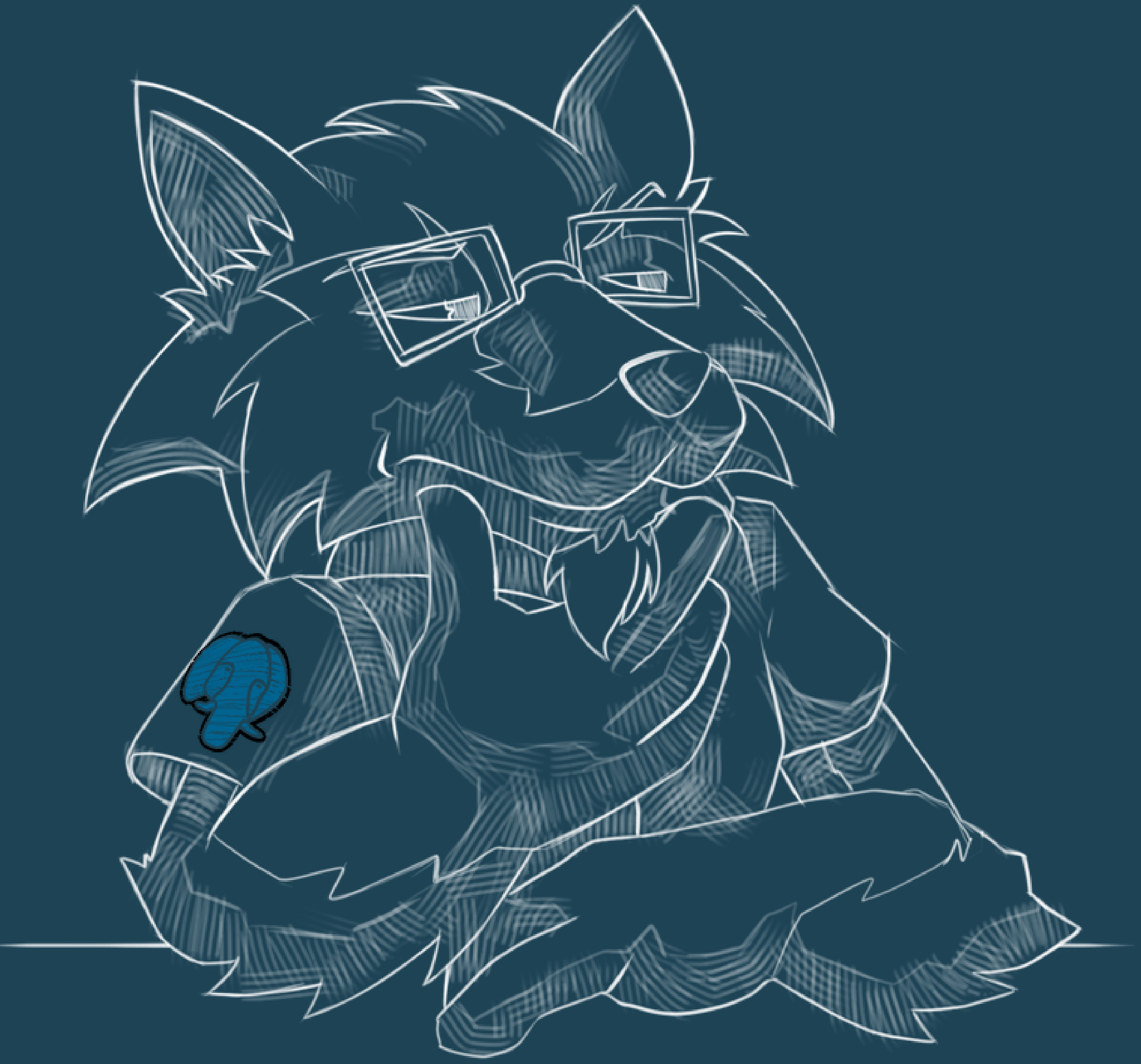
Hi! I'm Wolfnik. Postgres DBA and developer, today someone notified me that database slow...

Wolfgres

Have you ever heard?

