

PostgreSQL для начинающих

#4: анализ запросов

2023 Edition

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

SQL – декларативный язык



вы описываете, **что** хотите получить

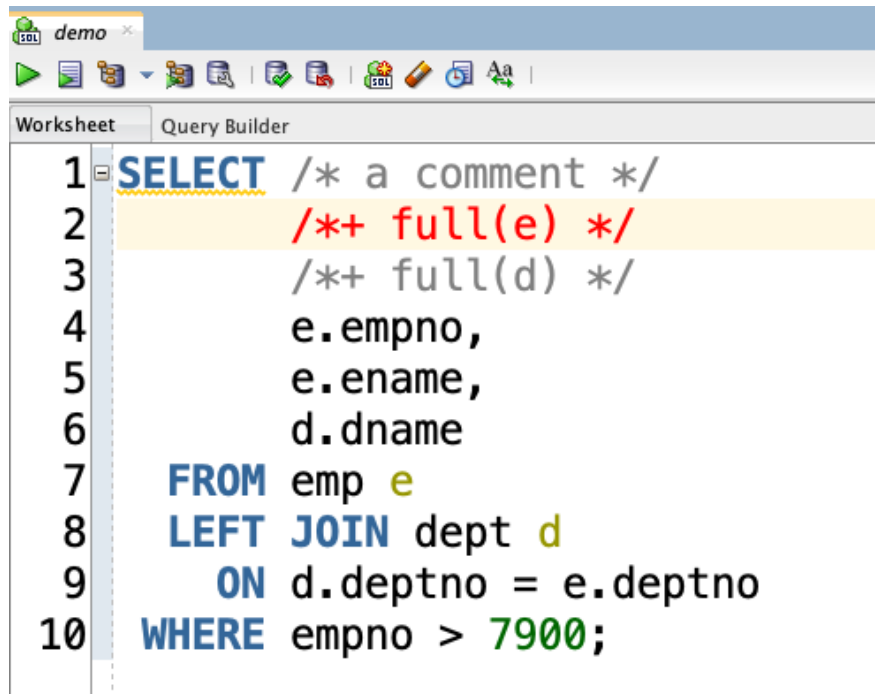


но СУБД лучше «знает», **как** это сделать

какие индексы использовать, в каком порядке
соединять таблицы и накладывать условия, ...

SQL – декларативный язык

некоторые СУБД принимают «подсказки»



```
1 SELECT /* a comment */  
2      /*+ full(e) */  
3      /*+ full(d) */  
4      e.empno,  
5      e.ename,  
6      d.dname  
7 FROM emp e  
8 LEFT JOIN dept d  
9      ON d.deptno = e.deptno  
10 WHERE empno > 7900;
```

PostgreSQL –

но всегда готов рассказать,
как конкретно он планирует
выполнять ваш запрос

(или уже как-то выполнил 😊)



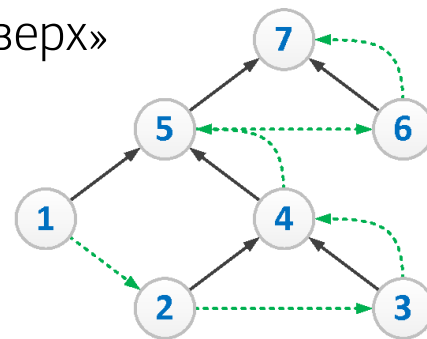
План запроса

Дерево в текстовом виде

каждый элемент – одна из операций

получение данных, построение битовых карт, обработка данных,
операция над множествами, соединение, вложенный запрос, ...

выполнение плана – обход дерева «снизу - вверх»



План запроса

«Почему тут выполнялось так долго?»

неэффективный **алгоритм**

неактуальная **статистика**

нехватка **ресурсов**

(CPU, RAM, storage)



План запроса

Query Text: explain (analyze, buffers, costs off)

```
SELECT * FROM pg_class WHERE (oid, relname) = (  
    SELECT oid, relname FROM pg_class WHERE relkind = 'r' LIMIT 1  
);
```

Index Scan using pg_class_relname_nsp_index on pg_class (actual time=0.049..0.050 rows=1 loops=1)

Index Cond: (relname = \$1)

Filter: (oid = \$0)

Buffers: shared hit=4

InitPlan 1 (returns \$0,\$1)

-> Limit (actual time=0.019..0.020 rows=1 loops=1)

Buffers: shared hit=1

-> Seq Scan on pg_class pg_class_1 (actual time=0.015..0.015 rows=1 loops=1)

Filter: (relkind = 'r'::"char")

Rows Removed by Filter: 5

Buffers: shared hit=1

План запроса



План запроса

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

EXPLAIN [**ANALYZE**] [**VERBOSE**] *оператор*

Здесь допускается *параметр*:

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

Где *оператор*:

{ **SELECT** | **INSERT** | **UPDATE** | **DELETE** | **CREATE TABLE ... AS ...** }

<https://postgrespro.ru/docs/postgresql/15/sql-explain>

<https://postgrespro.ru/docs/postgresql/15/using-explain>

План запроса

EXPLAIN . . .

как СУБД **планирует** выполнять запрос

не изменяется от запуска к запуску

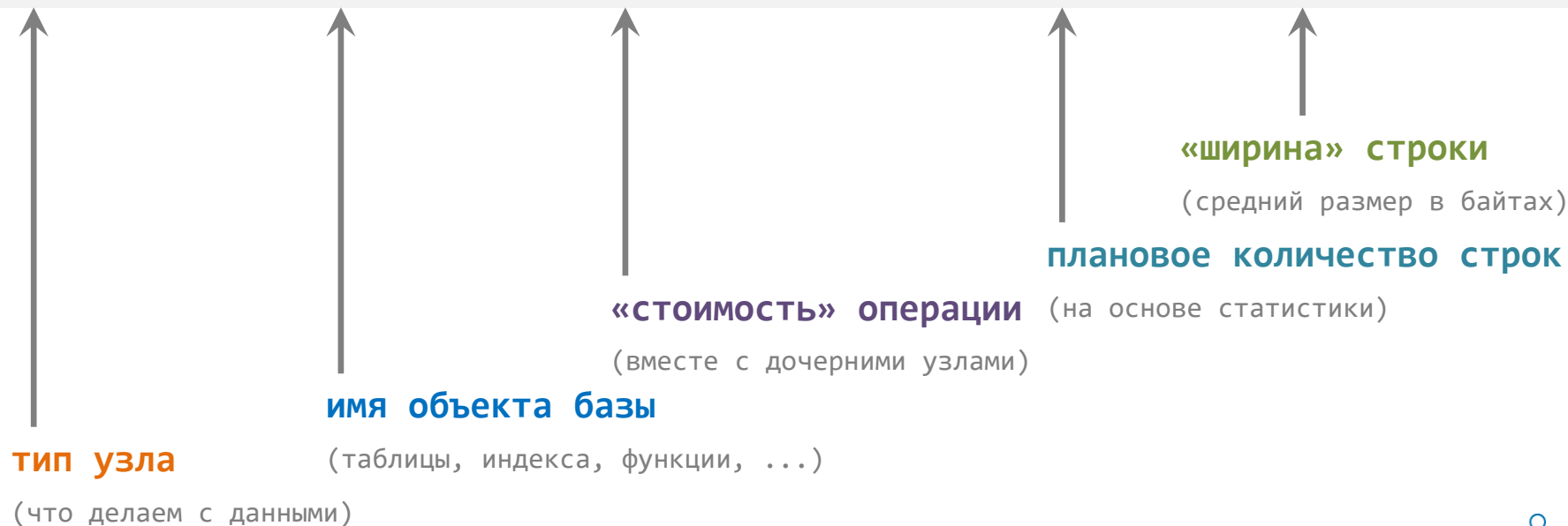
на основе накопленной **статистики** о данных базы

<https://postgrespro.ru/docs/postgresql/15/planner-stats>

План запроса

```
EXPLAIN SELECT * FROM pg_class;
```

Seq Scan on pg_class (cost=0.00..34.90 rows=418 width=265)



«Стоимость» операции

Вычисляемая **абстрактная** величина

от количества планируемых операций

(чтение страницы данных, обработка отдельной строки, ...)

Seq Scan on pg_class (cost=0.00..34.90 rows=418 width=265)



для первой строки



для **последней** строки

«Стоимость» операции



«Стоимость» операции

```
EXPLAIN SELECT * FROM pg_class ORDER BY relname;
```

```
Index Scan using pg_class_relname_nsp_index on pg_class (cost=0.27..44.15 rows=418 width=265)
```

```
SET enable_indexscan = off;
```

```
EXPLAIN SELECT * FROM pg_class ORDER BY relname;
```

```
Sort (cost=53.10..54.14 rows=418 width=265)
```

```
Sort Key: relname
```

```
-> Seq Scan on pg_class (cost=0.00..34.90 rows=418 width=265)
```

Seq Scan «дешевле», но необходимость сортировки все усложняет

«Стоимость» операции

“Эти параметры конфигурации дают возможность **грубо влиять** на планы, выбираемые оптимизатором запросов. Если автоматически выбранный оптимизатором план конкретного запроса оказался неоптимальным, **в качестве временного решения** можно воспользоваться одним из этих параметров и вынудить планировщик выбрать другой план.”

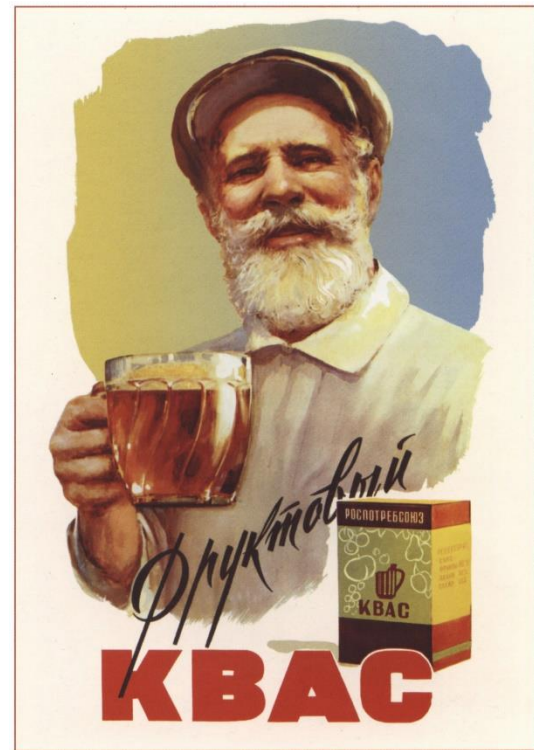
<https://postgrespro.ru/docs/postgresql/15/runtime-config-query>

```
SET enable_{operation} = off [on];
```

“Улучшить качество планов, выбираемых планировщиком, можно и более подходящими способами, в частности, скорректировать **константы стоимости** ...”

```
SET {operation}_cost = {floating point};
```

```
*_page_cost, *_tuple_cost, *_setup_cost, *_operator_cost, jit_*_above_cost  
*_size
```



546. Лапаев С.
Фруктовый квас. 1959

Изучаем настройки

```
EXPLAIN (SETTINGS) SELECT * FROM pg_class;
```

```
Seq Scan on pg_catalog.pg_class (cost=0.00..34.90 rows=418 width=265)
```

```
Settings: constraint_exclusion = 'on', cpu_tuple_cost = '0.05', effective_cache_size = '44GB',  
effective_io_concurrency = '128', random_page_cost = '1.1', temp_buffers = '256MB',  
work_mem = '512MB'
```

<https://postgrespro.ru/docs/postgresql/15/runtime-config>

Количество и «ширина» строк

Ориентировочный объем необходимой памяти

можно оценить **объем resultset** и памяти

Seq Scan on pg_class (cost=0.00..34.90 rows=418 width=265)

Тип поля	байт	Тип поля	байт
boolean	1	date	4
smallint	2	timestamp	8
integer	4	real	4
bigint	8	double precision	8
		varchar / text	n + 4

количество строк

«ширина» строки

Количество и «ширина» строк

```
EXPLAIN VERBOSE SELECT * FROM pg_class;
```

```
Seq Scan on pg_catalog.pg_class (cost=0.00..34.90 rows=418 width=265)
```

```
Output: oid, relname, relnamespace, reltype, ...
```

$418 \times 265 = 110\,770 = 108.1\text{KB}$

```
EXPLAIN VERBOSE SELECT oid FROM pg_class;
```

```
Seq Scan on pg_catalog.pg_class (cost=0.00..34.90 rows=418 width=4)
```

```
Output: oid
```

$418 \times 4 = 1\,672 = 1.6\text{KB}$

Output – поля, возвращаемые узлом

Реальные показатели

EXPLAIN ANALYZE ...

