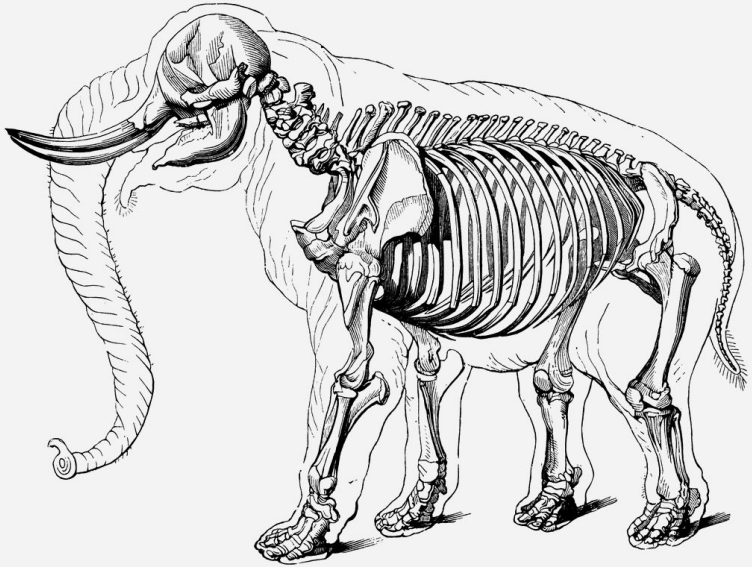
Ильдар Мусин
**Postgres Professional**

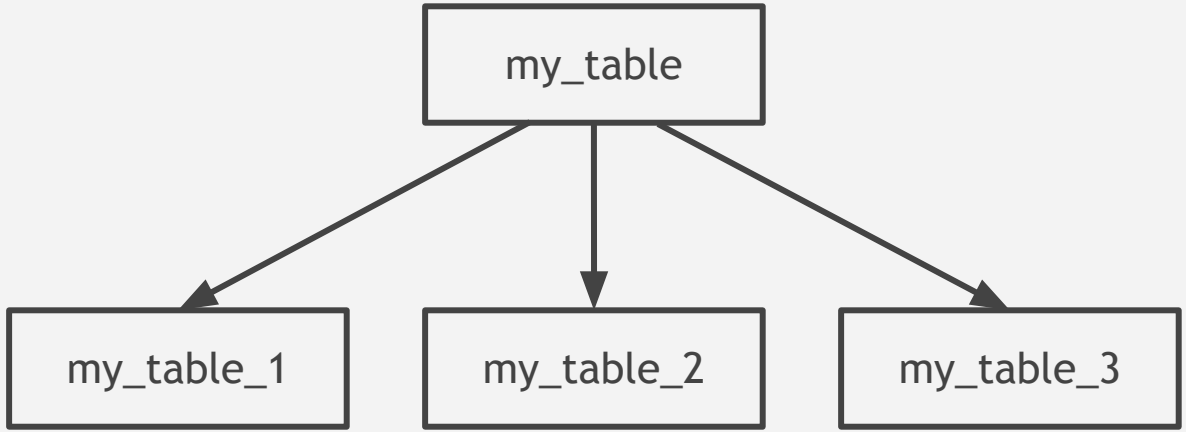
PROFESSIONAL Postgres

PGDAY'17 RUSSIA

КОНФЕРЕНЦИЯ ПО БАЗАМ ДАННЫХ



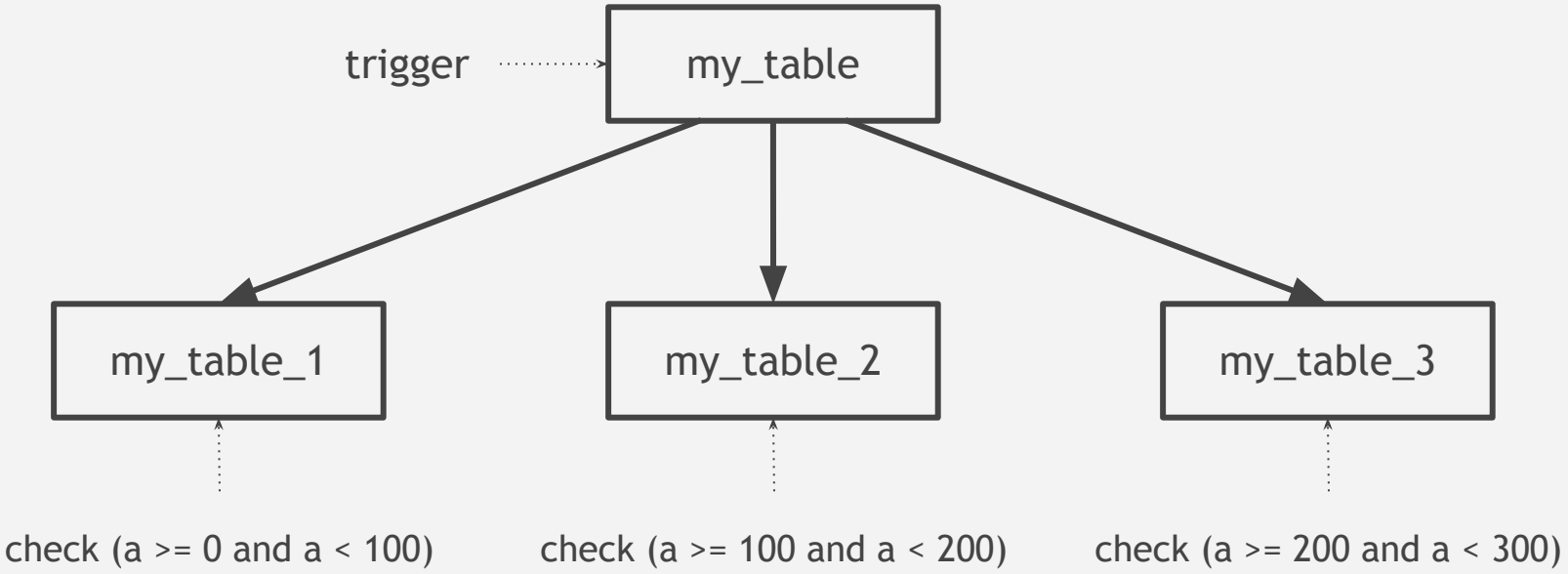
# Секционирование с **pg_pathman**


my_table
my_table_1
my_table_2
my_table_3

# Плюсы секционирования

1. Эффективное использование буфер-кэша в случае, когда часто используемые данные находятся в небольшом подмножестве партиций
2. Массовое удаление/загрузка данных
3. “Холодные” данные могут быть перемещены на более медленные/дешевые носители
4. Распределение данных между носителями, с целью снижения ожидания I/O
5. Раздельное обслуживание партиций

# Секционирование в PostgreSQL 9.6 и ниже



# Секционирование в PostgreSQL 9.6 и ниже

- Наследование
- Check constraints
- Триггер на вставку новых записей
- **LIST** и **RANGE** секционирование
- Оптимизация планировщика на основе механизма constraint exclusion (использует полный перебор)
- Отсутствие оптимизаций времени выполнения
- Отсутствие декларативного синтаксиса

# Что изменилось в PostgreSQL 10

- Декларативные секционирование
- Нативная поддержка INSERT-ов
- Каталог
  - таблица pg_partitioned_table
  - колонки relispartition, relpartbound в pg_class
- Constraint-ы по прежнему используются для оптимизации плана, но неявно

```
CREATE TABLE log (
     ts     TIMESTAMP,
     level  INTEGER,
     code   INTEGER,
     msg    TEXT)
PARTITION BY RANGE (ts);
```

```
CREATE TABLE log (
     ts     TIMESTAMP,
     level  INTEGER,
     code   INTEGER,
     msg    TEXT)
PARTITION BY RANGE (ts);

CREATE TABLE log_17_01 PARTITION OF log
FOR VALUES FROM ('2017-01-01') TO ('2017-02-01');
```

```
CREATE TABLE log (
     ts     TIMESTAMP,
     level  INTEGER,
     code   INTEGER,
     msg    TEXT)
PARTITION BY RANGE (ts);

CREATE TABLE log_17_01 PARTITION OF log
FOR VALUES FROM ('2017-01-01') TO ('2017-02-01');

CREATE TABLE log_17_02 PARTITION OF log
FOR VALUES FROM ('2017-02-01') TO ('2017-03-01')
PARTITION BY LIST (code);
```

```
CREATE TABLE log (
     ts     TIMESTAMP,
     level  INTEGER,
     code   INTEGER,
     msg    TEXT)
PARTITION BY RANGE (ts);

CREATE TABLE log_17_01 PARTITION OF log
FOR VALUES FROM ('2017-01-01') TO ('2017-02-01');

CREATE TABLE log_17_02 PARTITION OF log
FOR VALUES FROM ('2017-02-01') TO ('2017-03-01')
PARTITION BY LIST (code);

CREATE TABLE log_17_02_errors   PARTITION OF log_17_02 FOR VALUES IN (1);
CREATE TABLE log_17_02_warnings PARTITION OF log_17_02 FOR VALUES IN (2);
```

