
for values from  (1) to (10);

create table test_2 partition of test
for values from (10) to (20);

create table test_3 (like test);

alter table test attach partition test_3
for values from (20) to (30);
```

```
\d+ test
...
Partition key: RANGE (id)
Partitions: test_1 FOR VALUES FROM  (1) TO (10),
            test_2 FOR VALUES FROM (10) TO (20),
            test_3 FOR VALUES FROM (20) TO (30)
```

```
insert into test select generate_series(1, 30, 8), random()
returning tableoid::regclass as partition, *;

 partition | id |    value
-----------+----+-------------------
 test_1    |  1 | 0.461184393148869
 test_1    |  9 | 0.972871645819396
 test_2    | 17 | 0.318843736313283
 test_3    | 25 | 0.737302441615611
(4 rows)
```

```
alter table test detach partition test_2;
alter table test detach partition test_3;

insert into test_2 select * from test_3;
drop table test_3;

alter table test attach partition test_2
for values from (10) to (30);

\d+ test
...
Partition key: RANGE (id)
Partitions: test_1 FOR VALUES FROM  (1) TO (10),
            test_2 FOR VALUES FROM (10) TO (30)
```

```
insert into test values(100, 0);
ERROR:  no partition of relation "test" found for row

create index on test (value);
ERROR:  cannot create index on partitioned table "test"

insert into test values (1, 0) on conflict (id)
do update set value=1;
ERROR:  ON CONFLICT clause is not supported with partitioned
tables

update test set id=15 where id = 1;
ERROR:  new row for relation "test_1" violates partition
constraint
```

```
create table abc (data jsonb) partition by range ((data->>'key')::int8);
ERROR:  syntax error at or near "::"
LINE 1: ... abc (data jsonb) partition by range ((data->>'key')::int8);


create table abc (data jsonb) partition by range (((data->>'key')::int8));
CREATE TABLE
```

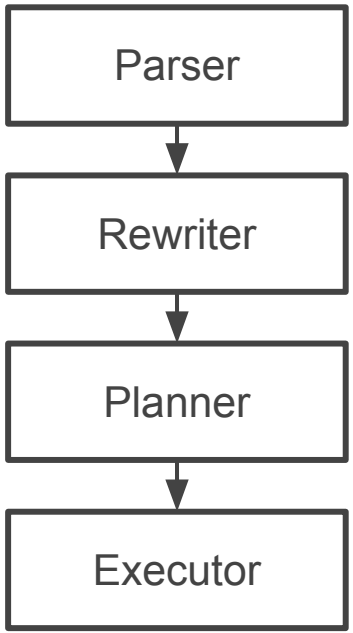
```
/* session #1 */
begin;

create table test_4 partition of test
for values from (30) to (40);

/* session #2 */
select * from test;