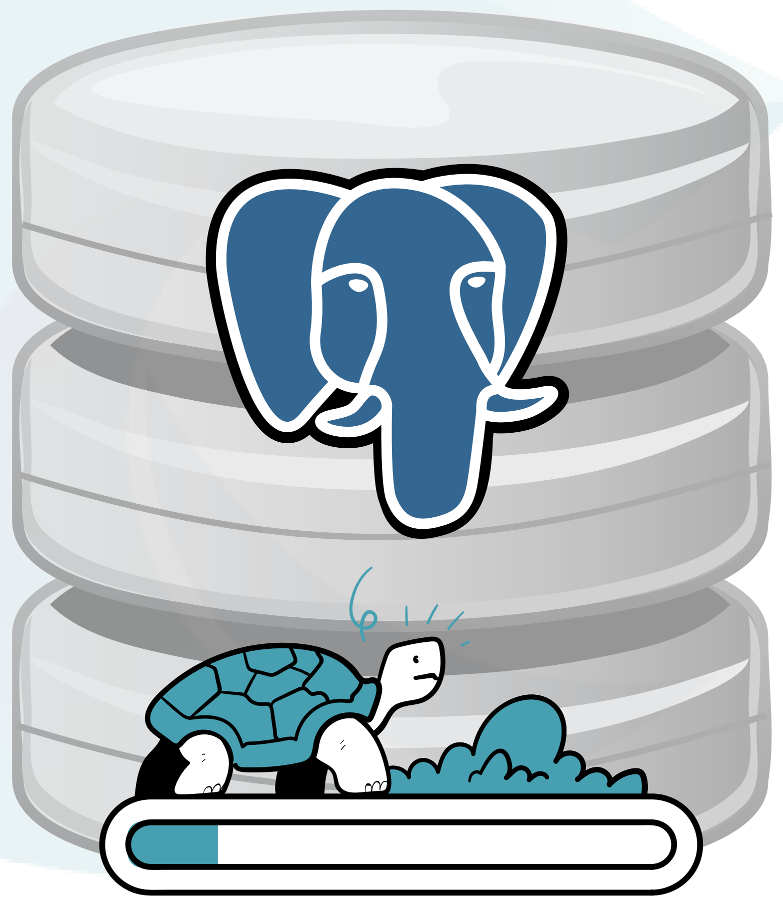
"the database is slow...."



Wolfgres

Possibles causes of the slow database

1. The hardware is not enough
    a. To monitor resources of your hardware
    b. To monitor how many OS's resources your database consumes
2. A bad configuration
    a. After monitoring check your parameters in postgrelsql.conf mainly memory section
3. Poorly designed queries
    a. First you need find the queries, some queries on pg_stat_activity could help you.
    b. Just a simple top command in Linux helps you too.
    c. Temporally, modify the log parameters in postgresql.conf to log slow queries
4. Another thousand possibles causes...



How to improve my query?

Tell me the truth...I'm...I'm ready to hear it.
THE PROBLEM ISN'T THE DATABASE. THE PROBLEM IS YOUR QUERY.
imgflip.com

Wolfgres

When we found slow query...

Fix it! you're the DBA!!...

Wolfnik read about a sentence that could help us to improve a Query

```
EXPLAIN
```

Wolfnik doesn’t know what EXPLAIN does and how it works...

```
EXPLAIN [my_slow_query];
```

What can be wrong?