# pg_pathman

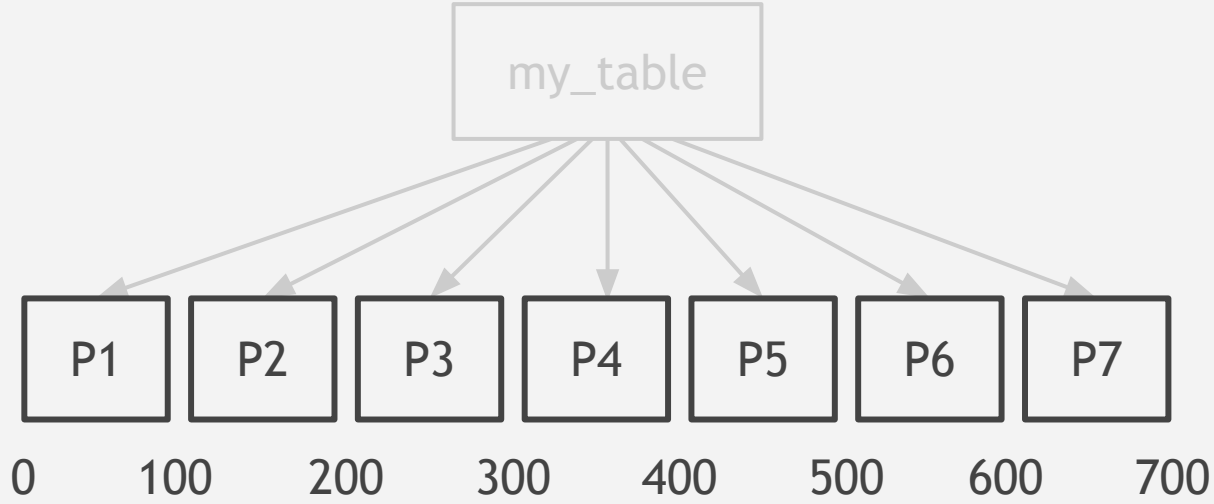
# pg_pathman

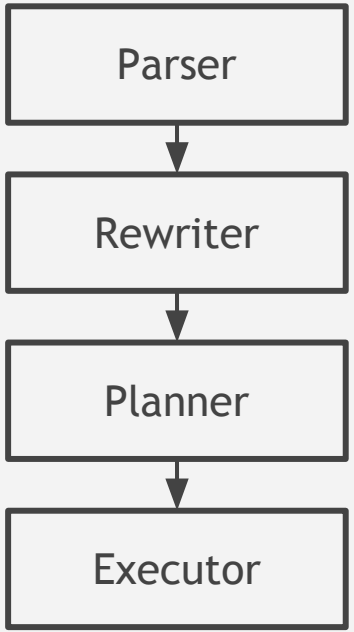
- оптимизация планировщика
- оптимизация executor-а
- **HASH** и **RANGE** секционирование
- custom node для INSERT-ов
- автоматическое создание секций + вызов пользовательского обработчика
- неблокирующая миграция данных
- поддержка FDW-таблиц в качестве секций
- удобный **API**

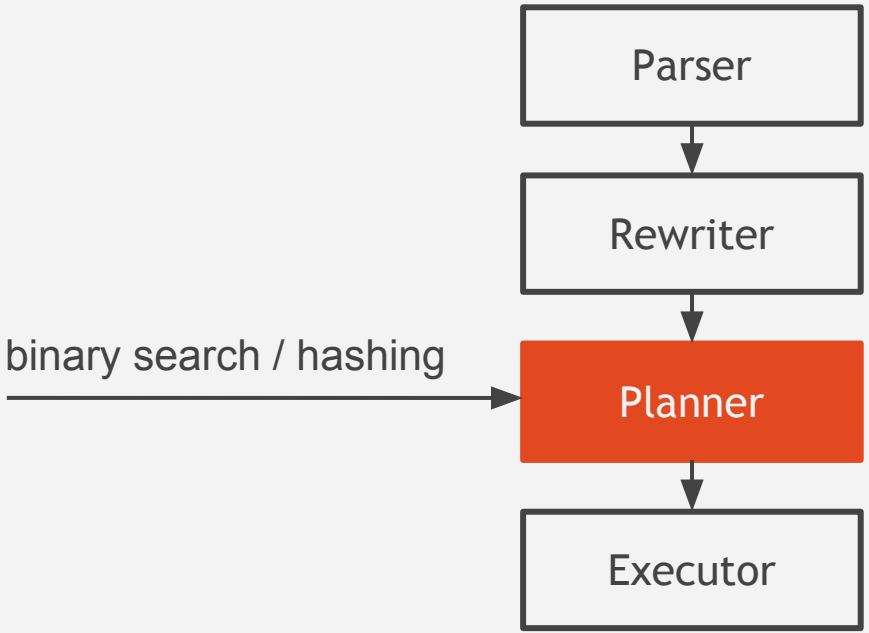
# Как работает pg_pathman

- конфигурация в таблицах
  - pathman_config
  - pathman_config_params
- check constraint-ы для хранения информации о границах партиции или хеш-значении:
  - CHECK (key >= **0** AND key < **100**)
  - CHECK (get_hash_part_idx(hashint4(key), 3) = **2**)
- кэширование
- использование хуков
- custom node
- plpgsql API