/* ps aux | grep postgres */
postgres: dmitry pathman [local] idle in transaction
postgres: dmitry pathman [local] SELECT waiting
```

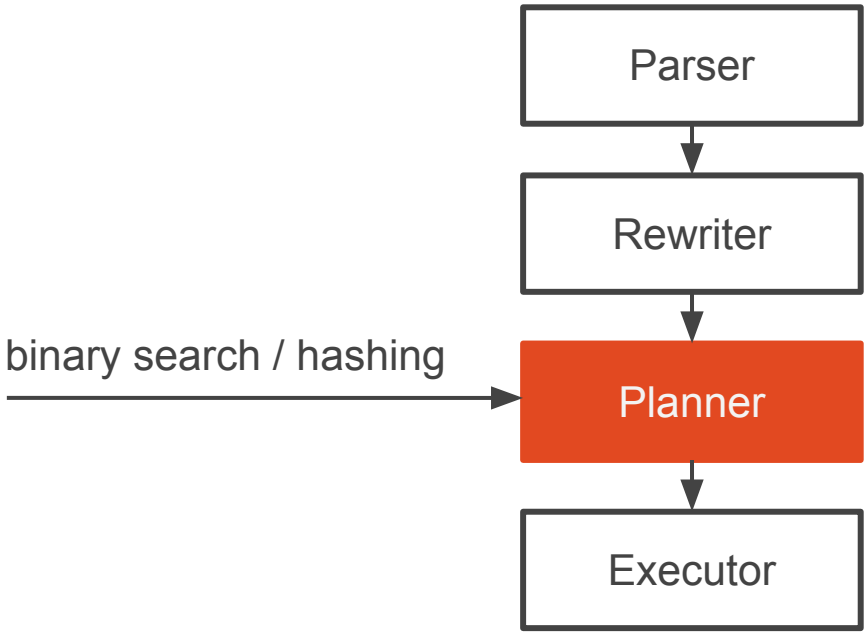
https://github.com/funbringer/YCSB

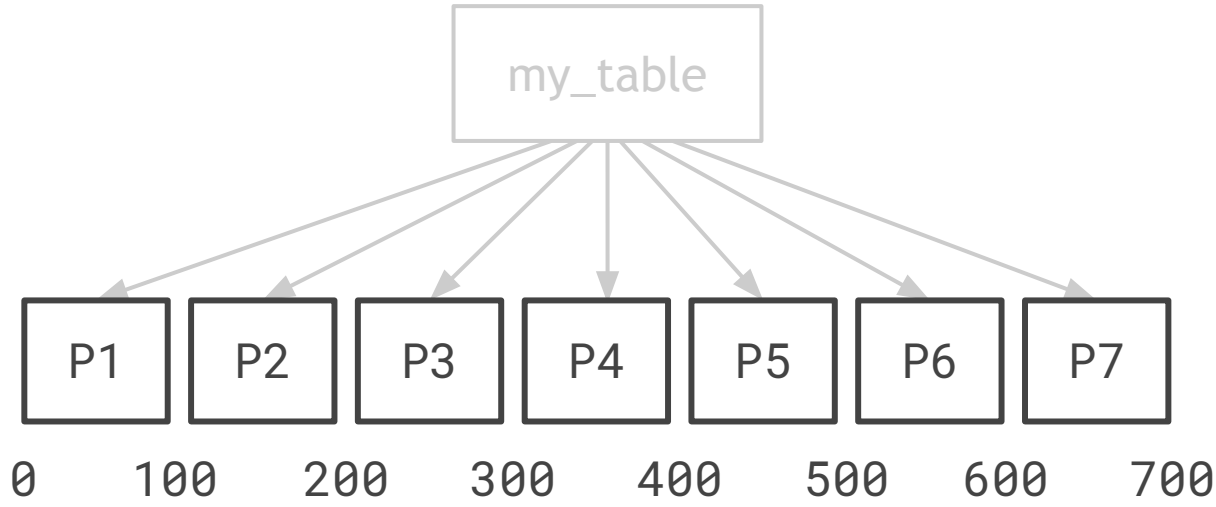
Прокачиваем секционирование

Возможности pg_pathman

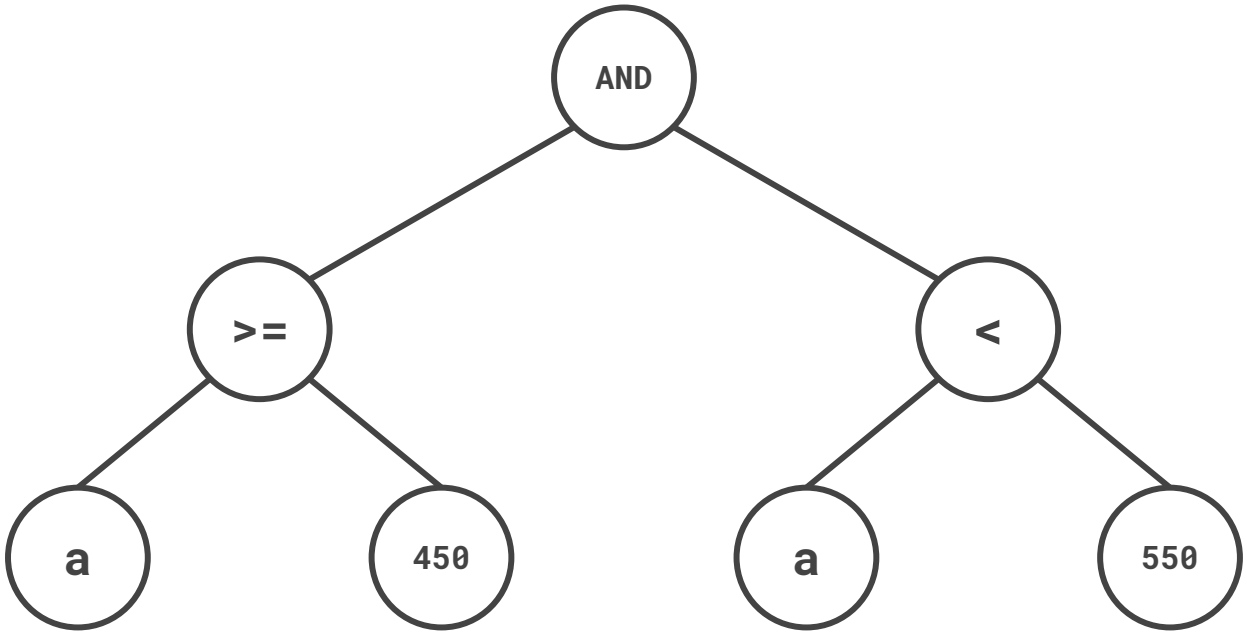
- HASH и RANGE секционирование
- Оптимизации планирования и исполнения