EXPLAIN (ANALYZE [, ...]) ...

протокол **фактического выполнения** запроса

EXPLAIN ANALYZE DELETE – не лучшая идея

```
BEGIN;                -- завернем выполнение в транзакцию
EXPLAIN ANALYZE ...;
ROLLBACK;             -- откатим изменения транзакции
```

Реальные показатели

```
EXPLAIN ANALYZE SELECT * FROM pg_class;
```

```
Seq Scan on pg_class (cost=0.00..34.90 rows=418 width=265) (actual time=0.004..0.044 rows=418 loops=1)
```

Planning Time: 0.056 ms

Execution Time: 0.085 ms

общее время выполнения
время планирования

время получения
первой/последней строки

фактическое
количество строк

количество повторов

Planning/Execution Time – время для всего плана

Убираем ненужное

```
EXPLAIN (ANALYZE, COSTS off) SELECT * FROM pg_class;
```

```
Seq Scan on pg_class (actual time=0.004..0.044 rows=418 loops=1)
```

```
Planning Time: 0.056 ms
```

```
Execution Time: 0.085 ms
```

```
EXPLAIN (ANALYZE, SUMMARY off) SELECT * FROM pg_class;
```

```
Seq Scan on pg_class (cost=0.00..34.90 rows=418 width=265) (actual time=0.004..0.044 rows=418 loops=1)
```

```
EXPLAIN (ANALYZE, COSTS off, SUMMARY off) SELECT * FROM pg_class;
```

```
Seq Scan on pg_class (actual time=0.004..0.044 rows=418 loops=1)
```

Невыполнявшиеся узлы

```
EXPLAIN (ANALYZE, COSTS off, SUMMARY off)  
SELECT * FROM pg_class LIMIT 0;
```

```
Limit (actual time=0.001..0.001 rows=0 loops=1)
```

```
-> Seq Scan on pg_class (never executed)
```



узел не выполнялся

Время планирования

```
Seq Scan on pg_class (cost=0.00..34.90 rows=418 width=265)  
      (actual time=0.004..0.044 rows=418 loops=1)
```

```
Planning Time: 0.056 ms
```

```
Execution Time: 0.085 ms
```

время планирования сравнимо со временем исполнения – путь к prepared statements

PREPARE, **EXECUTE**, **DEALLOCATE**

<https://postgrespro.ru/docs/postgresql/15/sql-prepare>

Актуальность статистики

```
Seq Scan on pg_class (cost=0.00..34.90 rows=418 width=265)  
      (actual time=0.004..0.044 rows=418 loops=1)
```

```
Planning Time: 0.056 ms
```

```
Execution Time: 0.085 ms
```

сильная разница в любую сторону – неактуальная статистика

<https://postgrespro.ru/docs/postgresql/15/planner-stats>

Актуальность статистики

Устаревшая статистика

ANALYZE / VACUUM ANALYZE

<https://postgrespro.ru/docs/postgresql/15/sql-analyze>

<https://postgrespro.ru/docs/postgresql/15/sql-vacuum>

Актуальность статистики

ALTER TABLE ... SET (...)

autovacuum_analyze_threshold

“Задаёт минимальное число добавленных, изменённых или удалённых кортежей, при котором будет выполняться ANALYZE для отдельно взятой таблицы. Значение по умолчанию – 50 кортежей.”

<https://postgrespro.ru/docs/postgresql/15/runtime-config-autovacuum#GUC-AUTOVACUUM-ANALYZE-THRESHOLD>

autovacuum_analyze_scale_factor

“Задаёт процент от размера таблицы, который будет добавляться к autovacuum_analyze_threshold при выборе порога срабатывания команды ANALYZE. Значение по умолчанию – 0.1 (10% от размера таблицы).”

<https://postgrespro.ru/docs/postgresql/15/runtime-config-autovacuum#GUC-AUTOVACUUM-ANALYZE-SCALE-FACTOR>

Актуальность статистики

Недостаточная статистика

```
ALTER TABLE ... ALTER COLUMN ...
```

```
SET STATISTICS [0..10000]
```

<https://postgrespro.ru/docs/postgresql/15/sql-altertable#id-1.9.3.35.5>

```
CREATE STATISTICS
```

<https://postgrespro.ru/docs/postgresql/15/sql-createstatistics>

Время выполнения

```
Seq Scan on pg_class (cost=0.00..34.90 rows=418 width=265)  
      (actual time=0.004..0.044 rows=418 loops=1)
```

```
Planning Time: 0.056 ms
```

```
Execution Time: 0.085 ms
```

`loops * actual time` -> **полное время** выполнения

Сетевой трафик

```
Seq Scan on pg_class (cost=0.00..34.90 rows=418 width=265)  
      (actual time=0.004..0.044 rows=418 loops=1)
```

```
Planning Time: 0.056 ms
```

```
Execution Time: 0.085 ms
```

`loops * actual rows * width` -> размер **resultset**

Buffers

Объем прочитанных/записанных данных (страниц)

```
EXPLAIN (ANALYZE, BUFFERS) SELECT * FROM pg_class;
```

```
Seq Scan on pg_class (cost=0.00..34.90 rows=418 width=265) (actual time=0.017..0.109 rows=418 loops=1)
```

```
  Buffers: shared hit=14
```

```
Planning:
```

```
  Buffers: shared hit=107 read=12 written=11
```

```
  I/O Timings: shared/local read=18.118 write=12.037
```

```
Planning Time: 30.910 ms
```

```
Execution Time: 1.162 ms
```

$N \text{ (pages)} * 8KB \text{ (размер страницы)} = \text{физический объём}$

```
SHOW block_size;
```

Buffers

	hit из памяти получено	read с диска прочитано	dirtied в памяти «испачкано»	written на диск записано
local локальная память процесса	очень быстро	медленно	быстро	медленно
shared разделяемая память сервера	быстро	медленно	быстро	медленно
temp временные файлы	---	медленно	---	медленно

Buffers: **shared** **hit**=12673094, **local** **hit**=7 **read**=1 **dirtied**=1, **temp** **read**=11402 **written**=11402

Buffers

shared read – чтение с диска при отсутствии в кэше

не надо читать так много!

увеличить `shared_buffers`

<https://postgrespro.ru/docs/postgresql/15/runtime-config-resource#GUC-SHARED-BUFFERS>

увеличить память на сервере

Buffers

temp written – сброс на диск промежуточных выборок

не надо обрабатывать так много!

увеличить **work_mem**

<https://postgrespro.ru/docs/postgresql/15/runtime-config-resource#GUC-WORK-MEM>

вынести **temp_tablespaces** на RAMdrive

<https://postgrespro.ru/docs/postgresql/15/runtime-config-client#GUC-TEMP-TABLESPACES>

I/O Timings

Время чтения/записи на диск (ms)

```
SET track_io_timing = on; EXPLAIN (ANALYZE, BUFFERS) SELECT ...
```

```
Seq Scan on pg_class (cost=0.00..34.90 rows=418 width=265) (actual time=0.017..0.109 rows=418 loops=1)
```

```
  Buffers: shared hit=14
```

```
Planning:
```

```
  Buffers: shared hit=107 read=12 written=11
```

```
  I/O Timings: shared/local read=18.118 write=12.037
```

```
Planning Time: 30.910 ms
```

```
Execution Time: 1.162 ms
```

I/O Timings

	read с диска прочитано	write на диск записано
shared/local разделяемая память сервера	медленно	медленно
temp временные файлы	медленно	медленно

I/O Timings: **shared/local** read=1951.514, **temp** read=743.034 **write**=1302.919

I/O Timings

Buffers: shared hit=107 read=12 written=11

I/O Timings: shared/local read=18.118 write=12.037

Read = 12 * 8KB / 18.118ms = 5.17MB/s

Write = 11 * 8KB / 12.037ms = 7.14MB/s

Собственные показатели узла

```
EXPLAIN (ANALYZE, BUFFERS, COSTS off, SUMMARY off)  
SELECT * FROM pg_class LIMIT 1;
```

```
Limit (actual time=0.017..0.018 rows=1 loops=1) -- 0.018 x 1 - 0.016 x 1 = 0.002  
  Buffers: shared hit=1                        --          1 -          1 =      0  
-> Seq Scan on pg_class (actual time=0.015..0.016 rows=1 loops=1)  
    Buffers: shared hit=1
```

loops * actual time - subnodes -> собственное время

Buffers - subnodes -> собственные данные

I/O Timings - subnodes -> собственные IO-задержки

Операции в плане

Получение данных

`Result, ... Scan`

Построение битовых карт

`Bitmap...`

Вложенные запросы

`InitPlan, SubPlan`

Операции над множествами

`Append, Intersect, Except, SetOp, ...`

Обработка

`Limit, Sort, Incremental Sort, Unique`

Группировка

`Aggregate, Group, GroupAggregate, ...`

Соединения

`Nested Loop, Hash/Merge Join, ... Semi/Anti Join`

Параллелизм операций

`Parallel ..., Gather, Gather Merge`

Изменение данных

`Insert, Update, Delete, Merge`

Триггеры/Foreign Keys

`Trigger, Trigger for constraint`

Получение данных



Result

Константный результат

```
EXPLAIN SELECT 1;
```

```
Result (cost=0.00..0.05 rows=1 width=4)
```

Result

Вызов простой функции в поле

```
EXPLAIN SELECT random();
```

```
Result (cost=0.00..0.05 rows=1 width=8)
```


Result

Чтение из **однострочного VALUES**

```
EXPLAIN VALUES (1);
```

```
Result (cost=0.00..0.05 rows=1 width=4)
```

Values Scan

Чтение из **многострочного VALUES**

```
EXPLAIN VALUES (1), (2);
```

```
Values Scan on "*VALUES*" (cost=0.00..0.11 rows=2 width=4)
```



имя **источника данных**
(таблицы, функции, выборки)

Function Scan

Генерирующая функция во **FROM**


```
EXPLAIN SELECT * FROM generate_series(1, 4);
```

```
Function Scan on generate_series (cost=0.00..0.20 rows=4 width=4)
```

```
EXPLAIN SELECT * FROM generate_series(1, 4) i;
```

```
Function Scan on generate_series i (cost=0.00..0.20 rows=4 width=4)
```