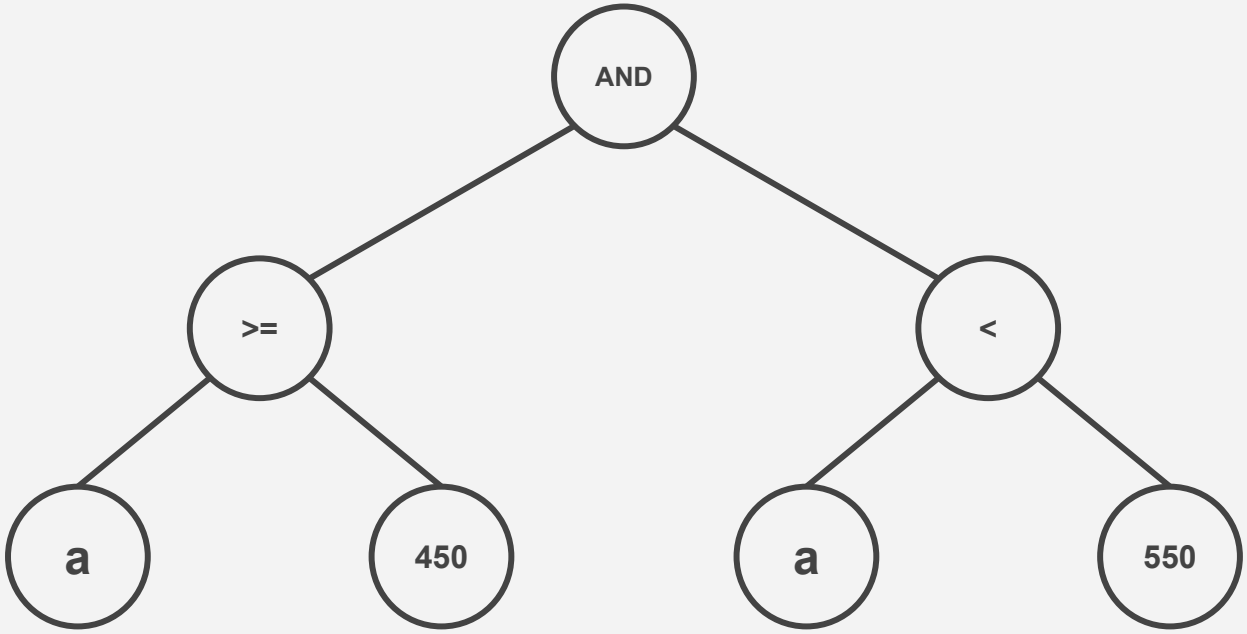
# Кэширование

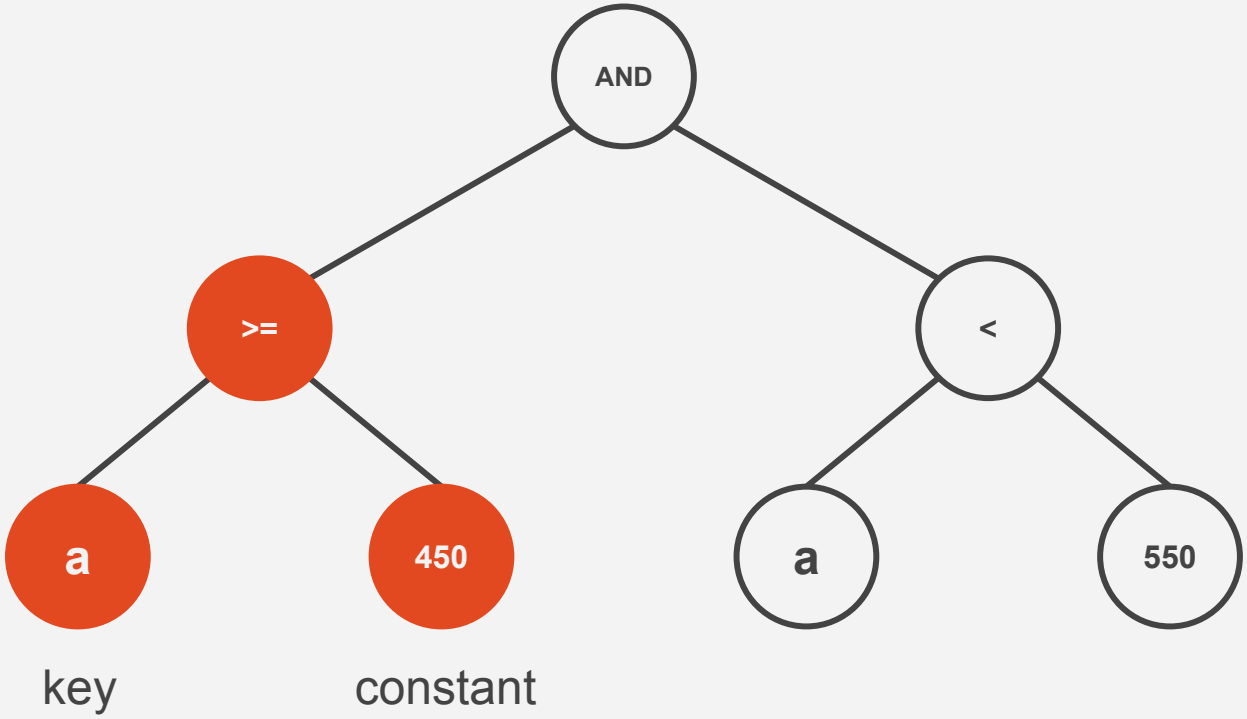



Parser
Rewriter
Planner
Executor

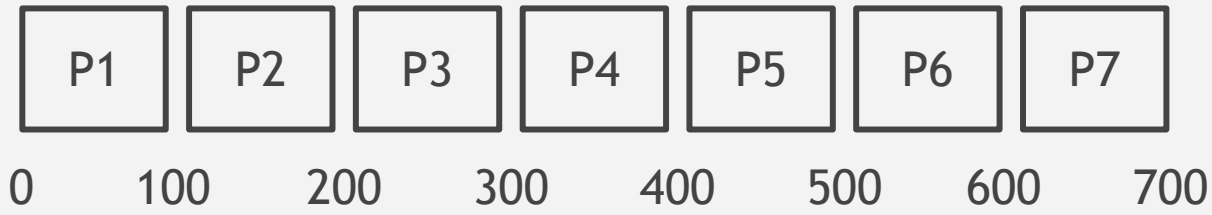

Parser
Rewriter
binary search / hashing
Planner
Executor

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

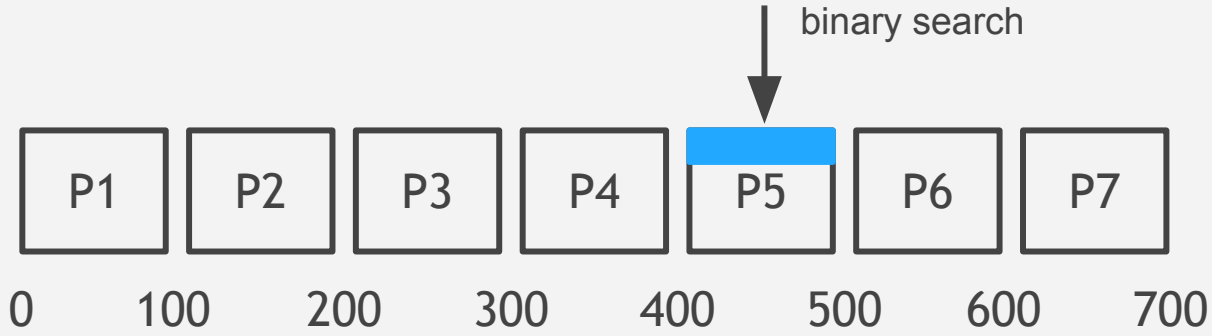

AND
>=
<
a
450
a
550


AND
>=
<
a
450
a
550
key
constant

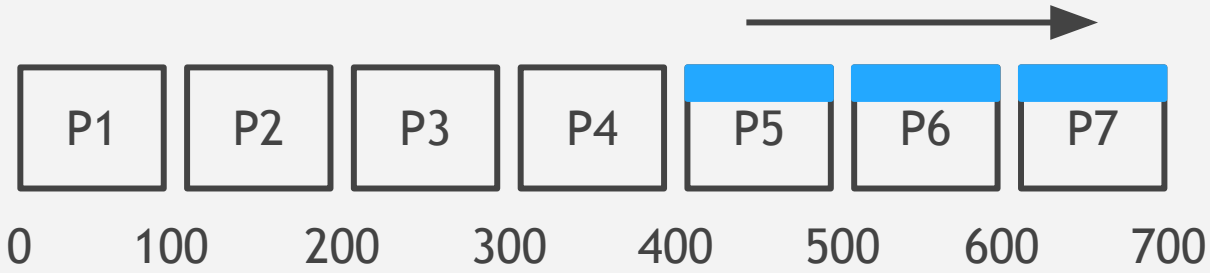
a >= 450



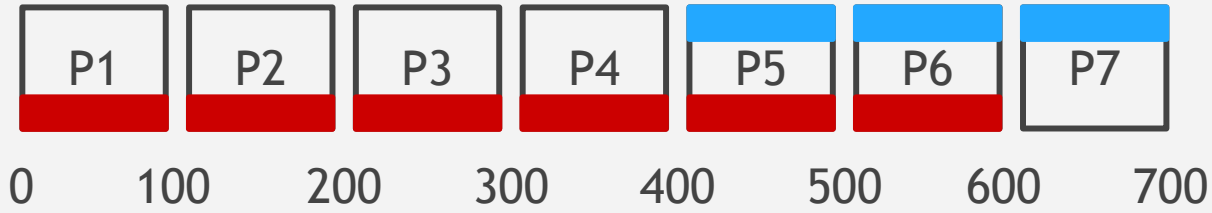
a >= 450



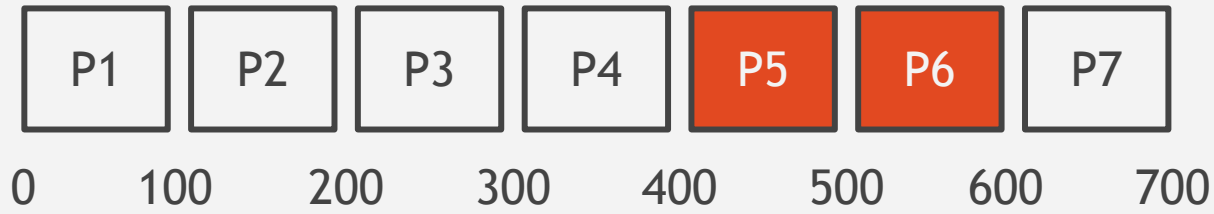
a >= 450



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)
```

# Push down условий фильтрации

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

# Push down условий фильтрации

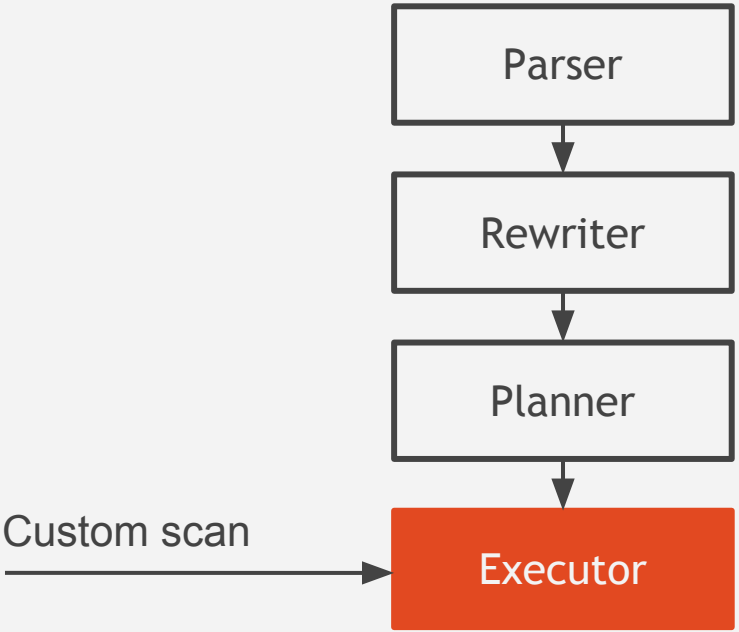
```
                QUERY PLAN
-------------------------------------
 Append
   ->  Seq Scan on my_table_5
         Filter: (a >= 450)
   ->  Seq Scan on my_table_6
         Filter: (a >= 450)
   ->  Seq Scan on my_table_7
         Filter: (a >= 450)
(9 rows)
```

# Push down условий фильтрации

```
                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)
```

# Push down условий фильтрации