- PartitionFilter вместо триггеров (быстрый INSERT)
- Автоматическое создание секций при вставке
- Пользовательские колбеки на создание секций
- Поддержка FDW-партиций
- Неблокирующее секционирование
- **Удобный API**

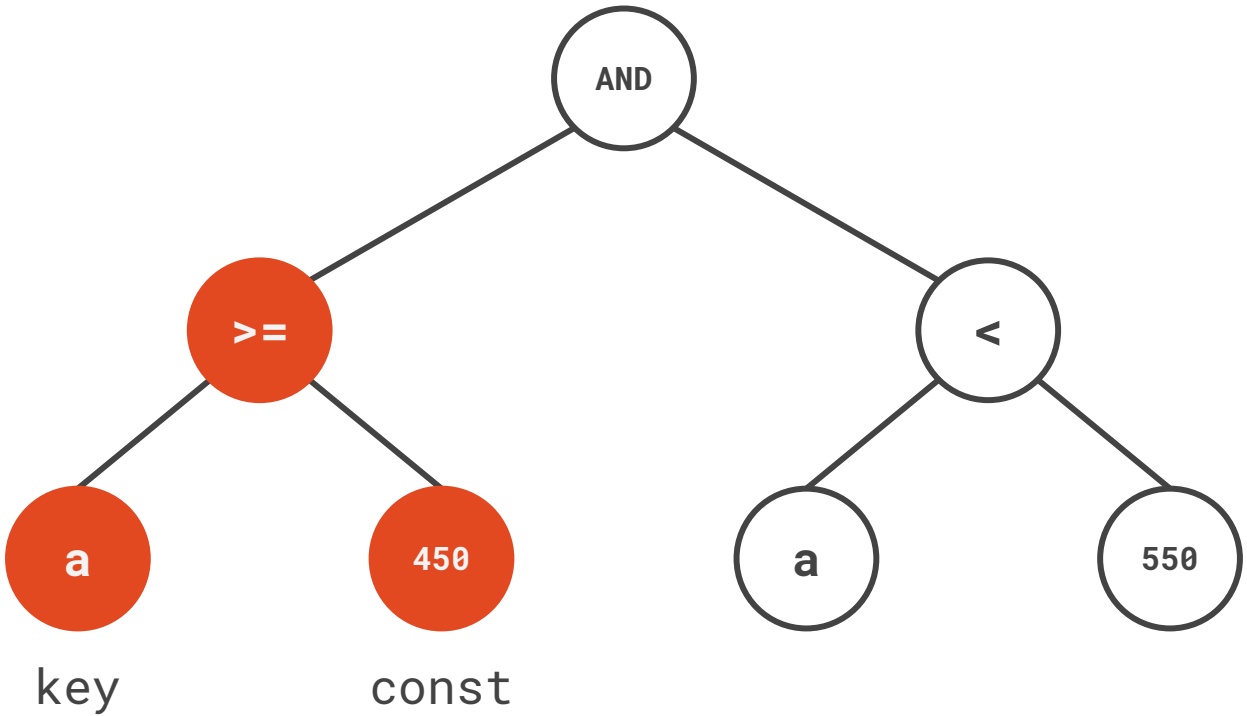

Parser
Rewriter
Planner
Executor


Parser
Rewriter
binary search / hashing
Planner
Executor

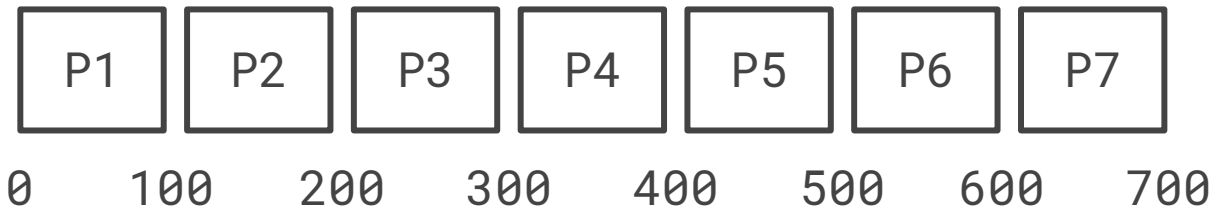