↑
алиас



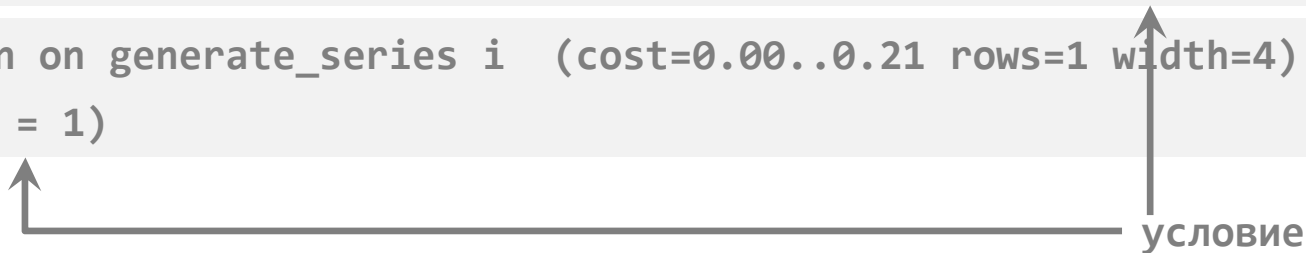
Filter

Условие фильтрации записей

```
EXPLAIN SELECT * FROM generate_series(1, 4) i WHERE i = 1;
```

```
Function Scan on generate_series i (cost=0.00..0.21 rows=1 width=4)
```

```
Filter: (i = 1)
```



Rows Removed by Filter

Количество записей, отброшенных по условию

```
EXPLAIN (ANALYZE, COSTS off, SUMMARY off)  
SELECT * FROM generate_series(1, 4) i WHERE i = 1;
```

```
Function Scan on generate_series i (actual time=0.009..0.010 rows=1 loops=1)  
  Filter: (i = 1)  
    Rows Removed by Filter: 3
```

ProjectSet

Генерирующая функция в поле

```
EXPLAIN SELECT generate_series(1, 4); -- Set-Returning Function
```

```
ProjectSet (cost=0.00..0.18 rows=4 width=4)  
-> Result (cost=0.00..0.05 rows=1 width=0)
```

```
EXPLAIN SELECT random(); -- не генерирующая, потому сразу Result
```

```
Result (cost=0.00..0.05 rows=1 width=8)
```

Subquery Scan

Обращение к значениям вложенной выборки

```
EXPLAIN SELECT * FROM (SELECT random() x) T WHERE x < 0.5;
```

```
Subquery Scan on t (cost=0.00..0.11 rows=1 width=8)
```

```
Filter: (t.x < '0.5'::double precision)
```

```
-> Result (cost=0.00..0.05 rows=1 width=8)
```

CTE/CTE Scan

Запрос с **WITH [MATERIALIZED]**

```
EXPLAIN WITH T AS MATERIALIZED (VALUES (1), (2)) TABLE T;
```

```
CTE Scan on t (cost=0.11..0.31 rows=2 width=4)
```

```
CTE t
```

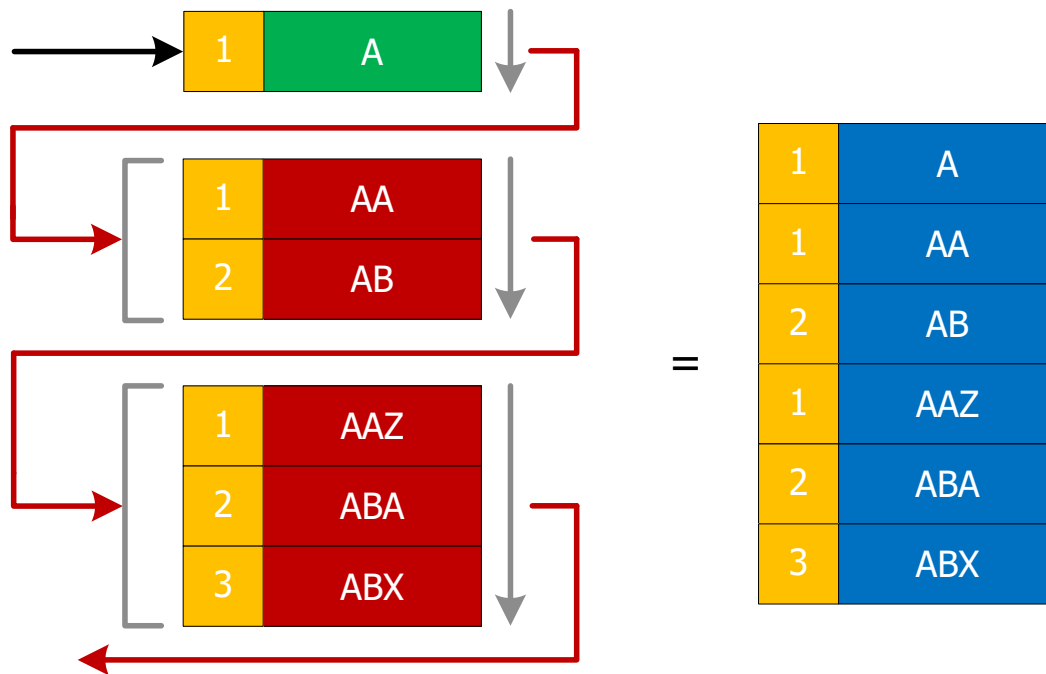
```
-> Values Scan on "*VALUES*" (cost=0.00..0.11 rows=2 width=4)
```

```
EXPLAIN WITH T AS (VALUES (1), (2)) TABLE T; -- однократное чтение
```

```
Values Scan on "*VALUES*" (cost=0.00..0.11 rows=2 width=4)
```


Recursive Union/workTable Scan

Запрос с **WITH RECURSIVE**



Recursive Union/WorkTable Scan

Запрос с **WITH RECURSIVE**

```
CTE Scan on fib (cost=13.63..16.73 rows=31 width=12)
```

```
CTE fib
```

```
-> Recursive Union (cost=0.00..13.63 rows=31 width=12)
```

```
  -> Result (cost=0.00..0.05 rows=1 width=12)
```

```
  -> WorkTable Scan on fib fib_1 (cost=0.00..1.05 rows=3 width=12)
```

```
      Filter: (i < 10)
```

Recursive Union/workTable Scan

Запрос с **WITH RECURSIVE**

```
EXPLAIN
WITH RECURSIVE fib(i, a, b) AS (
  VALUES(0, 0, 1) -- затравка
UNION ALL
  SELECT          -- шаг рекурсии
    i + 1
  , greatest(a, b)
  , a + b
FROM
  fib             -- обращение к себе
WHERE
  i < 10          -- условие продолжения
)
TABLE fib;
```

InitPlan

Вложенный запрос, **не зависящий** ни от чего

```
EXPLAIN SELECT (SELECT random()) * i FROM generate_series(1, 4) i;
```

```
Function Scan on generate_series i (cost=0.06..0.26 rows=4 width=8)
```

```
  InitPlan 1 (returns $0)
```

```
    -> Result (cost=0.00..0.05 rows=1 width=8)
```

SubPlan

Вложенный запрос, **зависящий** от других

```
EXPLAIN SELECT (SELECT random() * i) FROM generate_series(1, 4) i;
```

```
Function Scan on generate_series i (cost=0.00..0.43 rows=4 width=8)
```

```
SubPlan 1
```

```
-> Result (cost=0.00..0.06 rows=1 width=8)
```

Разница **InitPlan/SubPlan**

```
SELECT (SELECT random()) * i FROM generate_series(1, 4) i;
```

0.10015162491382235 -- x1

0.2003032498276447 -- x2

0.30045487474146704 -- x3

0.4006064996552894 -- x4 – размножилось **одно** значение

```
SELECT (SELECT random() * i) FROM generate_series(1, 4) i;
```

0.6490647188923726

0.6605839339960129

0.5961901827957088 -- **ничего общего**

1.476483188079107 -- все значения вычислены **независимо**

Разница **InitPlan**/**SubPlan**

```
EXPLAIN (ANALYZE, ...) SELECT (SELECT random()) * i FROM generate_series(1, 4) i;
```

Function Scan on generate_series i (actual time=0.010..0.011 rows=4 loops=1)

InitPlan 1 (returns \$0)

-> Result (actual time=0.001..0.001 rows=1 loops=1)

```
EXPLAIN (ANALYZE, ...) SELECT (SELECT random() * i) FROM generate_series(1, 4) i;
```

Function Scan on generate_series i (actual time=0.009..0.012 rows=4 loops=1)

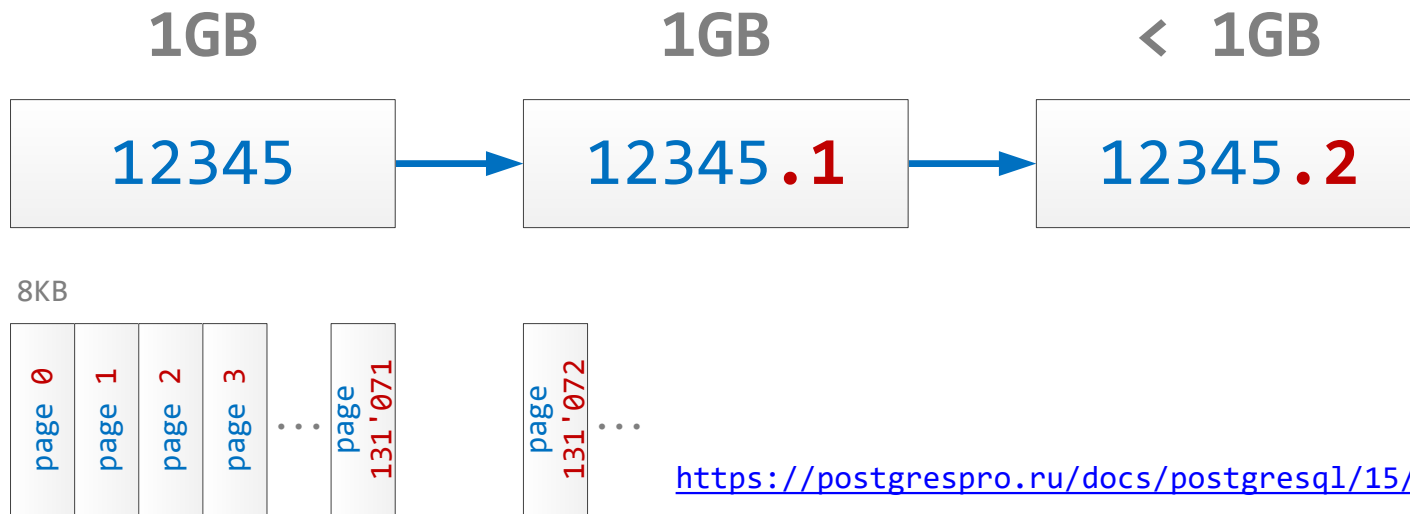
SubPlan 1

-> Result (actual time=0.000..0.001 rows=1 loops=4) -- по каждой записи

Физическое хранение данных

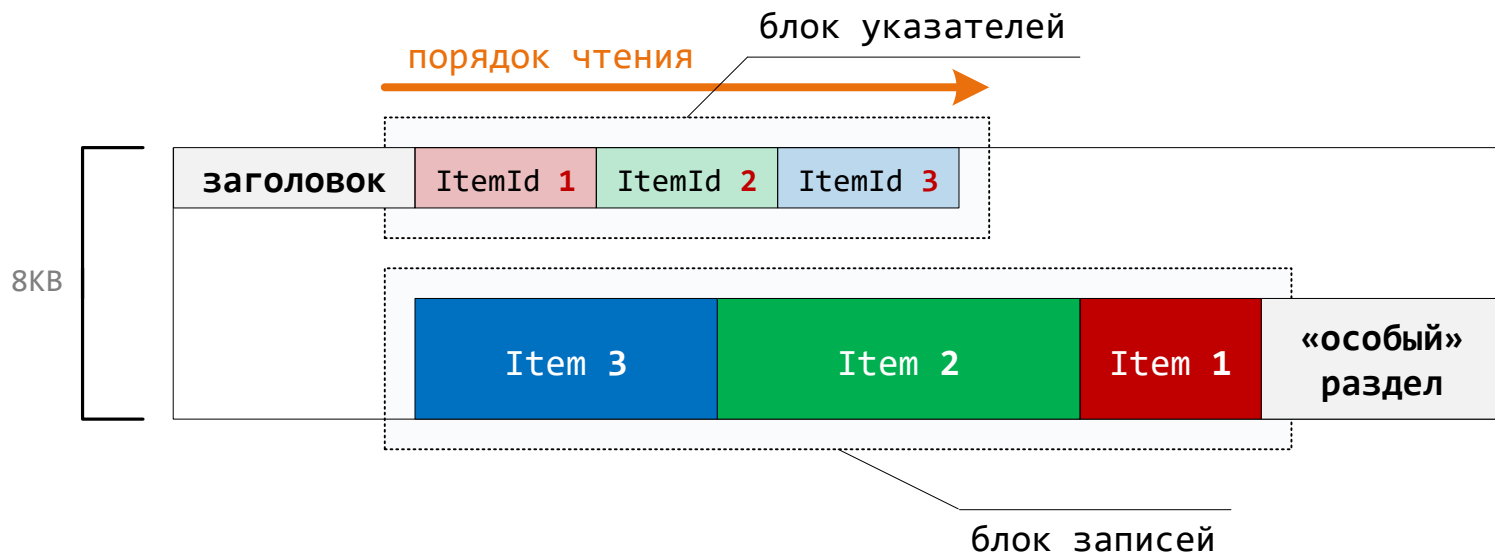
Файл таблицы/индекса разделен на сегменты по **1GB**