```
                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)
```

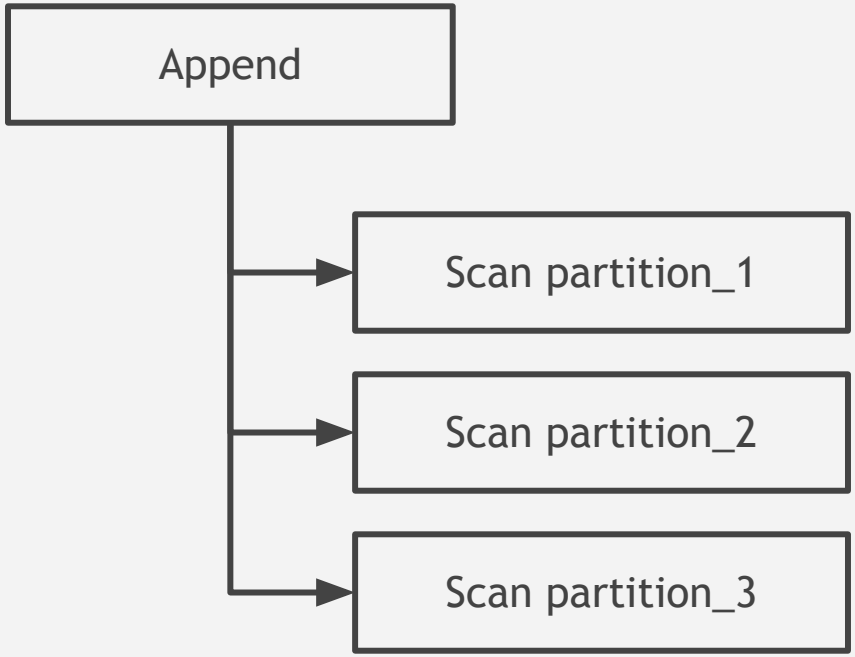

Parser
Rewriter
Planner
Custom scan
Executor

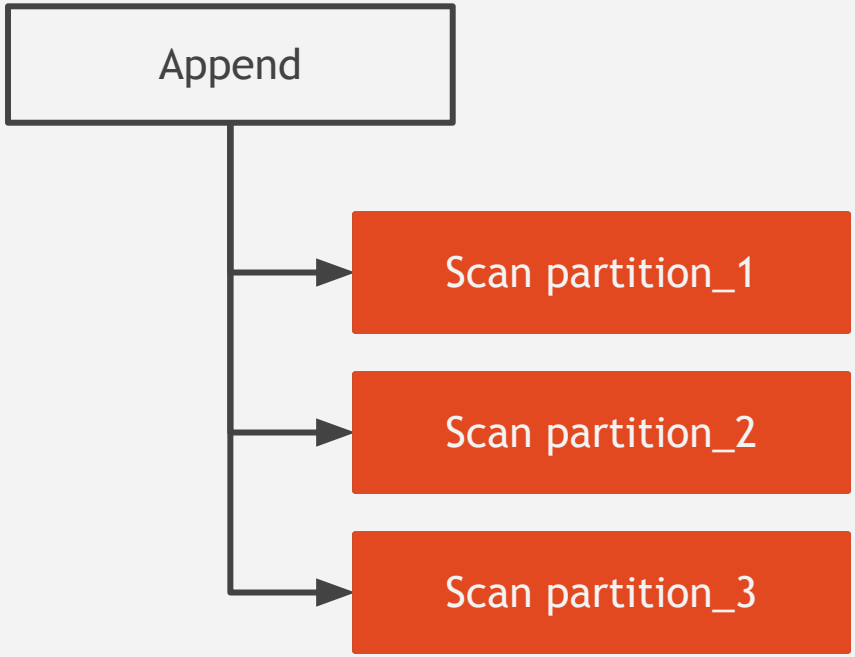
```
SELECT * FROM my_table
WHERE a = ?
```

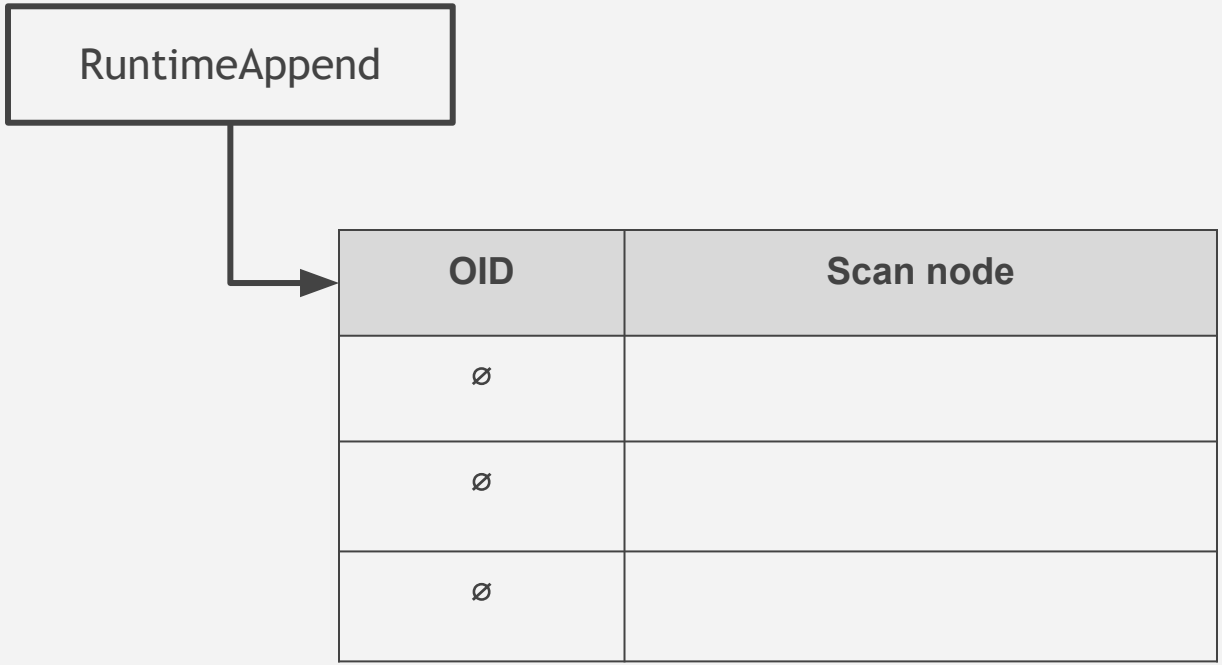
SELECT * FROM my_table
WHERE **a = ?**

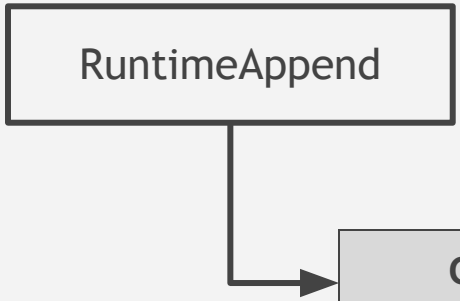
**Параметризованные запросы:**

- prepared statements
- nested loops
- подзапросы

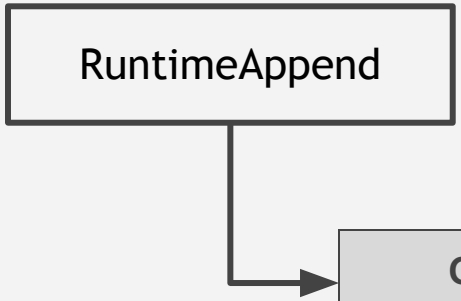

Append
Scan partition_1
Scan partition_2
Scan partition_3


Append
Scan partition_1
Scan partition_2
Scan partition_3

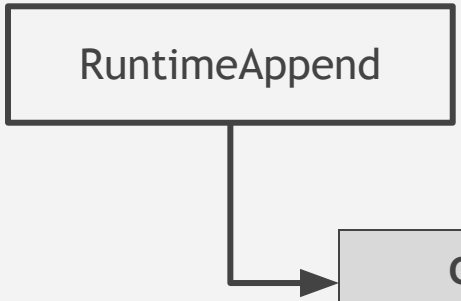

RuntimeAppend
OID
Scan node
∅
∅
∅



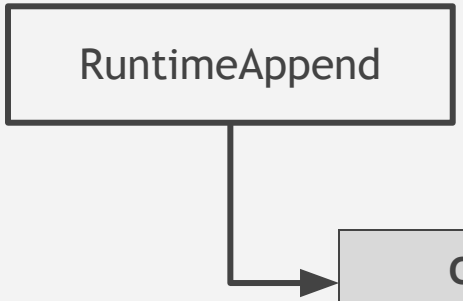

| OID | Scan node |
|---|---|
| 118179 | Scan partition_2 |
| ∅ | |
| ∅ | |




| OID | Scan node |
|---|---|
| 118179 | Scan partition_2 |
| 118173 | Scan partition_1 |
| ∅ | |




| OID | Scan node |
|---|---|
| 118179 | Scan partition_2 |
| 118173 | Scan partition_1 |
| ∅ | |




| OID | Scan node |
|---|---|
| 118179 | Scan partition_2 |
| 118173 | Scan partition_1 |
| ∅ | |

# Пример

```
SELECT * FROM my_table
WHERE a = (SELECT id FROM sometable LIMIT 1);
```

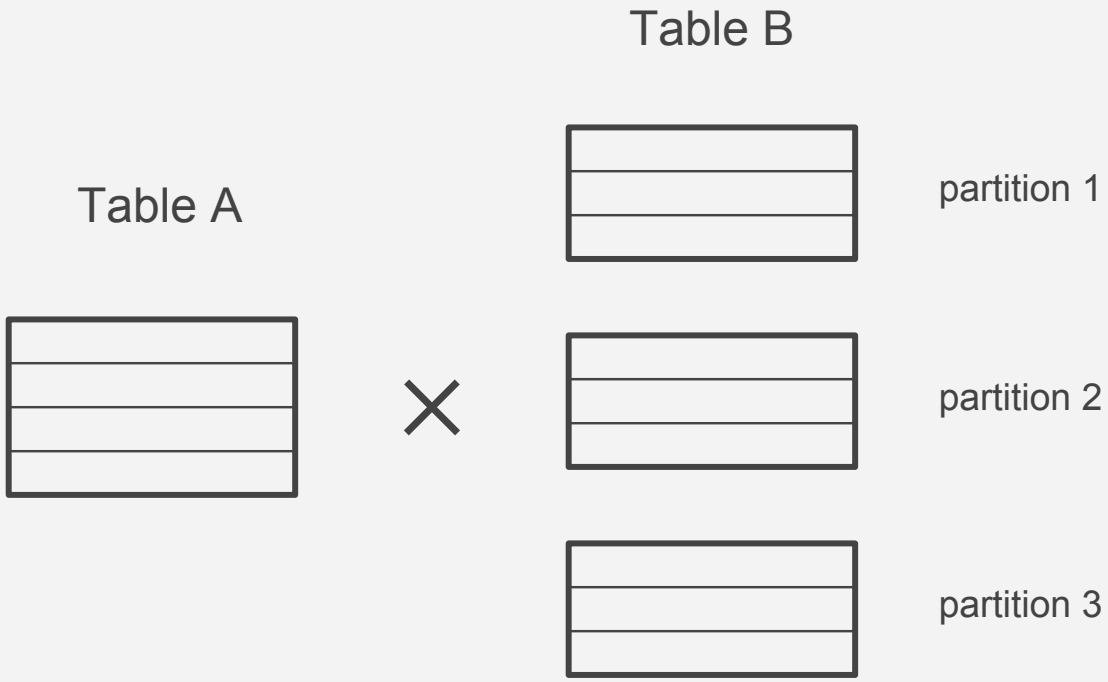
# Было:

```
EXPLAIN (ANALYZE, COSTS OFF, TIMING OFF)
SELECT * FROM my_table WHERE a = (SELECT id FROM sometable LIMIT 1);