my_table
P1
P2
P3
P4
P5
P6
P7
0
100
200
300
400
500
600
700

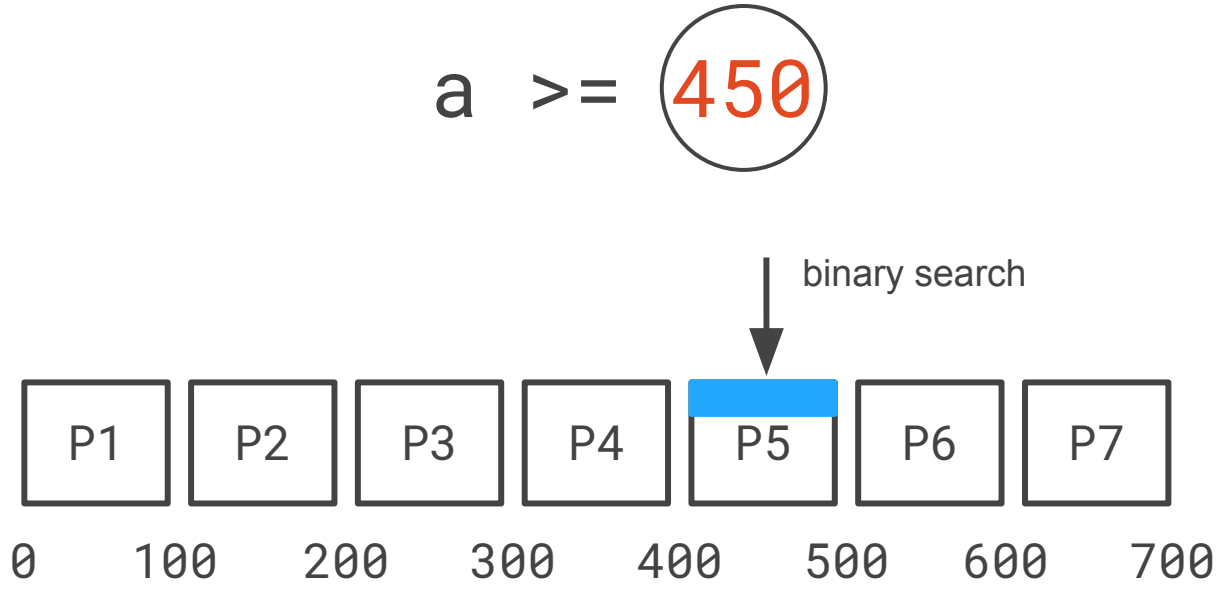
```
SELECT * FROM my_table
WHERE a >= 450 and a < 600
```


AND
>=
<
a
450
a
550

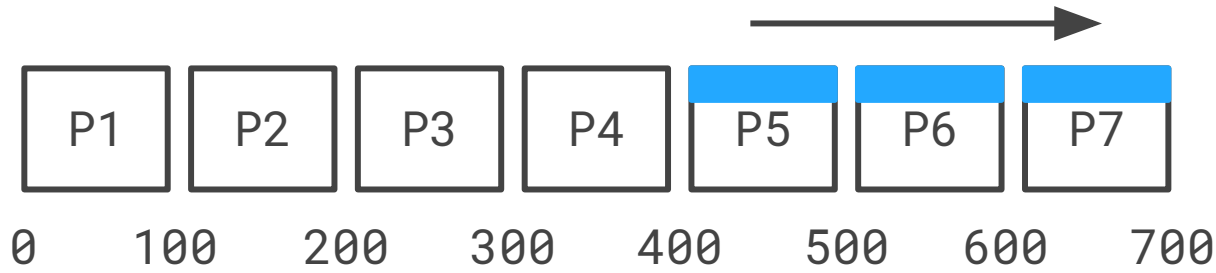

AND
>=
<
a
450
a
550
key
const

a >= 450

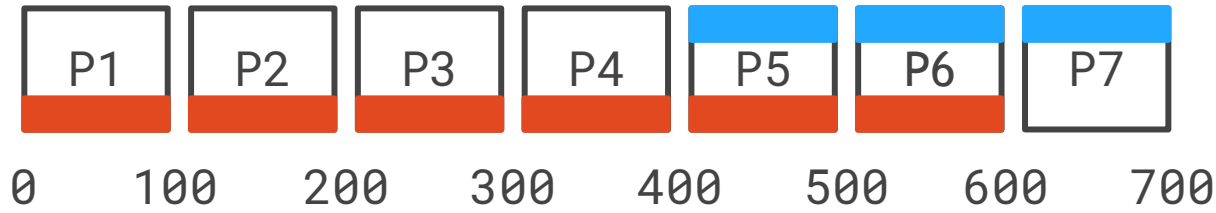



a >= 450
binary search
P1
P2
P3
P4
P5
P6
P7
0
100
200
300
400
500
600
700

a >= 450