каждый сегмент – на страницы по **8KB**



Физическое хранение данных

Каждая страница содержит список кортежей



<https://postgrespro.ru/docs/postgresql/15/storage-page-layout>

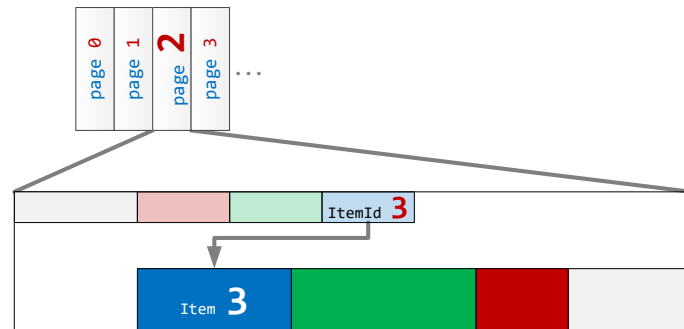
TID Scan

Чтение конкретной записи по **физическому** адресу

```
EXPLAIN SELECT * FROM pg_class WHERE ctid = '(2,3)';
```

```
Tid Scan on pg_class (cost=0.00..1.15 rows=1 width=265)
```

```
TID Cond: (ctid = '(2,3)::tid)
```



TID Scan

Чтение **конкретной записи** по физическому адресу

идентификация записей **таблицы без Primary Key**

«цепочка» запросов по одним и тем же строкам

например, удаление дублей строк из таблицы

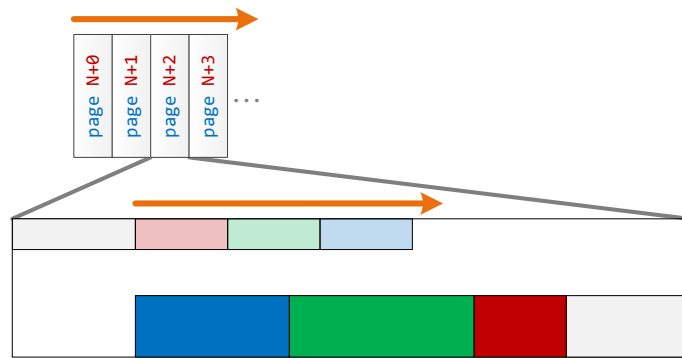
<https://habr.com/ru/companies/tensor/articles/481352/>

Seq Scan

Последовательный просмотр страниц и их записей

```
EXPLAIN SELECT * FROM pg_class;
```

```
Seq Scan on pg_class (cost=0.00..34.90 rows=418 width=265)
```



Seq Scan

Последовательный просмотр страниц и их записей

самый простой, а потому **самый быстрый** способ

оптимален **для небольших таблиц** (сотни строк)

... или когда иначе никак (нет нужных индексов)

Index Scan [Backward]

Индексированный поиск в таблице

```
EXPLAIN SELECT * FROM pg_class WHERE relname = 'pg_class';
```

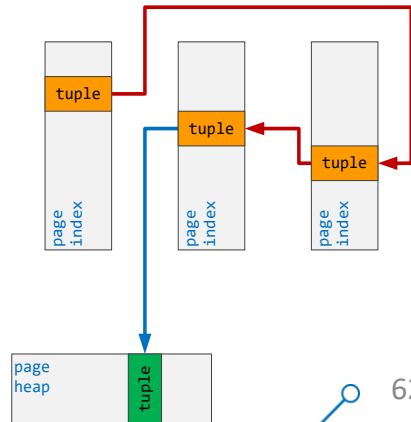
Index Scan using pg_class_relname_nsp_index on pg_class (cost=0.27..2.53 rows=1 width=265)

Index Cond: (relname = 'pg_class'::name)

↑
имя **индекса**

↑
имя **таблицы**

Index Cond – ключ индексного поиска



Index Scan [Backward]

Индексированный поиск в таблице

подходящий к условиям индекс PostgreSQL подбирает сам

спускаемся по «дереву» индекса, затем идем в страницу данных

не стоит читать много-много записей (**random read!**)

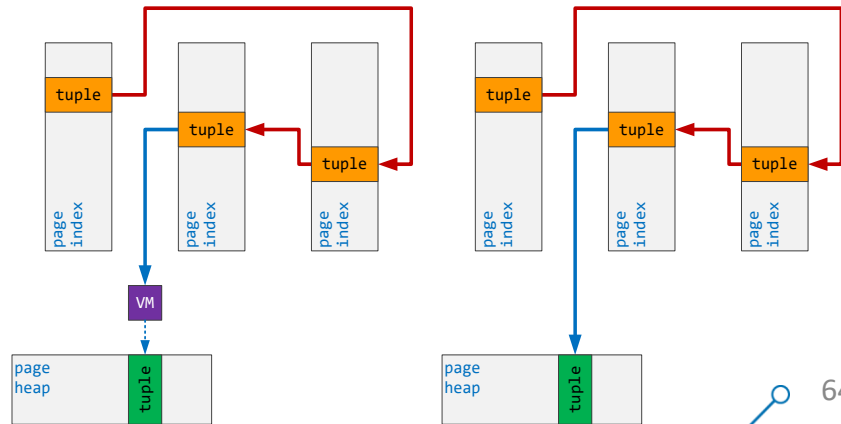
Index Only Scan [Backward]

Чтение из индекса, без обращения к таблице

```
EXPLAIN SELECT relname FROM pg_class WHERE relname = 'pg_class';
```

Index Only Scan using pg_class_relname_nsp_index on pg_class (cost=0.27..1.43 rows=1 width=64)

Index Cond: (relname = 'pg_class'::name)



Index Only Scan [Backward]

Чтение из индекса, без обращения к таблице

получение только ключевых/включенных полей индекса

```
CREATE INDEX ON tbl(a, b, c) INCLUDE (x, y, z)
```

<https://postgrespro.ru/docs/postgresql/15/indexes-index-only-scans>

Heap Fetches

Строк получено из таблицы при **IOS**

когда их много, **IOS** проиграет «обычному» **IS**

<https://habr.com/ru/companies/tensor/articles/751458/>

```
EXPLAIN (ANALYZE, ...) SELECT relname FROM pg_class WHERE relname = 'pg_class';
```

```
Index Only Scan using pg_class_relname_nsp_index on pg_class (actual time=0.014..0.014 rows=1 loops=1)
```

```
Index Cond: (relname = 'pg_class'::name)
```

```
Heap Fetches: 1
```

Order By

Индексное упорядочивание (\leftrightarrow , k-NN search)

если его поддерживает тип индекса и класс оператора

<https://postgrespro.ru/docs/postgresql/15/indexes-types#INDEXES-TYPE-GIST>

GiST

<https://postgrespro.ru/docs/postgresql/15/gist-builtin-opclasses#GIST-BUILTIN-OPCLASSES-TABLE>

SP-GiST

<https://postgrespro.ru/docs/postgresql/15/spgist-builtin-opclasses#SPGIST-BUILTIN-OPCLASSES-TABLE>

Order By

Индексное упорядочивание (k-NN search)

```
CREATE TABLE places AS
  SELECT
    point(random(), random()) p
  FROM
    generate_series(1, 1e6) i; -- миллион случайных точек [0..1, 0..1]
CREATE INDEX ON places USING gist(p); -- GiST-индекс
EXPLAIN (ANALYZE, ...) SELECT * FROM places ORDER BY p <-> '(0.5,0.5)::point LIMIT 10;

Limit (actual time=0.324..0.333 rows=10 loops=1)
-> Index Only Scan using places_p_idx on places (actual time=0.323..0.331 rows=10 loops=1)
   Order By: (p <-> '(0.5,0.5)::point)
   Heap Fetches: 0
```

Sample Scan

Чтение «примера» из таблицы

```
EXPLAIN SELECT * FROM pg_class TABLESAMPLE BERNOULLI(1);
```

```
Sample Scan on pg_class (cost=0.00..14.20 rows=4 width=265)  
Sampling: bernoulli ('1'::real)
```

Sampling – правила отбора записей

Sample Scan

Чтение «примера» из таблицы

быстрый аналог **WHERE random()** < **X**

TABLESAMPLE *метод_выборки* (*аргумент* [, ...]) [**REPEATABLE** (*затравка*)]

“Методы выборки **BERNOULLI** и **SYSTEM** принимают единственный аргумент, определяющий, какой процент таблицы должен попасть в выборку, от **0** до **100**. Этот аргумент может задаваться любым выражением со значением типа `real`. (Другие методы выборки могут принимать дополнительные или другие параметры.) Оба этих метода возвращают случайную выборку таблицы, содержащую примерно указанный процент строк таблицы. Метод **BERNOULLI** сканирует всю таблицу и выбирает или игнорирует **отдельные строки независимо**, с заданной вероятностью. Метод **SYSTEM** строит выборку на уровне блоков, определяя **для каждого блока** шанс его задействовать, и возвращает все строки из каждого задействуемого блока.”