                          QUERY PLAN
---------------------------------------------------------------
 Append (actual rows=2 loops=1)
   InitPlan 1 (returns $0)
     ->  Limit (actual rows=1 loops=1)
           ->  Seq Scan on sometable (actual rows=1 loops=1)
   ->  Seq Scan on my_table_1 (actual rows=2 loops=1)
         Filter: (a = $0)
         Rows Removed by Filter: 92
   ->  Seq Scan on my_table_2 (actual rows=0 loops=1)
         Filter: (a = $0)
         Rows Removed by Filter: 88
   ...
   ->  Seq Scan on my_table_100 (actual rows=0 loops=1)
         Filter: (a = $0)
         Rows Removed by Filter: 101
 Planning time: 2.799 ms
 Execution time: 2.615 ms
(306 rows)
```

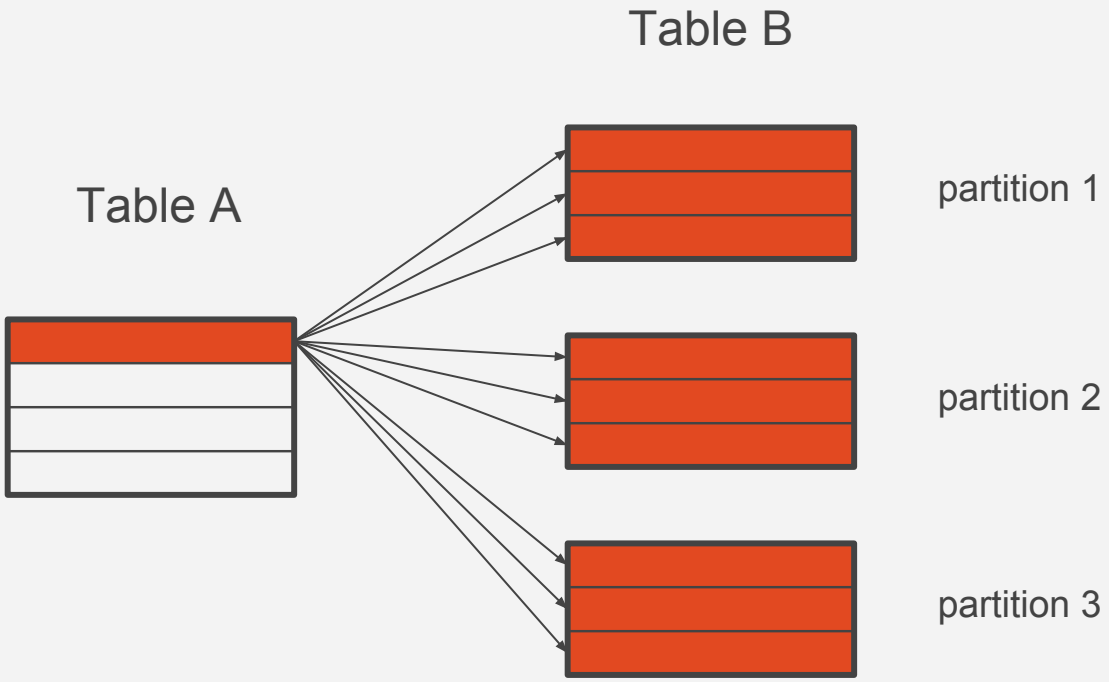
# Стало:

```
EXPLAIN (ANALYZE, COSTS OFF, TIMING OFF)
SELECT * FROM my_table WHERE a = (SELECT id FROM sometable LIMIT 1);