```
a >= 450 and a < 550
```



```
a >= 450 and a < 550
```

```
explain (costs off)
select * from my_table where a >= 450 and a < 550;

           QUERY PLAN
--------------------------------
 Append
   ->  Seq Scan on my_table_5
         Filter: (a >= 450)
   ->  Seq Scan on my_table_6
         Filter: (a < 550)
(5 rows)
```

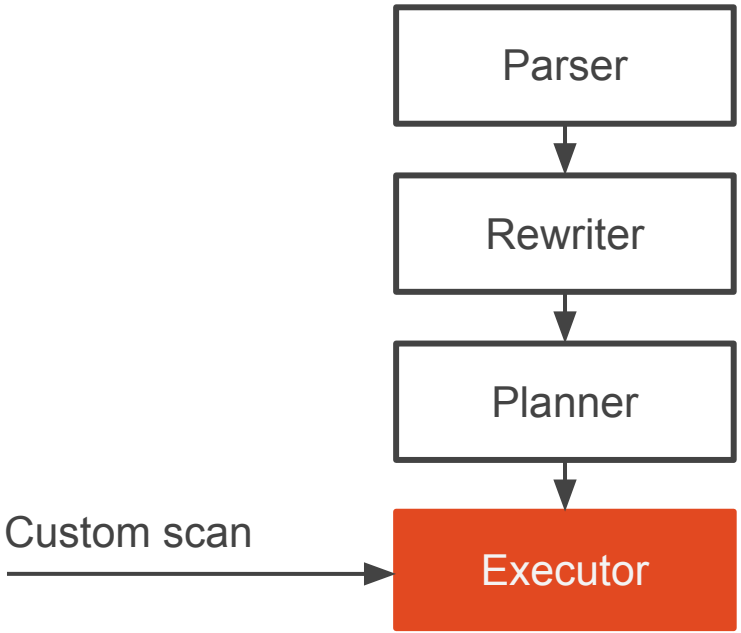
Оптимизация условий

```
SELECT * FROM my_table
WHERE a >= 450
```

```
                QUERY PLAN
-------------------------------------
 Append
   ->  Seq Scan on my_table_5
         Filter: (a >= 450)
   ->  Seq Scan on my_table_6
         Filter: (a >= 450)
   ->  Seq Scan on my_table_7
         Filter: (a >= 450)
(9 rows)
```

```
                QUERY PLAN
-------------------------------------
 Append
   ->  Seq Scan on my_table_5
         Filter: (a >= 450)
   ->  Seq Scan on my_table_6
         Filter: (a >= 450)
   ->  Seq Scan on my_table_7
         Filter: (a >= 450)
(9 rows)
```