<https://postgrespro.ru/docs/postgresql/15/sql-select#SQL-FROM>

Table Function Scan

Выполнение функции **XMLTABLE**

<https://postgrespro.ru/docs/postgresql/15/functions-xml#FUNCTIONS-XML-PROCESSING-XMLTABLE>

```
EXPLAIN SELECT * FROM XMLTABLE('/' PASSING '<xml/>' COLUMNS id integer);
```

```
Table Function Scan on "xmltable" (cost=0.00..5.00 rows=100 width=4)
```

Foreign Scan

Обращение к стороннему серверу

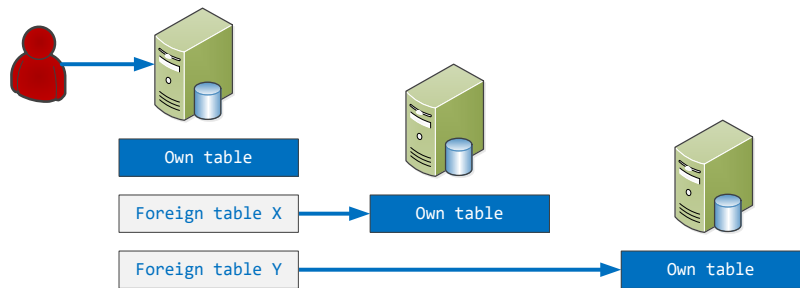
```
EXPLAIN VERBOSE SELECT * FROM foreign_pg_class;
```

Foreign Scan on public.foreign_pg_class (cost=100.00..207.90 rows=890 width=68)

Output: oid, relname

Remote SQL: SELECT oid, relname FROM pg_catalog.pg_class

Remote SQL – запрос к целевому серверу



Foreign Scan

Обращение к стороннему серверу

масштабирование и распределение нагрузки

<https://postgrespro.ru/docs/postgresql/15/sql-createforeigntable>

```
CREATE EXTENSION postgres_fdw; -- устанавливаем расширение для доступа к внешнему PostgreSQL-серверу
CREATE SERVER foreign_server
    FOREIGN DATA WRAPPER postgres_fdw
    OPTIONS (host 'localhost', dbname '_test'); -- определяем координаты внешнего сервера
CREATE USER MAPPING FOR PUBLIC
    SERVER foreign_server
    OPTIONS (password 'postgres'); -- пробрасываем доступ пользователям
```

Foreign Scan

Обращение к стороннему серверу

масштабирование и распределение нагрузки

```
CREATE FOREIGN TABLE foreign_pg_class(  
    oid oid  
    , relname name  
    ) -- имя и формат «локальной» прокси-таблицы  
    SERVER foreign_server  
    OPTIONS (  
        schema_name 'pg_catalog'  
        , table_name 'pg_class'  
    ); -- схема и имя таблицы на внешнем сервере
```

Foreign Scan

Обращение к стороннему серверу

перенос соединений (JOIN) на внешний сервер

```
EXPLAIN SELECT * FROM  
  foreign_pg_class c JOIN foreign_pg_index i  
    ON i.indexrelid = c.oid  
LIMIT 1;
```

```
Foreign Scan (cost=100.00..100.55 rows=1 width=76)  
  Relations: (public.foreign_pg_class c) INNER JOIN (public.foreign_pg_index i)
```

Relations – соединение на внешнем сервере

Foreign Scan

Обращение к стороннему серверу

```
CREATE FOREIGN TABLE foreign_pg_index(indexrelid oid, indrelid oid)  
  SERVER foreign_server  
  OPTIONS (schema_name 'pg_catalog', table_name 'pg_index');
```

Async Foreign Scan

Обращение к стороннему серверу

```
EXPLAIN SELECT * FROM proxy_pg_class;
```

```
Append (cost=0.00..460.33 rows=1781 width=68)
```

- > Seq Scan on proxy_pg_class proxy_pg_class_1 (cost=0.00..0.00 rows=1 width=68)
- > Async Foreign Scan on foreign_1_pg_class proxy_pg_class_2 (cost=100.00..207.90 rows=890 width=68)
- > Async Foreign Scan on foreign_2_pg_class proxy_pg_class_3 (cost=100.00..207.90 rows=890 width=68)

Async Foreign Scan

Обращение к стороннему серверу

```
CREATE TABLE proxy_pg_class(oid oid, relname name);           -- имя и формат локальной таблицы
CREATE SERVER foreign_server_1
  FOREIGN DATA WRAPPER postgres_fdw
  OPTIONS (host 'localhost', dbname '_test', async_capable 'true'); -- включаем асинхронность
CREATE USER MAPPING FOR PUBLIC
  SERVER foreign_server_1
  OPTIONS (password 'postgres');
CREATE FOREIGN TABLE foreign_1_pg_class()
  INHERITS (proxy_pg_class)                                     -- наследуемся от локальной таблицы
  SERVER foreign_server_1
  OPTIONS (schema_name 'pg_catalog', table_name 'pg_class');
-- повторить для foreign_server_2 / foreign_2_pg_class
```

Async Foreign Scan

Обращение к стороннему серверу

```
EXPLAIN (ANALYZE, VERBOSE, ...) SELECT * FROM proxy_pg_class LIMIT 1;
```

```
Limit (actual time=4.851..4.853 rows=1 loops=1)
```

```
Output: proxy_pg_class.oid, proxy_pg_class.relname
```

```
-> Append (actual time=4.849..4.851 rows=1 loops=1)
```

```
-> Seq Scan on public.proxy_pg_class proxy_pg_class_1 (actual time=0.004..0.004 rows=0 loops=1)
```

```
Output: proxy_pg_class_1.oid, proxy_pg_class_1.relname
```

```
-> Async Foreign Scan on public.foreign_1_pg_class proxy_pg_class_2 (actual time=4.086..4.087 rows=1 loops=1)
```

```
Output: proxy_pg_class_2.oid, proxy_pg_class_2.relname
```

```
Remote SQL: SELECT oid, relname FROM pg_catalog.pg_class
```

```
-> Async Foreign Scan on public.foreign_2_pg_class proxy_pg_class_3 (actual time=0.750..0.750 rows=0 loops=1)
```

```
Output: proxy_pg_class_3.oid, proxy_pg_class_3.relname
```

```
Remote SQL: SELECT oid, relname FROM pg_catalog.pg_class
```

Named Tuplestore Scan

Обращение к записям внутри триггера

NEW/OLD внутри **STATEMENT**-триггера

<https://postgrespro.ru/docs/postgresql/15/sql-createtrigger>

```
Result (actual time=0.007..0.007 rows=1 loops=1)
```

```
  InitPlan 1 (returns $0)
```

```
    -> Aggregate (actual time=0.004..0.005 rows=1 loops=1)
```

```
      -> Named Tuplestore Scan (actual time=0.001..0.001 rows=2 loops=1)
```


Named Tuplestore Scan

Обращение к записям внутри триггера

```
CREATE TABLE tbl(i integer);

CREATE OR REPLACE FUNCTION count_trigger() RETURNS trigger AS $$
BEGIN
    RAISE NOTICE 'count = %', (SELECT count(*) FROM new_ref); -- вывели в консоль количество вставляемых записей
    RETURN NULL;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER tbl_trigger AFTER INSERT ON tbl
    REFERENCING NEW TABLE AS new_ref -- "пробросили" NEW внутрь триггера под именем new_ref
    FOR EACH STATEMENT                -- только для STATEMENT-триггеров
    EXECUTE PROCEDURE count_trigger();
```

Named Tuplestore Scan

Обращение к записям внутри триггера

перехватываем планы внутри триггеров

<https://postgrespro.ru/docs/postgresql/15/auto-explain>

```
LOAD 'auto_explain';
SET auto_explain.log_analyze = 'on';
SET auto_explain.log_buffers = 'on';
SET auto_explain.log_timing = 'on';
SET auto_explain.log_min_duration = 0;          -- снимаем планы вообще всех запросов
SET auto_explain.log_nested_statements = 'on';  -- ... и вложенных - тоже
SET auto_explain.log_triggers = 'on';
```

Named Tuplstore Scan

Обращение к записям внутри триггера

```
INSERT INTO tbl VALUES (1), (2);
```

```
NOTICE: count = 2
```

```
Query returned successfully: 2 rows affected, 13 msec execution time.
```

```
Result (cost=0.26..0.30 rows=1 width=8) (actual time=0.007..0.007 rows=1 loops=1)
```

```
  InitPlan 1 (returns $0)
```

```
    -> Aggregate (cost=0.21..0.26 rows=1 width=8) (actual time=0.004..0.005 rows=1 loops=1)
```

```
      -> Named Tuplstore Scan (cost=0.00..0.20 rows=2 width=0) (actual time=0.001..0.001 rows=2 loops=1)
```

Custom Scan

Пользовательский метод доступа к данным

```
Custom Scan (HypertableModify)
```

```
-> Insert on _materialized_hypertable_15
```

```
-> Custom Scan (ChunkDispatch)
```

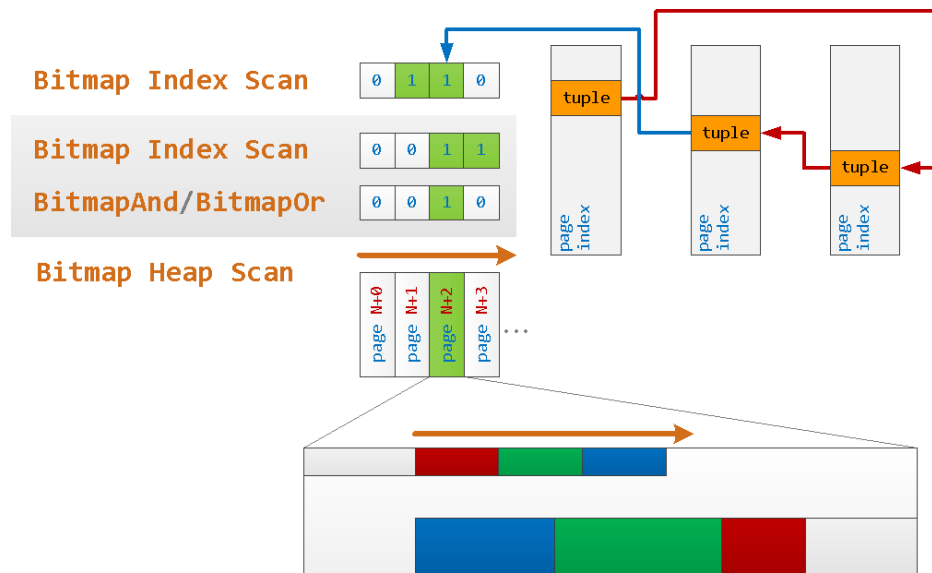
```
...
```

↑
пользовательский метод-провайдер

Работа с битовыми картами

Последовательный просмотр страниц по маске

можно комбинировать несколько индексов/условий



Работа с битовыми картами

Последовательный просмотр страниц по маске

```
EXPLAIN SELECT * FROM pg_class WHERE relname = 'pg_class' OR relname = 'pg_index';
```

```
Bitmap Heap Scan on pg_class (cost=2.76..5.00 rows=2 width=265)
```

```
  Recheck Cond: ((relname = 'pg_class'::name) OR (relname = 'pg_index'::name))
```

```
    -> BitmapOr (cost=2.76..2.76 rows=2 width=0)
```

```
      -> Bitmap Index Scan on pg_class_relname_nsp_index (cost=0.00..1.38 rows=1 width=0)
```

```
        Index Cond: (relname = 'pg_class'::name)
```

```
      -> Bitmap Index Scan on pg_class_relname_nsp_index (cost=0.00..1.38 rows=1 width=0)
```

```
        Index Cond: (relname = 'pg_index'::name)
```

Recheck Cond – перепроверка индексного условия

Работа с битовыми картами

Объединение/пересечение индексных условий

```
CREATE TABLE bitmap_tst AS  
  SELECT random() FROM generate_series(1, 1e6); -- миллион строк  
CREATE INDEX ON bitmap_tst(random);
```

```
EXPLAIN (ANALYZE, COSTS off, TIMING off, SUMMARY off)  
  SELECT * FROM bitmap_tst WHERE random < 0.01 OR random > 0.99;
```

Rows Removed by Index Recheck

Количество отброшенных перепроверкой записей

```
Bitmap Heap Scan on bitmap_tst (actual rows=20107 loops=1)
```

```
Recheck Cond: ((random < '0.01'::double precision) OR (random > '0.99'::double precision))
```

```
Rows Removed by Index Recheck: 801273
```

```
Heap Blocks: exact=752 lossy=3624
```

```
-> BitmapOr (actual rows=0 loops=1)
```

```
    -> Bitmap Index Scan on bitmap_tst_random_idx (actual rows=9989 loops=1)
```

```
        Index Cond: (random < '0.01'::double precision)
```

```
    -> Bitmap Index Scan on bitmap_tst_random_idx (actual rows=10118 loops=1)
```

```
        Index Cond: (random > '0.99'::double precision)
```


Heap Blocks

Количество прочитанных из таблицы блоков

```
Bitmap Heap Scan on bitmap_tst (actual rows=20107 loops=1)
```

```
...
```

```
Heap Blocks: exact=752 lossy=3624
```