                          QUERY PLAN
---------------------------------------------------------------
 Custom Scan (RuntimeAppend) (actual rows=2 loops=1)
   Prune by: (my_table.a = $0)
   InitPlan 1 (returns $0)
     ->  Limit (actual rows=1 loops=1)
           ->  Seq Scan on sometable (actual rows=1 loops=1)
   ->  Seq Scan on my_table_1 (actual rows=2 loops=1)
         Filter: (a = $0)
         Rows Removed by Filter: 92
 Planning time: 2.811 ms
 Execution time: 0.106 ms
(10 rows)
```

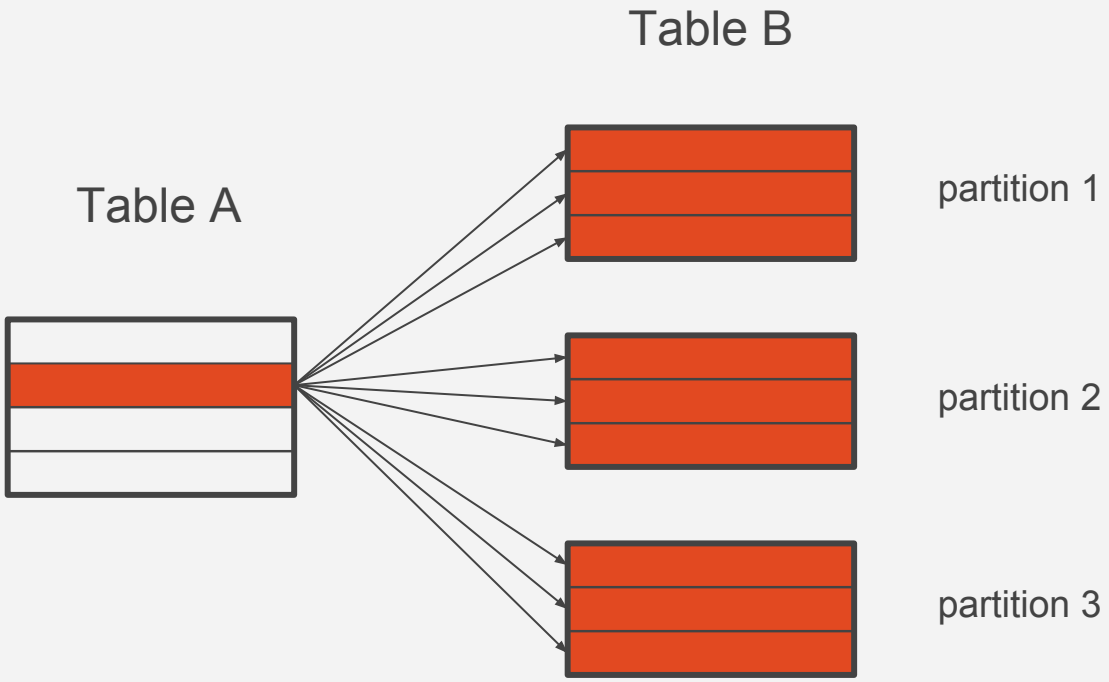
# Пример
## Nested loop



# Пример
## Nested loop

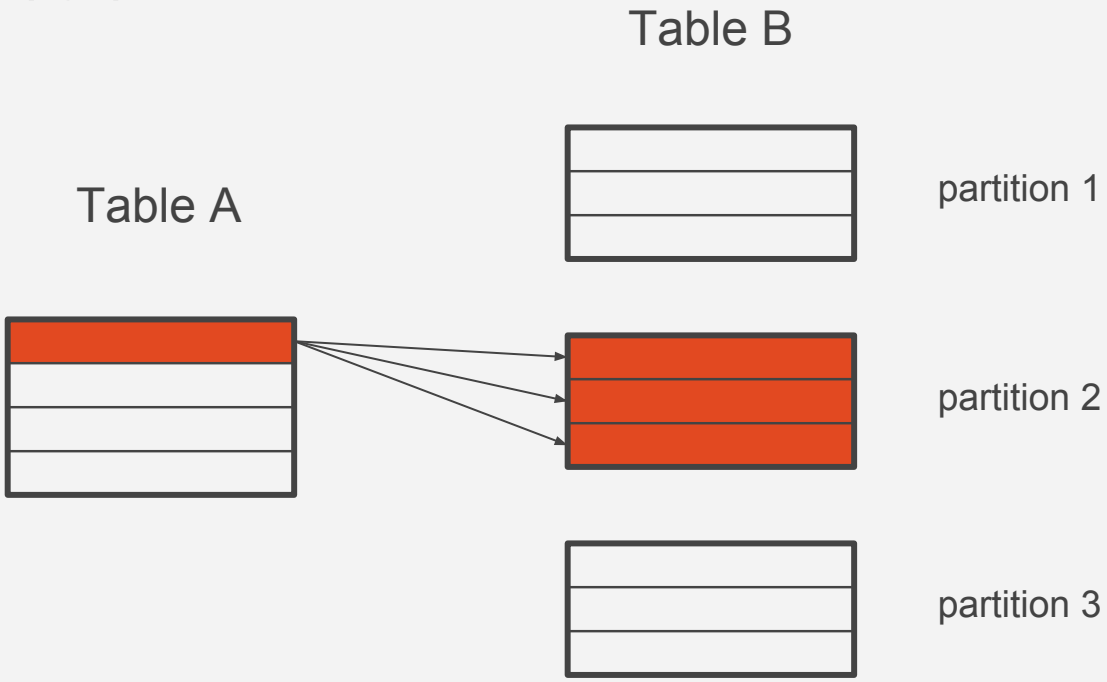


# Пример
## Nested loop



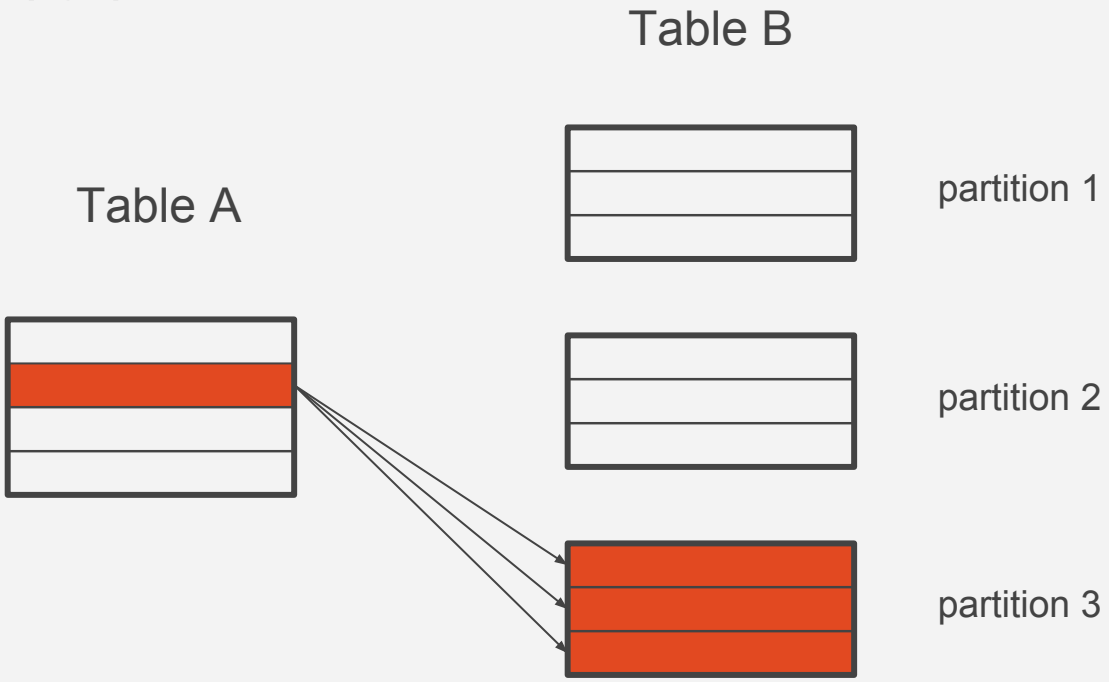
# Пример

## Nested loop with pg_pathman



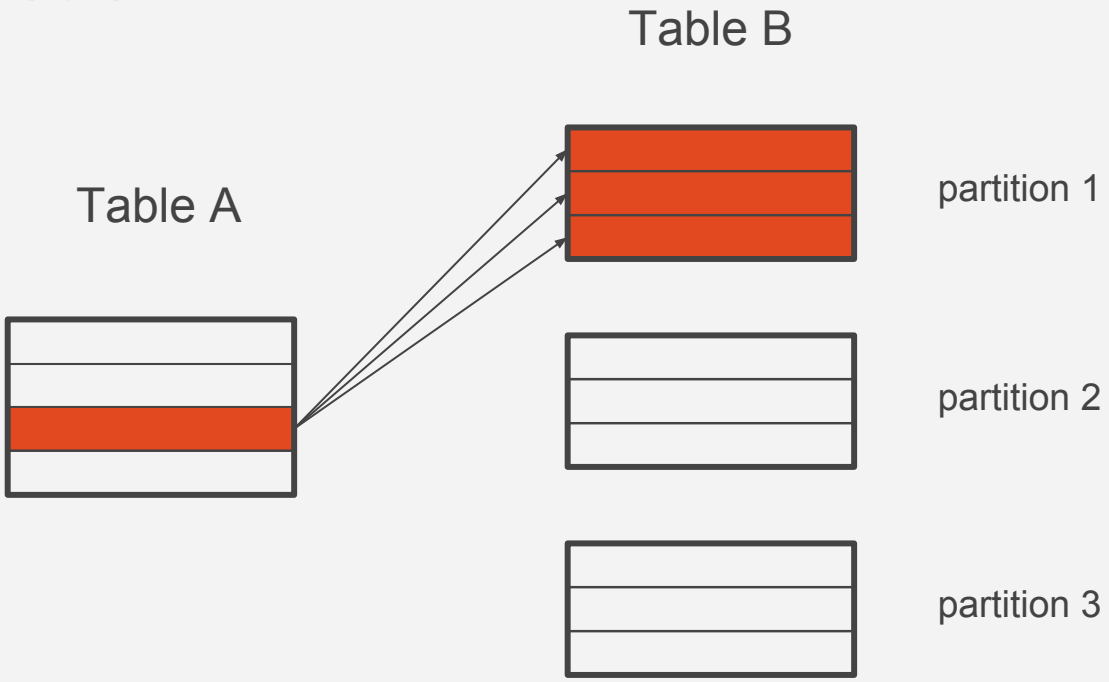
# Пример

## Nested loop with pg_pathman

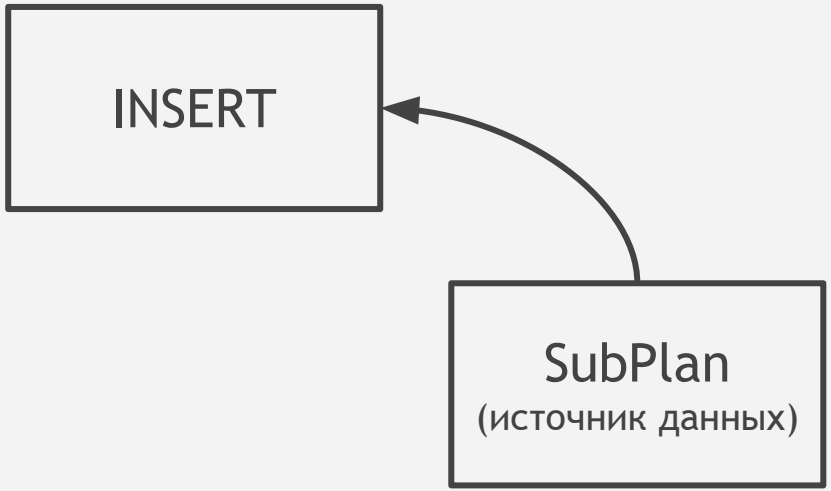


# Пример

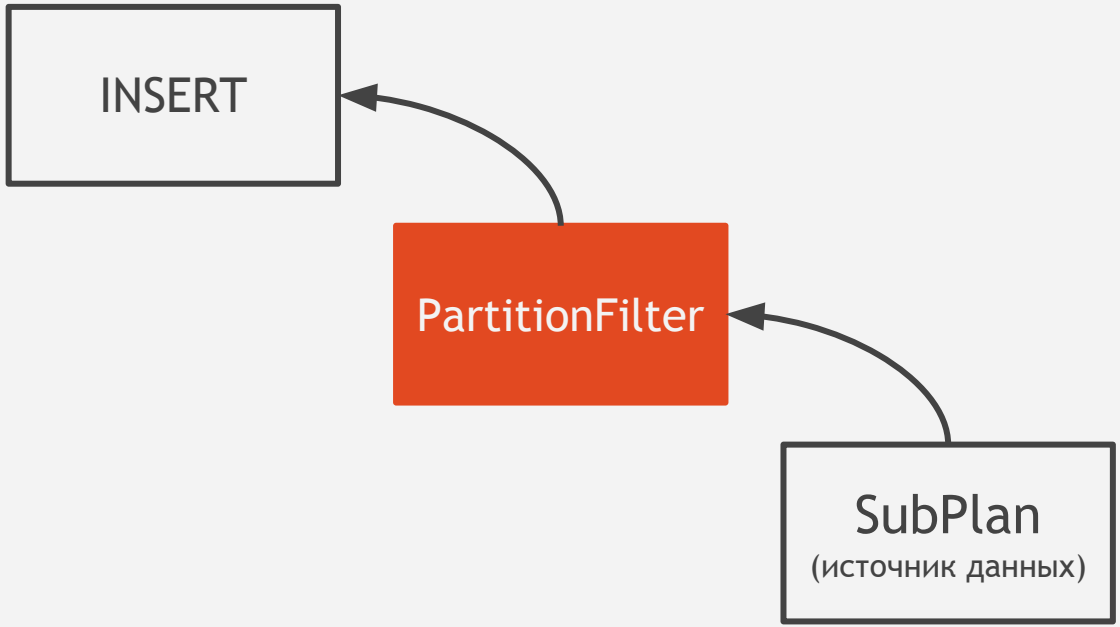
## Nested loop with pg_pathman



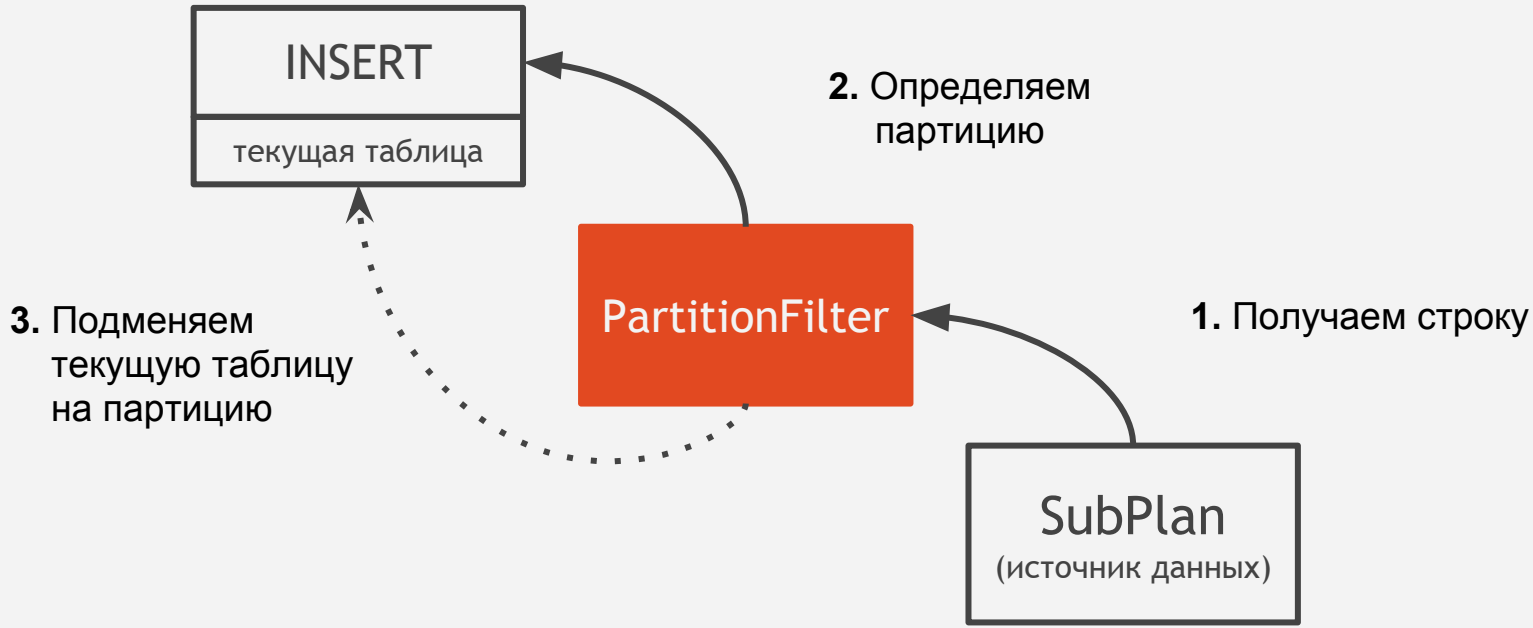
# INSERT



# INSERT



# INSERT


INSERT
текущая таблица
2. Определяем партицию
PartitionFilter
1. Получаем строку
3. Подменяем текущую таблицу на партицию
SubPlan
(источник данных)

# INSERT

**Было**