```
                QUERY PLAN
-------------------------------------
 Append
   ->  Seq Scan on my_table_5
         Filter: (a >= 450)
   ->  Seq Scan on my_table_6
   ->  Seq Scan on my_table_7
(7 rows)
```


Parser
Rewriter
Planner
Custom scan
Executor

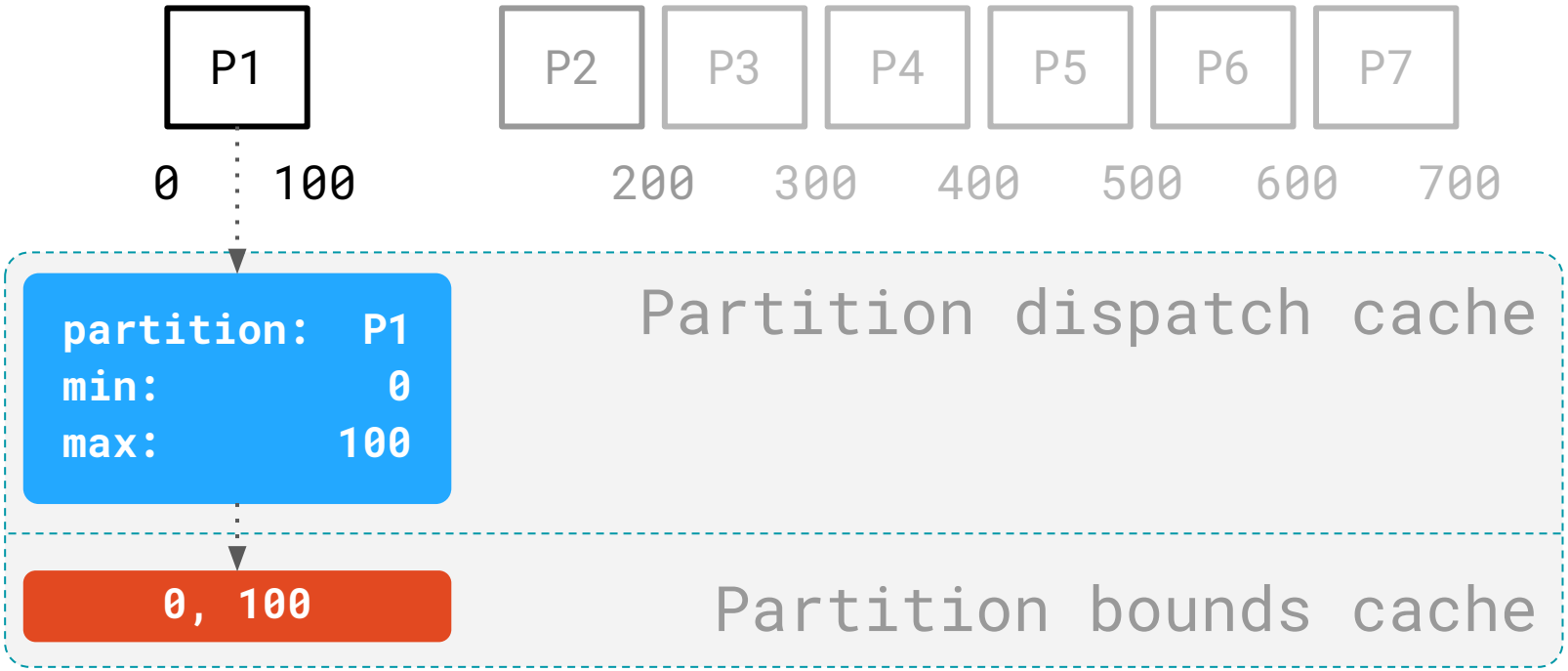
```
explain (analyze, costs off, timing off)
select * from abc join dummy using (val);

                        QUERY PLAN
-----------------------------------------------------------
 Nested Loop (actual rows=0 loops=1)
   Join Filter: (abc.val = dummy.val)
   ->  Append (actual rows=0 loops=1)
         ->  Seq Scan on abc (actual rows=0 loops=1)
         ->  Seq Scan on abc_1 (actual rows=0 loops=1)
         ->  Seq Scan on abc_2 (actual rows=0 loops=1)
          ....
         ->  Seq Scan on abc_99 (actual rows=0 loops=1)
         ->  Seq Scan on abc_100 (actual rows=0 loops=1)
   ->  Materialize (never executed)
         ->  Seq Scan on dummy (never executed)
 Planning time: 17.522 ms
 Execution time: 0.443 ms
(108 rows)
```