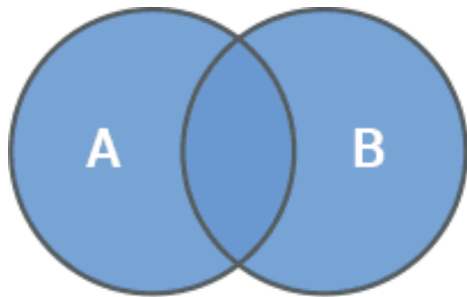
```
...
```

exact – в карте записан адрес до конкретной записи **(page, tuple)**

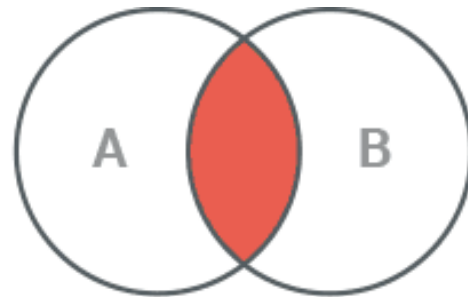
lossy – не хватило **work_mem**, поэтому в карте только **(page)**

каждую запись страницы приходится вычитывать и перепроверять

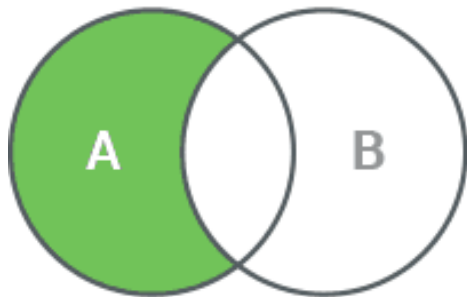
Операции над множествами



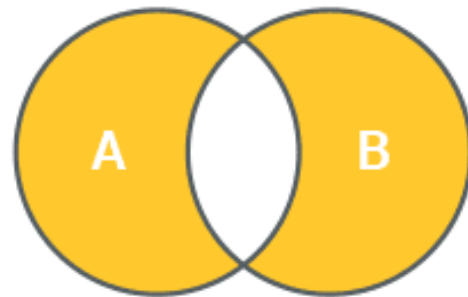
Union



Intersection



Difference



Symmetric Difference

Append

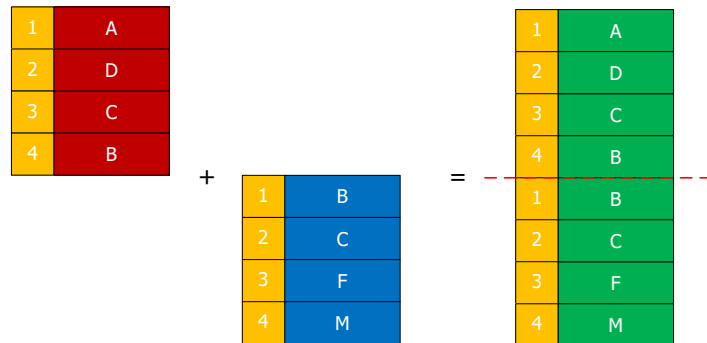
Объединение выборок

```
EXPLAIN (VALUES (1), (2), (3)) UNION ALL (VALUES (1), (2));
```

Append (cost=0.00..0.39 rows=5 width=4) -- **сумма**

-> Values Scan on "*VALUES*" (cost=0.00..0.16 rows=3 width=4)

-> Values Scan on "*VALUES*_1" (cost=0.00..0.11 rows=2 width=4)



Append + Unique

Объединение простых выборок с уникализацией

```
EXPLAIN (VALUES (1), (2), (3)) UNION (VALUES (1), (2));
```

```
Unique (cost=0.70..0.72 rows=5 width=4) -- сумма, а по факту будет 3 – погрешность
-> Sort (cost=0.70..0.71 rows=5 width=4)
    Sort Key: "*VALUES*".column1
    -> Append (cost=0.00..0.64 rows=5 width=4) -- сумма
        -> Values Scan on "*VALUES*" (cost=0.00..0.16 rows=3 width=4)
        -> Values Scan on "*VALUES*_1" (cost=0.00..0.11 rows=2 width=4)
```

Sort Key – набор выражений сортировки

Append + HashAggregate

Объединение сложных выборок с уникализацией

```
EXPLAIN (COSTS off) (TABLE pg_class) UNION (TABLE pg_class);
```

HashAggregate

Group Key: pg_class.oid, pg_class.relname, ...

-> **Append**

-> Seq Scan on pg_class

-> Seq Scan on pg_class pg_class_1

Group Key – набор выражений группировки

Merge Append

Объединение сортированных выборок

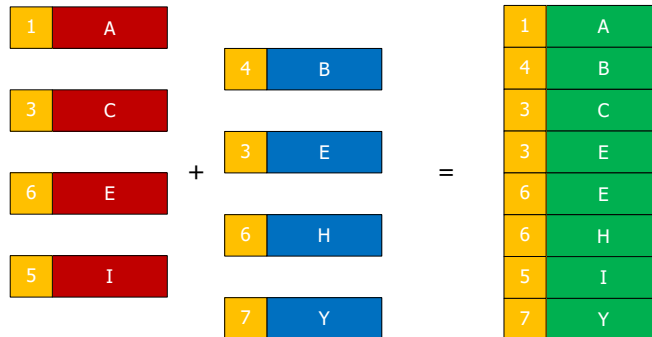
```
EXPLAIN (COSTS off) (TABLE pg_class) UNION ALL (TABLE pg_class) ORDER BY relname;
```

Merge Append

Sort Key: pg_class.relname

-> **Index Scan** using pg_class_relname_nsp_index on pg_class

-> **Index Scan** using pg_class_relname_nsp_index on pg_class pg_class_1



SetOp Intersect/Except [All]

Пересечение/исключение выборок

```
EXPLAIN (VALUES (1), (2), (3)) INTERSECT ALL (VALUES (1), (2));
```

```
SetOp Intersect All (cost=0.70..0.72 rows=2 width=8) -- минимум
```

```
-> Sort (cost=0.70..0.71 rows=5 width=8)
```

```
Sort Key: "*SELECT* 2".column1
```

```
-> Append (cost=0.00..0.64 rows=5 width=8) -- сумма
```

```
-> Subquery Scan on "*SELECT* 2" (cost=0.00..0.21 rows=2 width=8)
```

```
-> Values Scan on "*VALUES*" (cost=0.00..0.11 rows=2 width=4)
```

```
-> Subquery Scan on "*SELECT* 1" (cost=0.00..0.31 rows=3 width=8)
```

```
-> Values Scan on "*VALUES*_1" (cost=0.00..0.16 rows=3 width=4)
```

SetOp Intersect/Except [A11]

Пересечение/исключение выборок

```
EXPLAIN (VALUES (1), (2), (3)) INTERSECT (VALUES (1), (2));
```

```
SetOp Intersect (cost=0.70..0.72 rows=2 width=8) -- минимум
```

```
-> Sort (cost=0.70..0.71 rows=5 width=8)
```

```
Sort Key: "*SELECT* 2".column1
```

```
-> Append (cost=0.00..0.64 rows=5 width=8) -- сумма
```

```
-> Subquery Scan on "*SELECT* 2" (cost=0.00..0.21 rows=2 width=8)
```

```
-> Values Scan on "*VALUES*" (cost=0.00..0.11 rows=2 width=4)
```

```
-> Subquery Scan on "*SELECT* 1" (cost=0.00..0.31 rows=3 width=8)
```

```
-> Values Scan on "*VALUES*_1" (cost=0.00..0.16 rows=3 width=4)
```


HashSetOp Intersect/Except [All]

Пересечение/исключение выборок

```
EXPLAIN (COSTS off) (TABLE pg_class) INTERSECT ALL (TABLE pg_class);
```

HashSetOp Intersect All

-> Append

-> Subquery Scan on "*SELECT* 1"

-> Seq Scan on pg_class

-> Subquery Scan on "*SELECT* 2"

-> Seq Scan on pg_class pg_class_1

Обработка данных



Limit

LIMIT – ограничение выборки

```
EXPLAIN (COSTS off) VALUES (1), (2), (3) LIMIT 1;
```

```
Limit (cost=0.00..0.05 rows=1 width=4)
```

```
-> Values Scan on "*VALUES*" (cost=0.00..0.16 rows=3 width=4)
```

Sort

ORDER BY – сортировка

```
EXPLAIN (COSTS off) VALUES (1), (2), (3) ORDER BY 1;
```

Sort

Sort Key: column1

-> Values Scan on "*VALUES*"

Sort Key – набор выражений сортировки

Sort

ORDER BY – сортировка

```
EXPLAIN (ANALYZE, COSTS off, TIMING off, SUMMARY off)  
VALUES (1), (2), (3) ORDER BY 1;
```

Sort (actual rows=3 loops=1)

Sort Key: column1

Sort Method: quicksort Memory: 25kB

-> Values Scan on "**VALUES*" (actual rows=3 loops=1)

Sort Method – алгоритм (quicksort/TOP-N) и статистика сортировки

Sort исчезающий

ORDER BY – сортировка по индексу

```
EXPLAIN (ANALYZE, COSTS off, TIMING off, SUMMARY off)
```

```
TABLE pg_class ORDER BY relname, relnamespace LIMIT 1;
```

```
Limit (actual rows=1 loops=1) -- прочитали ровно сколько просили
```

```
-> Index Scan using pg_class_relname_nsp_index on pg_class (actual rows=1 loops=1)  
    -- pg_class(relname, relnamespace)
```

Incremental Sort

Нарастающая сортировка по индексу

```
EXPLAIN (COSTS off) TABLE pg_class ORDER BY relname, oid LIMIT 1;
```

Limit

-> Incremental Sort

Sort Key: relname, oid

Presorted Key: relname

-> Index Scan using pg_class_relname_nsp_index on pg_class

-- pg_class(relname, relnamespace)

Presorted Key – предварительно отсортированный набор ключей

Incremental Sort

Нарастающая сортировка по индексу

```
EXPLAIN (ANALYZE, COSTS off, TIMING off, SUMMARY off)  
TABLE pg_class ORDER BY relname, oid LIMIT 1;
```

Limit (actual rows=1 loops=1)

-> Incremental Sort (actual rows=1 loops=1) -- а прочитали +1, чтобы «закончить» группу

Sort Key: relname, oid

Presorted Key: relname

Full-sort Groups: 1 Sort Method: quicksort Average Memory: 25kB Peak Memory: 25kB

-> Index Scan using pg_class_relname_nsp_index on pg_class (actual rows=2 loops=1)

Full-sort Groups – алгоритм и статистика предсортировки

Unique

DISTINCT – уникализация

```
EXPLAIN SELECT DISTINCT ON(i) * FROM (VALUES (2),(1),(1),(2)) T(i);
```

```
Unique (cost=0.25..0.27 rows=4 width=4)
```

```
-> Sort (cost=0.25..0.26 rows=4 width=4)
```

```
Sort Key: "*VALUES*".column1
```

```
-> Values Scan on "*VALUES*" (cost=0.00..0.21 rows=4 width=4)
```

Unique

DISTINCT – уникализация по индексу

```
EXPLAIN (COSTS off) SELECT DISTINCT ON(relname) * FROM pg_class;
```

Unique

```
-> Index Scan using pg_class_relname_nsp_index on pg_class
```

Unique + Append

UNION [DISTINCT] – уникализация объединения

```
EXPLAIN (VALUES (1), (2), (3)) UNION DISTINCT (VALUES (1), (2));
```

```
Unique (cost=0.70..0.72 rows=5 width=4)
```

```
-> Sort (cost=0.70..0.71 rows=5 width=4)
```

```
Sort Key: "*VALUES*".column1
```

```
-> Append (cost=0.00..0.64 rows=5 width=4)
```

```
-> Values Scan on "*VALUES*" (cost=0.00..0.16 rows=3 width=4)
```

```
-> Values Scan on "*VALUES*_1" (cost=0.00..0.11 rows=2 width=4)
```