```
EXPLAIN (COSTS OFF)
INSERT INTO partitioned_table
SELECT generate_series(1, 10), random();
                QUERY PLAN
-----------------------------------------
 Insert on partitioned_table
   ->  Subquery Scan on “*SELECT*”
         -> Result
```

**Стало:**

```
EXPLAIN (COSTS OFF)
INSERT INTO partitioned_table
SELECT generate_series(1, 10), random();
                QUERY PLAN
-----------------------------------------
 Insert on partitioned_table
   -> Custom Scan (PartitionFilter)
         ->  Subquery Scan on “*SELECT*”
               -> Result
```

# pg_pathman's API

# HASH-секционирование

```
SELECT create_hash_partitions(
    <table_name>,
    <key>,
    <partitions_count>
);
```

Примечания:

- Ключ не должен быть NULL;
- Количество партиций нельзя изменить после разбиения.

# RANGE-секционирование

```
SELECT create_range_partitions(
    <table_name>,
    <key>,
    <start_value>,
    <interval>,
    <partitions_count>,
);
```

Примечания:

- Ключ не должен быть NULL;
- Над типом ключа должны быть определены операторы <, <=, =, >=, >
- Автоматическое создание партиций возможно, если тип поддерживает оператор +
- Если количество партиций не задано, pg_pathman определит его автоматически

# Если тип не поддерживает оператор +

Например, если ключ имеет тип TEXT:

```
CREATE TABLE abc (key TEXT NOT NULL);
SELECT create_range_partitions('abc',       -- table name
                               'key',       -- key column
                               NULL::TEXT,  -- start value is not set
                               NULL::TEXT,  -- interval is not set
                               0);          -- partition count set to zero

SELECT add_range_partition('abc', 'a'::TEXT, 'b'::TEXT);
SELECT add_range_partition('abc', 'b'::TEXT, 'c'::TEXT);
...
```

# Секционирование по выражению

```
CREATE TABLE abc (doc jsonb not null);
SELECT create_hash_partitions('abc', 'doc->>''name''', 3);

INSERT INTO abc VALUES ('{"name": "Oleg"}'::jsonb);
INSERT INTO abc VALUES ('{"name": "Teodor"}'::jsonb);
INSERT INTO abc VALUES ('{"name": "Alexander"}'::jsonb);

EXPLAIN (COSTS OFF) SELECT * FROM abc WHERE doc->>'name' = 'Teodor';

                        QUERY PLAN
-----------------------------------------------------------
 Append
   ->  Seq Scan on abc_2
         Filter: ((doc ->> 'name'::text) = 'Teodor'::text)
(3 rows)
```

# pg_pathman's API

```
SELECT append_range_partition('abc');
SELECT prepend_range_partition('abc');
SELECT add_range_partition('abc',
                           '2016-05-01'::date,
                           '2016-06-01'::date);
```

# pg_pathman's API

```
SELECT append_range_partition('abc');
SELECT prepend_range_partition('abc');
SELECT add_range_partition('abc',
                           '2016-05-01'::date,
                           '2016-06-01'::date);

SELECT attach_range_partition('abc', 'some_table',
                              '2016-06-01'::date,
                              '2016-07-01'::date);
SELECT detach_range_partition('some_table');
SELECT replace_hash_partition('abc_0', 'some_table');
```

# pg_pathman's API

```
SELECT append_range_partition('abc');
SELECT prepend_range_partition('abc');
SELECT add_range_partition('abc',
                           '2016-05-01'::date,
                           '2016-06-01'::date);

SELECT attach_range_partition('abc', 'some_table',
                              '2016-06-01'::date,
                              '2016-07-01'::date);
SELECT detach_range_partition('some_table');
SELECT replace_hash_partition('abc_0', 'some_table');

SELECT merge_range_partitions('abc_1', 'abc_2');
SELECT split_range_partition('abc_1', '2016-02-01'::date);
```

# pg_pathman's API

```
SELECT append_range_partition('abc');
SELECT prepend_range_partition('abc');
SELECT add_range_partition('abc',
                           '2016-05-01'::date,
                           '2016-06-01'::date);

SELECT attach_range_partition('abc', 'some_table',
                              '2016-06-01'::date,
                              '2016-07-01'::date);
SELECT detach_range_partition('some_table');
SELECT replace_hash_partition('abc_0', 'some_table');

SELECT merge_range_partitions('abc_1', 'abc_2');
SELECT split_range_partition('abc_1', '2016-02-01'::date);

SELECT drop_range_partition('abc_0');
SELECT drop_partitions('parent_table', false);
```

# pg_pathman's API

```
SELECT parent, partition, range_min, range_max,
       pg_size_pretty(pg_relation_size(partition)) AS size
FROM pathman_partition_list;
```

```
   parent    |  partition   | range_min | range_max |   size
-------------+--------------+-----------+-----------+------------
 test_hash   | test_hash_0  |           |           | 96 kB
 test_hash   | test_hash_1  |           |           | 88 kB
 test_hash   | test_hash_2  |           |           | 96 kB
 test_hash   | test_hash_3  |           |           | 96 kB
 test_range  | test_range_1 | 1         | 2001      | 72 kB
 test_range  | test_range_2 | 2001      | 4001      | 72 kB
 test_range  | test_range_3 | 4001      | 6001      | 72 kB
 test_range  | test_range_4 | 6001      | 8001      | 72 kB
 test_range  | test_range_5 | 8001      |           | 8192 bytes
```

# Неблокирующая миграция данных

```
SELECT partition_table_concurrently('test');
SELECT stop_concurrent_part_task('test');
```

```
SELECT * FROM pathman_concurrent_part_tasks;
```

```
 userid | pid  | dbid  | relid | processed | status
--------+------+-------+-------+-----------+---------
 user   | 7367 | 16384 | test  |    472000 | working
(1 row)
```

# Дополнительные параметры

```
SELECT set_interval      ('abc', '2 months');
SELECT set_auto          ('abc', true);
SELECT set_enable_parent ('abc', true);
SELECT set_init_callback ('abc', 'my_func(jsonb)');
```

**set_auto(<таблица>, <true/false>);**

Активирует автоматическое создание партиций. Тип ключа секционирования должен поддерживать оператор сложения (+)

Примеры типов:

- INTEGER
- FLOAT
- DECIMAL
- DATE
- TIMESTAMP

# set_enable_parent(<таблица>, <true/false>);

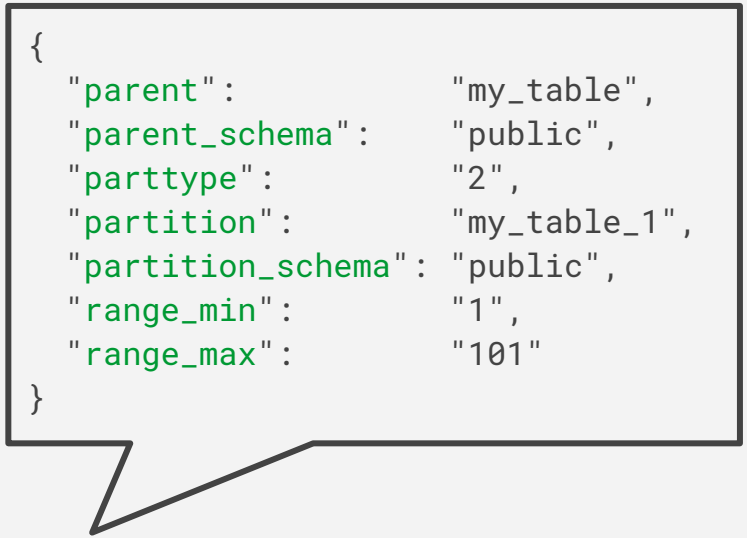
Включает/выключает родительскую таблицу в (из) плана запроса

```
SELECT set_enable_parent('abc', false);

EXPLAIN (COSTS OFF) SELECT * FROM abc;

       QUERY PLAN
-------------------------
 Append
   ->  Seq Scan on abc_1
   ->  Seq Scan on abc_2
   ->  Seq Scan on abc_3
(4 rows)
```

```
SELECT set_enable_parent('abc', true);

EXPLAIN (COSTS OFF) SELECT * FROM abc;

       QUERY PLAN
-------------------------
 Append
   ->  Seq Scan on abc
   ->  Seq Scan on abc_1
   ->  Seq Scan on abc_2
   ->  Seq Scan on abc_3
(5 rows)
```

## set_init_callback(<таблица>, <имя_функции>)

Устанавливает пользовательскую функцию в качестве обработчика создания новой партиции




```
CREATE FUNCTION my_callback(args JSONB) ... ;
SELECT set_init_callback('abc', 'my_callback(jsonb)');
```

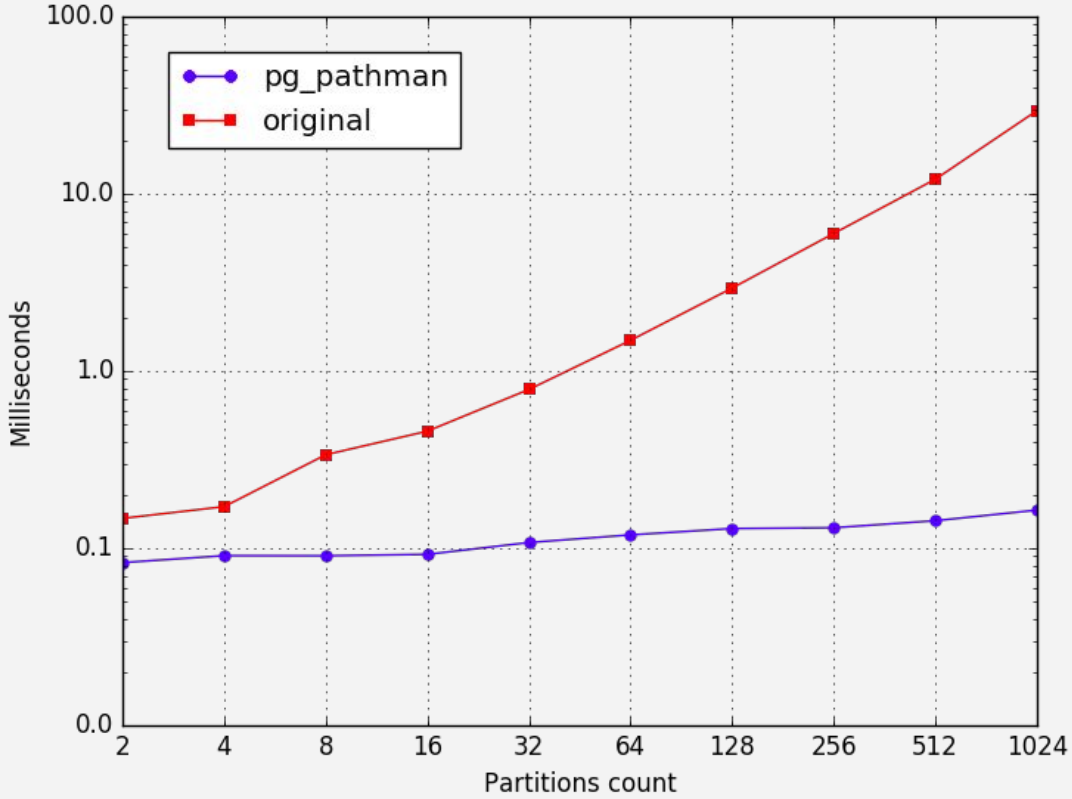
# Пример №1: именование партиций