```
explain (analyze, costs off, timing off)
select * from abc join dummy using (val);

                      QUERY PLAN
------------------------------------------------------------
 Nested Loop (actual rows=0 loops=1)
   ->  Seq Scan on dummy (actual rows=5 loops=1)
   ->  Custom Scan (RuntimeAppend) (actual rows=0 loops=5)
         Prune by: (dummy.val = abc.val)
         ->  Seq Scan on abc_1 (actual rows=0 loops=4)
               Filter: (dummy.val = val)
         ->  Seq Scan on abc_2 (actual rows=0 loops=1)
               Filter: (dummy.val = val)
 Planning time: 6.259 ms
 Execution time: 0.535 ms
(10 rows)
```

```
prepare q(int, int, int) as
select * from abc where val in ($1, $2, $3);

explain (analyze, costs off, timing off) execute q(1, 200, 100);
                      QUERY PLAN
-------------------------------------------------------
 Append (actual rows=0 loops=1)
   ->  Seq Scan on abc_1 (actual rows=0 loops=1)
         Filter: (val = ANY ('{1,200,100}'::integer[]))
   ->  Seq Scan on abc_2 (actual rows=0 loops=1)
         Filter: (val = ANY ('{1,200,100}'::integer[]))
 Planning time: 0.478 ms
 Execution time: 0.053 ms
(7 rows)
```

```
prepare q(int, int, int) as
select * from abc where val in ($1, $2, $3);

explain (analyze, costs off, timing off) execute q(1, 200, 100);
                      QUERY PLAN
-------------------------------------------------------
 Custom Scan (RuntimeAppend) (actual rows=0 loops=1)
   Prune by: (abc.val = ANY (ARRAY[$1, $2, $3]))
   ->  Seq Scan on abc_1 (actual rows=0 loops=1)
         Filter: (val = ANY (ARRAY[$1, $2, $3]))
   ->  Seq Scan on abc_2 (actual rows=0 loops=1)
         Filter: (val = ANY (ARRAY[$1, $2, $3]))
 Planning time: 0.365 ms
 Execution time: 0.203 ms
(8 rows)
```

Накладные расходы pg_pathman


P1
P2
P3
P4
P5
P6
P7
0
100
200
300
400
500
600
700
partition: P1
min: 0
max: 100
Partition dispatch cache
0, 100
Partition bounds cache

```
explain
select * from partitioned join dummy
using (id);
```

```
...
Planning time: 15.527 ms
Execution time: 0.136 ms

...
Planning time: 4.194 ms
Execution time: 0.145 ms

...
Planning time: 5.450 ms
Execution time: 0.198 ms
```

```
create table test (id int8 not null);
select create_range_partitions('test', 'id', 1, 10, 1000);

select context,
       pg_size_pretty(used) as used,
       pg_size_pretty(size) as size,
       Entries
from pathman_cache_stats;

         context          |   used    |  size  | entries
--------------------------+-----------+--------+---------
 maintenance              | 120 bytes | 56 kB  |       0
 partition dispatch cache | 51 kB     | 52 kB  |       1
 partition parents cache  | 93 kB     | 120 kB |    1000
 partition bounds cache   | 133 kB    | 248 kB |    1000
(4 rows)
```

Примеры работы с API

- Как объединить несколько партиций?

```
select * from pathman_partition_list where parent = 'abc'::regclass and range_max::int <= 101;
 parent | partition | parttype | expr | range_min | range_max
--------+-----------+----------+------+-----------+-----------
 abc    | abc_1     |        2 | val  | 1         | 50
 abc    | abc_101   |        2 | val  | 50        | 101

select merge_range_partitions('abc_1', 'abc_2');

select * from pathman_partition_list where parent = 'abc'::regclass and range_max::int <= 101;
 parent | partition | parttype | expr | range_min | range_max
--------+-----------+----------+------+-----------+-----------
 abc    | abc_1     |        2 | val  | 1         | 101
```

- Как разделить одну партицию на несколько?