Группировка



Group

GROUP BY – группировка

```
EXPLAIN SELECT * FROM (VALUES (2),(1),(1),(2)) GROUP BY 1;
```

```
Group (cost=0.25..0.27 rows=4 width=4) -- не знает, сколько останется
```

```
Group Key: "*VALUES*".column1
```

```
-> Sort (cost=0.25..0.26 rows=4 width=4)
```

```
Sort Key: "*VALUES*".column1
```

```
-> Values Scan on "*VALUES*" (cost=0.00..0.21 rows=4 width=4)
```

Group Key – набор выражений группировки

Group

GROUP BY – группировка по индексу

```
EXPLAIN (COSTS off) SELECT relname FROM pg_class GROUP BY 1;
```

Group

Group Key: relname

-> **Index Only Scan** using pg_class_relname_nsp_index on pg_class

Aggregate

Вычисление агрегатных функций

```
EXPLAIN (COSTS off) SELECT count(*) FROM pg_class;
```

Aggregate

-> Index Only Scan using pg_class_tblspc_relfilenode_index on pg_class

GroupAggregate

Вычисление агрегатных функций при группировке

```
EXPLAIN (COSTS off) SELECT relname, count(*) FROM pg_class GROUP BY 1;
```

GroupAggregate

Group Key: relname -- префикс индекса

-> Index Only Scan using pg_class_relname_nsp_index on pg_class

GroupAggregate

Вычисление агрегатных функций при группировке

```
EXPLAIN (COSTS off) SELECT relname, count(*) FROM pg_class GROUP BY CUBE(1);
```

GroupAggregate

Group Key: relname -- сразу несколько вариантов группировок

Group Key: ()

-> Index Only Scan using pg_class_relname_nsp_index on pg_class

HashAggregate

Вычисление агрегатных функций при группировке

```
EXPLAIN (COSTS off) SELECT relnamespace, count(*) FROM pg_class GROUP BY 1;
```

HashAggregate

Group Key: relnamespace -- не префикс

-> Index Only Scan using pg_class_relname_nsp_index on pg_class

MixedAggregate

Вычисление агрегатных функций при группировке

```
EXPLAIN (COSTS off) SELECT relname, relnamespace, count(*) FROM pg_class GROUP BY CUBE(1, 2);
```

MixedAggregate

Hash Key: relnamespace

Group Key: relname, relnamespace

Group Key: relname

Group Key: ()

-> Index Only Scan using pg_class_relname_nsp_index on pg_class

Hash Key – набор выражений хэширования

WindowAgg

Вычисление оконных функций

```
EXPLAIN (COSTS off) SELECT count(*) OVER() FROM pg_class;
```

WindowAgg

```
-> Index Only Scan using pg_class_tblspc_relfilenode_index on pg_class
```

Блокировка записей



LockRows

Принудительная **FOR**-блокировка записей

```
EXPLAIN (COSTS off) SELECT * FROM pg_class WHERE relname = 'pg_class' FOR UPDATE;
```

LockRows

```
-> Index Scan using pg_class_relname_nsp_index on pg_class  
    Index Cond: (relname = 'pg_class'::name)
```

Соединения



Nested Loop

Для каждой записи «слева» выполняем узел «справа»

```
EXPLAIN (ANALYZE, COSTS off, TIMING off, SUMMARY off)  
  SELECT * FROM generate_series(1, 4) i, generate_series(1, i) j;
```

Nested Loop (actual rows=10 loops=1)

- > Function Scan on generate_series i (actual rows=4 loops=1)
- > Function Scan on generate_series j (actual rows=2 loops=4) -- «средние» rows

Nested Loop Left Join – при LEFT/RIGHT JOIN

Join Filter

Фильтруем записи по ходу соединения

```
EXPLAIN (ANALYZE, COSTS off, TIMING off, SUMMARY off)
```

```
SELECT * FROM generate_series(1, 4) i JOIN generate_series(1, 4) j ON i + j > 4;
```

Nested Loop (actual rows=10 loops=1)

Join Filter: ((i.i + j.j) > 4)

Rows Removed by Join Filter: 6

-> Function Scan on generate_series i (actual rows=4 loops=1)

-> Function Scan on generate_series j (actual rows=4 loops=4)

Rows Removed by Join Filter – отброшено соединенных записей

Hash Join + Hash

«Справа» однократно формируем хэш

```
EXPLAIN (ANALYZE, COSTS off, TIMING off, SUMMARY off)  
  SELECT * FROM pg_index i JOIN pg_class c ON c.oid = i.indexrelid;
```

```
Hash Join (actual rows=166 loops=1)  
  Hash Cond: (c.oid = i.indexrelid)  
    -> Seq Scan on pg_class c (actual rows=424 loops=1)  
    -> Hash (actual rows=166 loops=1)  
        Buckets: 1024  Batches: 1  Memory Usage: 38kB  
        -> Seq Scan on pg_index i (actual rows=166 loops=1)
```

Hash Cond – условие соединения по хэшу

Hash Join + Hash

«Справа» однократно формируем хэш

```
EXPLAIN (ANALYZE, COSTS off, TIMING off, SUMMARY off)  
SELECT * FROM pg_index i LEFT JOIN pg_class c ON c.oid = i.indexrelid;
```

```
Hash Right Join (actual rows=424 loops=1)  
-> Hash (actual rows=166 loops=1)  
    Buckets: 1024  Batches: 1  Memory Usage: 38kB  
    ...
```

Hash Right Join – при LEFT JOIN Hash Full Join – при FULL JOIN

Hash Left Join – при RIGHT JOIN

Buckets – размер хэш-таблицы

Merge Join [+ Sort]

Соединение сортированных выборок

```
EXPLAIN (ANALYZE, COSTS off, TIMING off, SUMMARY off)  
  SELECT * FROM pg_index i JOIN pg_class c ON c.oid = i.indrelid;
```

Merge Join (actual rows=166 loops=1)

Merge Cond: (i.indrelid = c.oid)

-> Index Scan using pg_index_indrelid_index on pg_index i (actual rows=166 loops=1)

-> Index Scan using pg_class_oid_index on pg_class c (actual rows=416 loops=1)

Merge Left Join – при LEFT JOIN

Merge Full Join – при FULL JOIN

Merge Cond – условие соединения слиянием

... Semi Join

Соединение при наличии **EXISTS/IN**

```
EXPLAIN (COSTS off) SELECT * FROM generate_series(1, 4) i
WHERE EXISTS(SELECT * FROM generate_series(1, 4) j WHERE j = i);
```

```
EXPLAIN (COSTS off) SELECT * FROM generate_series(1, 4) i
WHERE i IN (SELECT * FROM generate_series(1, 4) j);
```

Merge Semi Join

Merge Cond: (i.i = j.j)

-> Sort

Sort Key: i.i

-> Function Scan on generate_series i

-> Sort

Sort Key: j.j

-> Function Scan on generate_series j

... Anti Join

Соединение при наличии **NOT EXISTS**

```
EXPLAIN (COSTS off) SELECT * FROM generate_series(1, 4) i
WHERE NOT EXISTS(SELECT * FROM generate_series(1, 4) j WHERE j = i);
```

Hash Anti Join

Hash Cond: (i.i = j.j)

-> Function Scan on generate_series i

-> Hash

-> Function Scan on generate_series j

Materialize

Сохранение результата подузла в памяти

```
EXPLAIN (ANALYZE, COSTS off, TIMING off, SUMMARY off)
```

```
SELECT * FROM pg_class c JOIN pg_index i ON i.indexrelid <> c.oid;
```

```
Nested Loop (actual rows=70975 loops=1)
```

```
Join Filter: (i.indexrelid <> c.oid)
```

```
Rows Removed by Join Filter: 167
```

```
-> Seq Scan on pg_class c (actual rows=426 loops=1)
```

```
-> Materialize (actual rows=167 loops=426) -- многократно достали из памяти
```

```
    -> Seq Scan on pg_index i (actual rows=167 loops=1) -- однократно прочитали
```

Memoize

Кэш повторяющихся ключей

```
EXPLAIN (ANALYZE, COSTS off, TIMING off, SUMMARY off)
```

```
SELECT * FROM pg_class c JOIN pg_namespace ns ON c.relnamespace = ns.oid LIMIT 10;
```

```
Limit (actual rows=10 loops=1)
```

```
-> Nested Loop (actual rows=10 loops=1)
```

```
    -> Seq Scan on pg_class c (actual rows=10 loops=1)
```

```
    -> Memoize (actual rows=1 loops=10)
```

```
        Cache Key: c.relnamespace
```

```
        Cache Mode: logical
```

```
        Hits: 8 Misses: 2 Evictions: 0 Overflows: 0 Memory Usage: 1kB -- всего 2 промаха
```

```
    -> Index Scan using pg_namespace_oid_index on pg_namespace ns (actual rows=1 loops=2)
```

```
        Index Cond: (oid = c.relnamespace)
```


Параллелизм операций



Gather [Merge] + Parallel ...

«Разделяй и властвуй!»

раздаем задачи worker-процессам, потом собираем
[отсортированные] результаты

запуск параллелизма можно форсировать

<https://postgrespro.ru/docs/postgresql/15/runtime-config-query#GUC-PARALLEL-SETUP-COST>

```
SET parallel_setup_cost = 0;  
SET parallel_tuple_cost = 0;  
SET min_parallel_index_scan_size = '0kB';  
SET min_parallel_table_scan_size = '0kB';
```

Gather [Merge] + Parallel ...

Получение данных

```
EXPLAIN (ANALYZE, COSTS off, TIMING off, SUMMARY off)  
  SELECT * FROM pg_class WHERE relname <> 'x';
```

Gather (actual rows=426 **loops=1**)

Workers Planned: 2

Workers Launched: 2

-> **Parallel Seq Scan** on **pg_class** (actual rows=142 **loops=3**) -- **leader + 2 workers**
 Filter: (relname <> 'x'::name)

workers Planned/Launched – сколько worker'ов хотели/получили

Finalize ... + Partial ...