```
CREATE OR REPLACE FUNCTION my_callback(params jsonb)
RETURNS VOID AS
$$
DECLARE
    range_min    timestamp;
    new_relname  text;
BEGIN
    range_min := (params->>'range_min')::timestamp;

    -- generate new name for partition based
    -- on its parent name and lower bound
    new_relname := format('%s_%s',
                          params->>'parent',
                          to_char(range_min, 'YYYY_MM'));

    -- rename partition
    EXECUTE format('ALTER TABLE %s.%s RENAME TO %s',
                   params->>'partition_schema',
                   params->>'partition',
                   new_relname);
END
$$ LANGUAGE plpgsql;
```

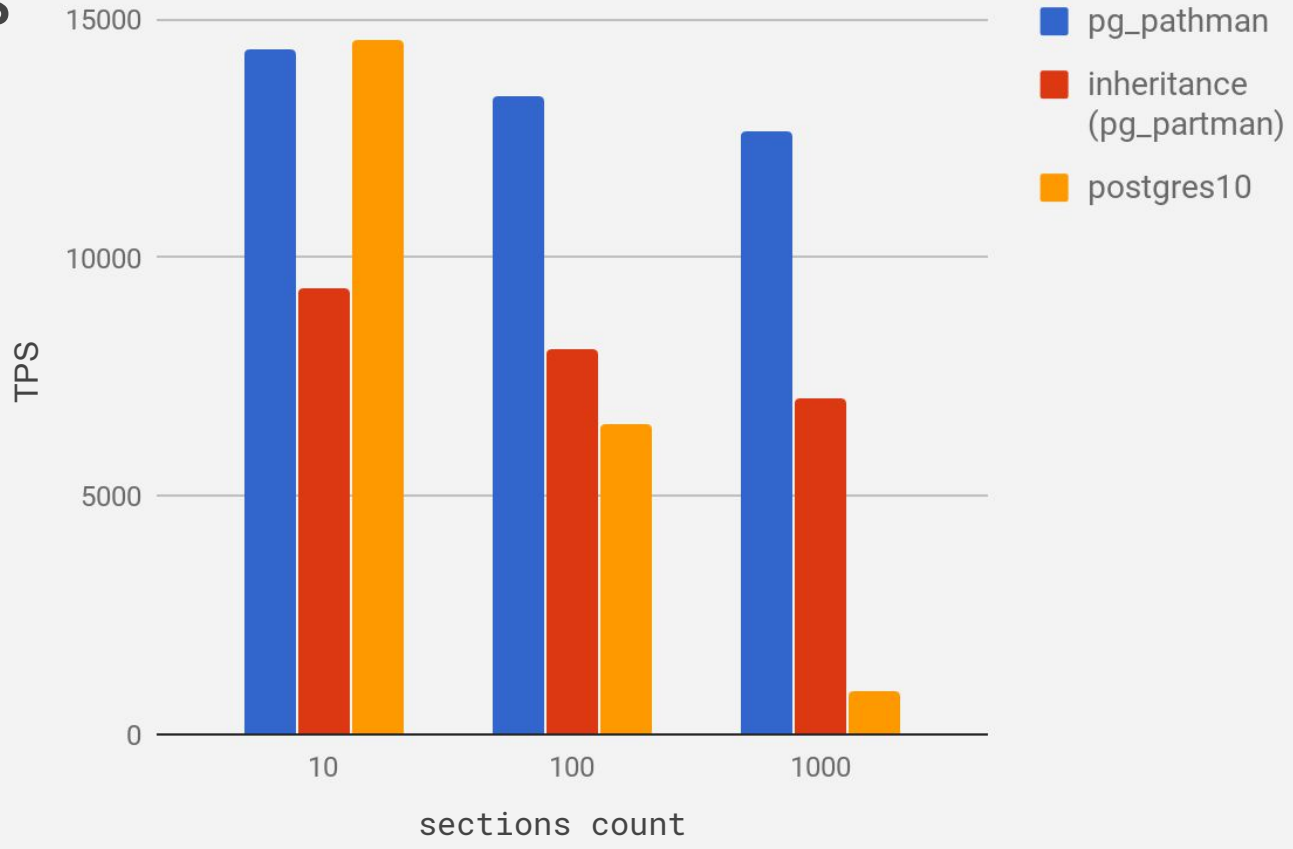
# Пример №2: распределение партиций по tablespace-ам

```
-- suppose we have three tablespaces: ts_0, ts_1, ts_2
CREATE SEQUENCE ts_counter;

CREATE OR REPLACE FUNCTION my_callback(params jsonb)
RETURNS VOID AS
$$
DECLARE
    ts_name  text;
BEGIN
  -- calculate tablespace name
    ts_name = 'ts_' || nextval('ts_counter') % 3;

    -- move partition to the tablespace
    EXECUTE format(ALTER TABLE %s.%s SET TABLESPACE %s',
        params->>'partition_schema',
        params->>'partition',
        ts_name);
END
$$
LANGUAGE plpgsql;
```

# Benchmarks
## planning time



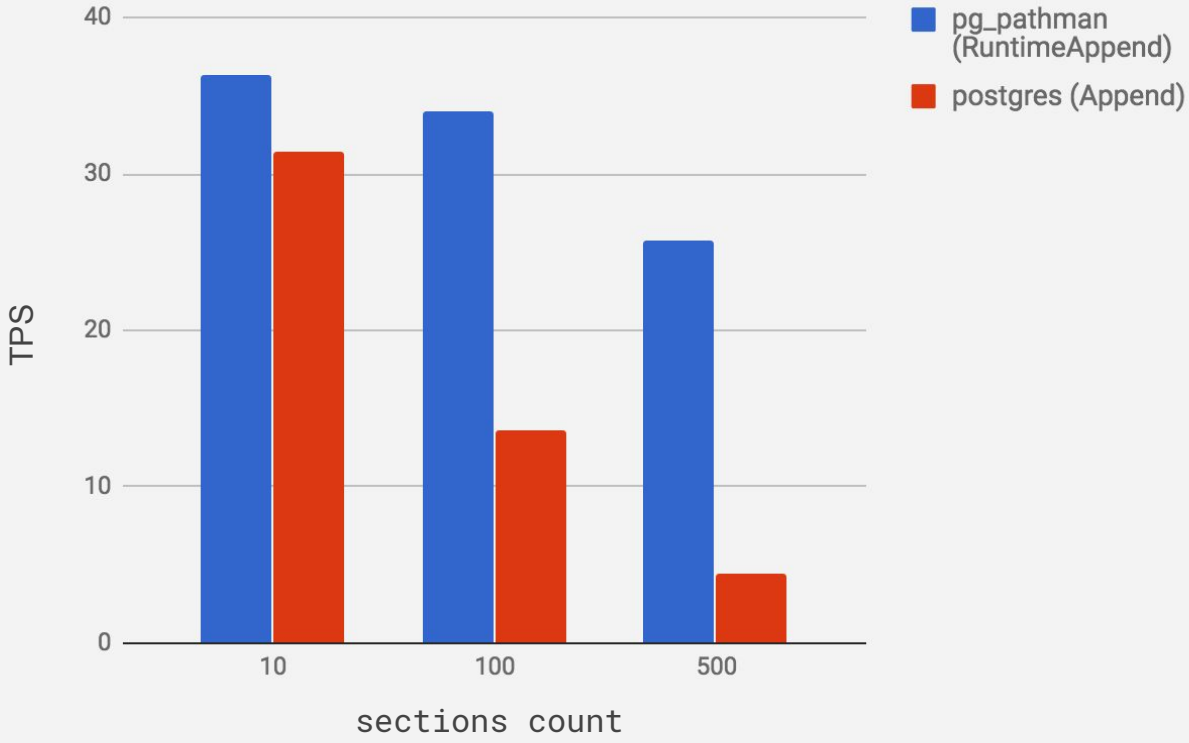
# Benchmarks
## INSERT



# Benchmarks
## parameterized query

```
SELECT * FROM partitioned
WHERE key = (
    SELECT part_key
    FROM sometable
    WHERE id = $1
    LIMIT 1
);
```

Rows per partition: 10 000



# Roadmap

| Задача | Ветка на github |
|---|---|
| 1. Многоуровневое секционирование | rel_future_multilevel |
| 2. Custom scan node для UPDATE | rel_future_update_node |
| 3. Foreign Key для партицированных таблиц | rel_future_ri |
| 4. NULL-партиции | |
| 5. Интеграция с синтаксисом Postgres 10 | |
| 6. LIST-секционирование | |


Спасибо за внимание!

github.com/postgrespro/pg_pathman

Ильдар Мусин i.musin@postgrespro.ru
Дмитрий Иванов d.ivanov@postgrespro.ru
Ильдус Курбангалиев i.kurbangaliev@postgrespro.ru