```
select * from pathman_partition_list where parent = 'abc'::regclass and range_max::int <= 101;
 parent | partition | parttype | expr | range_min | range_max
--------+-----------+----------+------+-----------+-----------
 abc    | abc_1     |      2   | val  | 1         | 101

select split_range_partition('abc_1', 50);

select * from pathman_partition_list where parent = 'abc'::regclass and range_max::int <= 101;
 parent | partition | parttype | expr | range_min | range_max
--------+-----------+----------+------+-----------+-----------
 abc    | abc_1     |      2   | val  | 1         | 50
 abc    | abc_101   |      2   | val  | 50        | 101
```

- Есть готовая таблица с данными, как сделать её партицией?

```
create table abc_attach (like abc);
insert into abc_attach values (-100), (-50);

select attach_range_partition('abc', 'abc_attach', -100, -10);

select * from pathman_partition_list where parent = 'abc'::regclass and range_min::int <= 0;
 parent | partition  | parttype | expr | range_min | range_max
--------+------------+----------+------+-----------+-----------
 abc    | abc_attach |        2 | val  | -100      | -10
```

- Как отключить партицию?

```
select * from pathman_partition_list where parent = 'abc'::regclass and range_min::int <= 0;
 parent | partition  | parttype | expr | range_min | range_max
--------+------------+----------+------+-----------+-----------
 abc    | abc_attach |        2 | val  | -100      | -10

select detach_range_partition('abc_attach');

select * from pathman_partition_list where parent = 'abc'::regclass and range_min::int <= 0;
 parent | partition | parttype | expr | range_min | range_max
--------+-----------+----------+------+-----------+-----------
```

Секционирование по выражению

```
create table test (data jsonb not null);

select create_hash_partitions(
    'test',
    '(data->>''key'')::int8',
    100);

insert into test
select format('{ "key": %s, "value": %s }',
              i, random())::jsonb
from generate_series(1, 1E5) as i;
```

```
select tableoid::regclass as partition, * from test
order by (data->>'key')::int8 asc
limit 5;

 partition |                    data
-----------+-----------------------------------------
 test_70   | {"key": 1, "value": 0.0304744695313275}
 test_26   | {"key": 2, "value": 0.547593057621 27}
 test_27   | {"key": 3, "value": 0.333599978126585}
 test_63   | {"key": 4, "value": 0.0428675869479775}
 test_88   | {"key": 5, "value": 0.22501541255042}
(5 rows)
```

```
explain (analyze, costs off, timing off)
select * from test
where (data->>'key')::int8 = 17;

                        QUERY PLAN
-----------------------------------------------------------
 Append (actual rows=1 loops=1)
   ->  Seq Scan on test_20 (actual rows=1 loops=1)
       Filter: (((data ->> 'key'::text))::bigint = 17)
       Rows Removed by Filter: 1000
 Planning time: 0.354 ms
 Execution time: 1.949 ms
```

```
set pg_pathman.enable = off;

explain (analyze, costs off, timing off)
select * from test
where (data->>'key')::int8 = 17;

                        QUERY PLAN
-----------------------------------------------------------
 Append (actual rows=1 loops=1)
   [ 100 partitions... ]
 Planning time: 6.867 ms
 Execution time: 65.540 ms
```

Серебряная пуля?

- Не все prepared statements одинаково полезны
- Нет пула соединений? Good luck with that! (привет, PHP)
- FOR EACH ROW триггеры не наследуются от родителя
- Не работают primary keys (кроме ключа секционирования)
- Не работает INSERT ON CONFLICT (в vanilla тоже)

- Нет модного синтаксиса (есть в Postgres Pro Enterprise!)
- Нет шардинга из коробки (развиваем pg_shardman)
- Не работают foreign keys
- (Пока что) нет LIST секционирования

Что мы делаем прямо сейчас?

- Многоуровневое секционирование
- Узел плана для UPDATE по ключу секционирования
- Базовая поддержка foreign keys (прототип)
- Шардинг с избыточностью (pg_shardman)
- Поддержка декларативного синтаксиса PostgreSQL 10
- Статистика по запросам (нагрузка на секции и тд)
- Создание новых индексов/триггеров на всех партициях

https://github.com/postgrespro/pg_pathman

Спасибо за внимание!