Вычисление агрегатов

```
EXPLAIN (ANALYZE, COSTS off, TIMING off, SUMMARY off)  
SELECT count(*) FROM pg_class WHERE relname <> 'x';
```

```
Finalize Aggregate (actual rows=1 loops=1)
```

```
-> Gather (actual rows=3 loops=1)
```

```
Workers Planned: 2
```

```
Workers Launched: 2
```

```
-> Partial Aggregate (actual rows=1 loops=3)
```

```
-> Parallel Index Only Scan using pg_class_relname_nsp_index on pg_class (actual rows=143 loops=3)
```

```
Filter: (relname <> 'x'::name)
```

```
Heap Fetches: 23
```

Параллелизм соединений

Соединение внутри worker'a

```
EXPLAIN (COSTS off) SELECT * FROM pg_index i JOIN pg_class c ON c.oid = i.indexrelid;
```

Gather

Workers Planned: 2

-> **Parallel Hash Join**

Hash Cond: (c.oid = i.indexrelid)

-> **Parallel Seq Scan** on pg_class c

-> **Parallel Hash**

-> **Parallel Seq Scan** on pg_index i

Параллелизм соединений

Соединение внутри leader'a

```
EXPLAIN (COSTS off) SELECT * FROM pg_index i FULL JOIN pg_class c ON c.oid = i.indexrelid;
```

Hash Full Join

Hash Cond: (c.oid = i.indexrelid)

-> Gather

Workers Planned: 2

-> Parallel Seq Scan on pg_class c

-> Hash

-> Gather

Workers Planned: 2

-> Parallel Seq Scan on pg_index i

Изменение данных



Insert

<https://postgrespro.ru/docs/postgresql/15/sql-insert>

Вставка данных

```
CREATE TABLE tbl(i integer);  
EXPLAIN (ANALYZE, COSTS off, TIMING off, SUMMARY off)  
  INSERT INTO tbl SELECT * FROM generate_series(1, 1e6);
```

```
Insert on tbl (actual rows=0 loops=1) -- просто так ничего не возвращает  
-> Function Scan on generate_series (actual rows=1000000 loops=1)
```

```
EXPLAIN (ANALYZE, COSTS off, TIMING off, SUMMARY off)  
  INSERT INTO tbl SELECT * FROM generate_series(1, 1e6) RETURNING *;
```

```
Insert on tbl (actual rows=1000000 loops=1) -- только с RETURNING  
-> Function Scan on generate_series (actual rows=1000000 loops=1)
```


Conflict Resolution

Вставка данных с **ON CONFLICT**

```
TRUNCATE TABLE tbl;  
CREATE UNIQUE INDEX ON tbl(i);  
EXPLAIN (ANALYZE, COSTS off, TIMING off, SUMMARY off)  
  INSERT INTO tbl SELECT * FROM generate_series(1, 1e6) ON CONFLICT DO NOTHING;
```

Insert on tbl (actual rows=0 loops=1)

Conflict Resolution: NOTHING

Tuples Inserted: 1000000

Conflicting Tuples: 0

-> Function Scan on generate_series (actual rows=1000000 loops=1)

Tuples Inserted – сколько записей вставлено

Conflict Resolution

Вставка данных с **ON CONFLICT**

```
EXPLAIN (ANALYZE, COSTS off, TIMING off, SUMMARY off)
INSERT INTO tbl SELECT * FROM generate_series(1, 1e3)
ON CONFLICT DO NOTHING;
```

Insert on tbl (actual rows=0 loops=1)

Conflict Resolution: NOTHING

Tuples Inserted: 0

Conflicting Tuples: 1000

-> Function Scan on generate_series (actual rows=1000 loops=1)

Conflicting Tuples – сколько записей вызвало конфликт вставки

Conflict Resolution

Вставка данных с **ON CONFLICT**

```
EXPLAIN (ANALYZE, COSTS off, TIMING off, SUMMARY off)
INSERT INTO tbl SELECT * FROM generate_series(1, 1e3)
ON CONFLICT(i) DO UPDATE SET i = EXCLUDED.i + 1e6;
```

Insert on tbl (actual rows=0 loops=1)

Conflict Resolution: UPDATE

Conflict Arbiter Indexes: tbl_i_idx

Tuples Inserted: 0

Conflicting Tuples: 1000

-> Function Scan on generate_series (actual rows=1000 loops=1)

Conflict Arbiter Indexes – конфликтующие уникальные индексы

Update

<https://postgrespro.ru/docs/postgresql/15/sql-update>

Обновление данных мимо индекса (типы!)

```
EXPLAIN (ANALYZE, COSTS off, TIMING off, SUMMARY off)
```

```
UPDATE tbl SET i = i + 2e6 WHERE i > 1e6;
```

```
Update on tbl (actual rows=0 loops=1)
```

```
-> Seq Scan on tbl (actual rows=1000 loops=1)
```

```
Filter: ((i)::numeric > '1000000'::numeric)
```

```
Rows Removed by Filter: 999000
```

Update

Обновление данных по индексу

```
EXPLAIN (ANALYZE, COSTS off, TIMING off, SUMMARY off)
```

```
UPDATE tbl SET i = i + 2e6 WHERE i > 1e6::integer;
```

```
Update on tbl (actual rows=0 loops=1)
```

```
-> Index Scan using tbl_i_idx on tbl (actual rows=1000 loops=1)
```

```
Index Cond: (i > 1000000)
```

Delete

<https://postgrespro.ru/docs/postgresql/15/sql-delete>

Удаление данных

```
EXPLAIN (ANALYZE, COSTS off, TIMING off, SUMMARY off)
```

```
DELETE FROM tbl WHERE i > 1e6::integer;
```

```
Delete on tbl (actual rows=0 loops=1)
```

```
-> Index Scan using tbl_i_idx on tbl (actual rows=1000 loops=1)
```

```
Index Cond: (i > 1000000)
```

Merge

<https://postgrespro.ru/docs/postgresql/15/sql-merge>

Совмещение данных

```
TRUNCATE TABLE tbl;  
EXPLAIN (ANALYZE, COSTS off, TIMING off, SUMMARY off)  
MERGE INTO tbl  
  USING (SELECT i::integer FROM generate_series(1, 1e3) i) T(i)  
  ON tbl.i = T.i  
  WHEN NOT MATCHED THEN  
    INSERT (i) VALUES (T.i)  
  WHEN MATCHED THEN  
    UPDATE SET i = T.i + 1e6;
```

Merge

Совмещение данных

Merge on tbl (actual rows=0 loops=1)

Tuples: inserted=1000

-> Nested Loop Left Join (actual rows=1000 loops=1)

-> Function Scan on generate_series i (actual rows=1000 loops=1)

-> Index Scan using tbl_i_idx on tbl (actual rows=0 loops=1000)

Index Cond: (i = (i.i)::integer)

Merge

Совмещение данных

```
TRUNCATE TABLE tbl;  
EXPLAIN (ANALYZE, COSTS off, TIMING off, SUMMARY off)  
MERGE INTO tbl  
  USING (SELECT 900 + i::integer FROM generate_series(1, 1e3) i) T(i)  
  ON tbl.i = T.i  
  WHEN NOT MATCHED THEN  
    INSERT (i) VALUES (T.i)  
  WHEN MATCHED THEN  
    UPDATE SET i = T.i + 1e6;
```

Merge

Совмещение данных

Merge on tbl (actual rows=0 loops=1)

Tuples: inserted=900 updated=100

-> Hash Left Join (actual rows=1000 loops=1)

Hash Cond: ((900 + (i.i)::integer) = tbl.i)

-> Function Scan on generate_series i (actual rows=1000 loops=1)

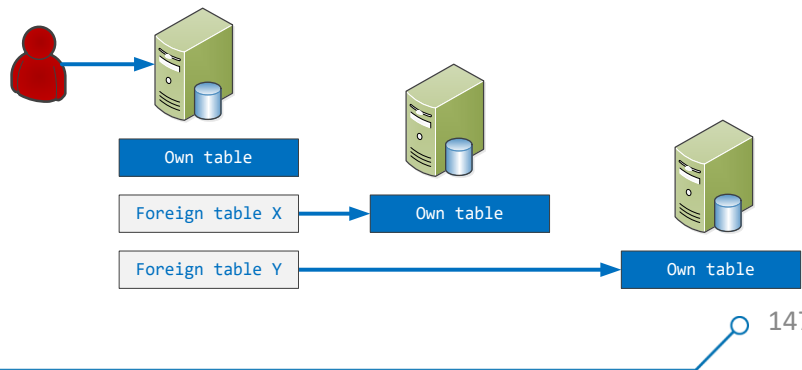
-> Hash (actual rows=1000 loops=1)

Buckets: 1024 Batches: 1 Memory Usage: 51kB

-> Seq Scan on tbl (actual rows=1000 loops=1)

Foreign *

Изменение данных на внешнем сервере



Триггеры



Trigger

<https://postgrespro.ru/docs/postgresql/15/sql-createtrigger>

STATEMENT-триггер

```
CREATE OR REPLACE FUNCTION count_trigger() RETURNS trigger AS $$  
BEGIN  
    RAISE NOTICE 'count = %', (SELECT count(*) FROM new_ref);  
    RETURN NULL;  
END;  
$$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER tbl_trigger_stm AFTER INSERT ON tbl  
    REFERENCING NEW TABLE AS new_ref  
    FOR EACH STATEMENT  
        EXECUTE PROCEDURE count_trigger();
```

Trigger

ROW-триггер

```
CREATE OR REPLACE FUNCTION notice_trigger() RETURNS trigger AS $$
BEGIN
    RAISE NOTICE 'NEW = %', NEW;
    RETURN NULL;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER tbl_trigger_row AFTER INSERT ON tbl
FOR EACH ROW
EXECUTE PROCEDURE notice_trigger();
```

Trigger

Время триггеров [часто*] идет плюсом

```
EXPLAIN (ANALYZE, COSTS off, SUMMARY off)
```

```
INSERT INTO tbl SELECT * FROM generate_series(1, 1e3);
```

```
Insert on tbl (actual time=2.887..2.888 rows=0 loops=1)
```

```
-> Function Scan on generate_series (actual time=0.168..0.415 rows=1000 loops=1)
```

```
Trigger tbl_trigger_row: time=5.147 calls=1000 -- для каждой записи
```

```
Trigger tbl_trigger_stm: time=0.255 calls=1 -- для запроса целиком
```

Trigger for constraint

Внешние ключи – тоже триггеры

<https://habr.com/ru/companies/tensor/articles/665118/>

```
CREATE TABLE tbl_ref(  
  i integer  
  REFERENCES tbl(i)  
  ON DELETE CASCADE  
, j integer  
);
```


Trigger for constraint

Внешние ключи (Foreign Keys) – тоже триггеры

```
EXPLAIN (ANALYZE, COSTS off, SUMMARY off)
```

```
INSERT INTO tbl_ref SELECT j, j FROM generate_series(1, 1e3) j;
```

```
Insert on tbl_ref (actual time=1.503..1.503 rows=0 loops=1)
```

```
-> Function Scan on generate_series j (actual time=0.164..0.459 rows=1000 loops=1)
```

```
Trigger for constraint tbl_ref_i_fkey: time=11.654 calls=1000
```

Trigger for constraint

Внешние ключи (Foreign Keys) – тоже триггеры

```
EXPLAIN (ANALYZE, COSTS off, SUMMARY off)
```

```
DELETE FROM tbl;
```

```
Delete on tbl (actual time=1.079..1.080 rows=0 loops=1)
```

```
-> Seq Scan on tbl (actual time=0.019..0.157 rows=1000 loops=1)
```

```
Trigger for constraint tbl_ref_i_fkey: time=62.533 calls=1000
```

Операции в плане

Получение данных

`Result, ... Scan`

Построение битовых карт

`Bitmap...`

Вложенные запросы

`InitPlan, SubPlan`

Операции над множествами

`Append, Intersect, Except, SetOp, ...`

Обработка

`Limit, Sort, Incremental Sort, Unique`

Группировка

`Aggregate, Group, GroupAggregate, ...`

Соединения

`Nested Loop, Hash/Merge Join, ... Semi/Anti Join`

Параллелизм операций

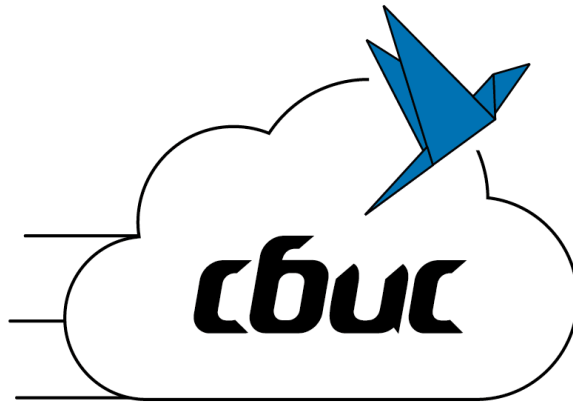
`Parallel ..., Gather, Gather Merge`

Изменение данных

`Insert, Update, Delete, Merge`

Триггеры/Foreign Keys

`Trigger, Trigger for constraint`



Спасибо за внимание!

Боровиков Кирилл

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

sbis.ru / tensor.ru