just execute it...

```
$ psql wolfgres_db wolfnik_dba
prod_db=# EXPLAIN SELECT * FROM my_slow_query;
```

```
QUERY PLAN
WindowAgg  (cost=353394.22..353394.24 rows=1 width=1487) (actual time=13655.595..13655.634 rows=36 loops=1)
  ->  Sort  (cost=353394.22..353394.22 rows=1 width=1479) (actual time=13655.559..13655.563 rows=36 loops=1)
          Sort Key: tango_seven.delta DESC
          Sort Method: quicksort  Memory: 43kB
        ->  Nested Loop  (cost=67915.41..353394.21 rows=1 width=1479) (actual time=13042.811..13655.436 rows=36 loops=1)
              ->  Nested Loop  (cost=67915.27..353394.04 rows=1 width=1491) (actual time=13042.800..13655.280 rows=36 loops=1)
                    ->  Nested Loop  (cost=67915.00..353391.11 rows=1 width=1503) (actual time=13042.788..13655.086 rows=36 loops=1)
                            Join Filter: ((hotel_seven.uniform = three_xray.uniform) AND (hotel_seven.yankee = three_xray.yankee))
                          ->  Nested Loop  (cost=67914.57..353386.87 rows=1 width=1507) (actual time=13042.726..13653.510 rows=36 loops=1)
                                ->  Gather  (cost=67914.14..353382.83 rows=1 width=1495) (actual time=13042.686..13700.001 rows=21 loops=1)
                                        Workers Planned: 2
                                        Workers Launched: 2
                                      ->  Hash Join  (cost=66914.14..352382.73 rows=1 width=1495) (actual time=13156.217..13641.035 rows=7 loops=3)
                                              Hash Cond: (hotel_seven.seven_six1 = papa_golf.two)
                                            ->  Parallel Hash Join  (cost=66821.47..352288.89 rows=4 width=1499) (actual time=13017.340..13612.669 rows=152560 loops=3)
                                                    Hash Cond: ((tango_seven.uniform = hotel_seven.uniform) AND (tango_seven.yankee = hotel_seven.yankee))
                                                  ->  Parallel Seq Scan on six_whiskey tango_seven  (cost=0.00..284103.16 rows=181898 width=1479) (actual time=5.950..11805.629 rows=156382 loops=3)
                                                          Filter: ((foxtrot_papa = 'four_uniform'::bit(1)) AND (kilo_juliet = 'kilo_india'::bpchar))
                                                          Rows Removed by Filter: 1159958
                                                  ->  Parallel Hash  (cost=66819.34..66819.34 rows=142 width=20) (actual time=1061.731..1061.731 rows=291342 loops=3)
                                                          Buckets: 65536 (originally 1024)  Batches: 16 (originally 1)  Memory Usage: 3552kB
                                                        ->  Hash Join  (cost=2.64..66819.34 rows=142 width=20) (actual time=0.124..798.322 rows=291342 loops=3)
                                                                Hash Cond: ((hotel_seven.six_lima = foxtrot_xray.six_lima) AND (hotel_seven.foxtrot_six = foxtrot_xray.foxtrot_six))
                                                              ->  Parallel Seq Scan on tango_sierra hotel_seven  (cost=0.00..64142.91 rows=356315 width=20) (actual time=0.037..621.636 rows=291342 loops=3)
                                                                      Filter: ((foxtrot_papa = 'four_uniform'::bit(1)) AND (golf = 'seven_tango'::bpchar))
                                                                      Rows Removed by Filter: 87444
                                                              ->  Hash  (cost=2.62..2.62 rows=1 width=8) (actual time=0.066..0.066 rows=49 loops=3)
                                                                      Buckets: 1024  Batches: 1  Memory Usage: 10kB
                                                                    ->  Seq Scan on oscar foxtrot_xray  (cost=0.00..2.62 rows=1 width=8) (actual time=0.015..0.050 rows=49 loops=3)
                                                                            Filter: (foxtrot_papa = 'four_uniform'::bit(1))
                                                                            Rows Removed by Filter: 1
                                            ->  Hash  (cost=89.80..89.80 rows=230 width=4) (actual time=0.705..0.705 rows=15 loops=3)
                                                    Buckets: 1024  Batches: 1  Memory Usage: 9kB
                                                  ->  Seq Scan on three_seven papa_golf  (cost=0.00..89.80 rows=230 width=4) (actual time=0.039..0.693 rows=15 loops=3)
                                                          Filter: ((foxtrot_papa = 'four_uniform'::bit(1)) AND ((golf)::bpchar = 'alpha'::bpchar))
                                                          Rows Removed by Filter: 1515
                                ->  Index Scan using three_alpha on papa_charlie charlie  (cost=0.43..4.03 rows=1 width=12) (actual time=0.014..0.015 rows=2 loops=21)
                                        Index Cond: ((uniform = hotel_seven.uniform) AND (yankee = hotel_seven.yankee))
                                        Filter: ((foxtrot_papa = 'four_uniform'::bit(1)) AND (hotel_seven.six_lima = six_lima))
                          ->  Index Scan using sierra on six_sierra three_xray  (cost=0.43..4.23 rows=1 width=8) (actual time=0.025..0.042 rows=1 loops=36)
                                  Index Cond: ((uniform = charlie.uniform) AND (yankee = charlie.yankee))
                                  Filter: ((foxtrot_papa = 'four_uniform'::bit(1)) AND (bravo_six = 'kilo_india'::bpchar))
                                  Rows Removed by Filter: 16
                    ->  Index Scan using whiskey on xray_victor four_alpha  (cost=0.27..2.92 rows=1 width=4) (actual time=0.003..0.004 rows=1 loops=36)
                            Index Cond: (uniform = hotel_seven.uniform)
                            Filter: ((foxtrot_papa)::text = 'four_uniform'::text)
              ->  Index Scan using hotel_charlie on xray_november bravo_two  (cost=0.14..0.16 rows=1 width=4) (actual time=0.002..0.003 rows=1 loops=36)
                      Index Cond: (six_lima = charlie.six_lima)
                      Filter: (foxtrot_papa = 'four_uniform'::bit(1))
Planning time: 56.132 ms
```

Wolfgres

How to understand EXPLAIN without dying in the attempt

Alfredo Rodríguez

@AlfredoDrakoRod

Wolfgres

SELECT * FROM me;

- PostgreSQL Architect
- Wolfgres founder and start up
- I use PostgreSQL since 2010
- I'm really enthusiastic about Open Source and PostgreSQL community
- Call me Drako

Wolfgres
Postgres Enterprise

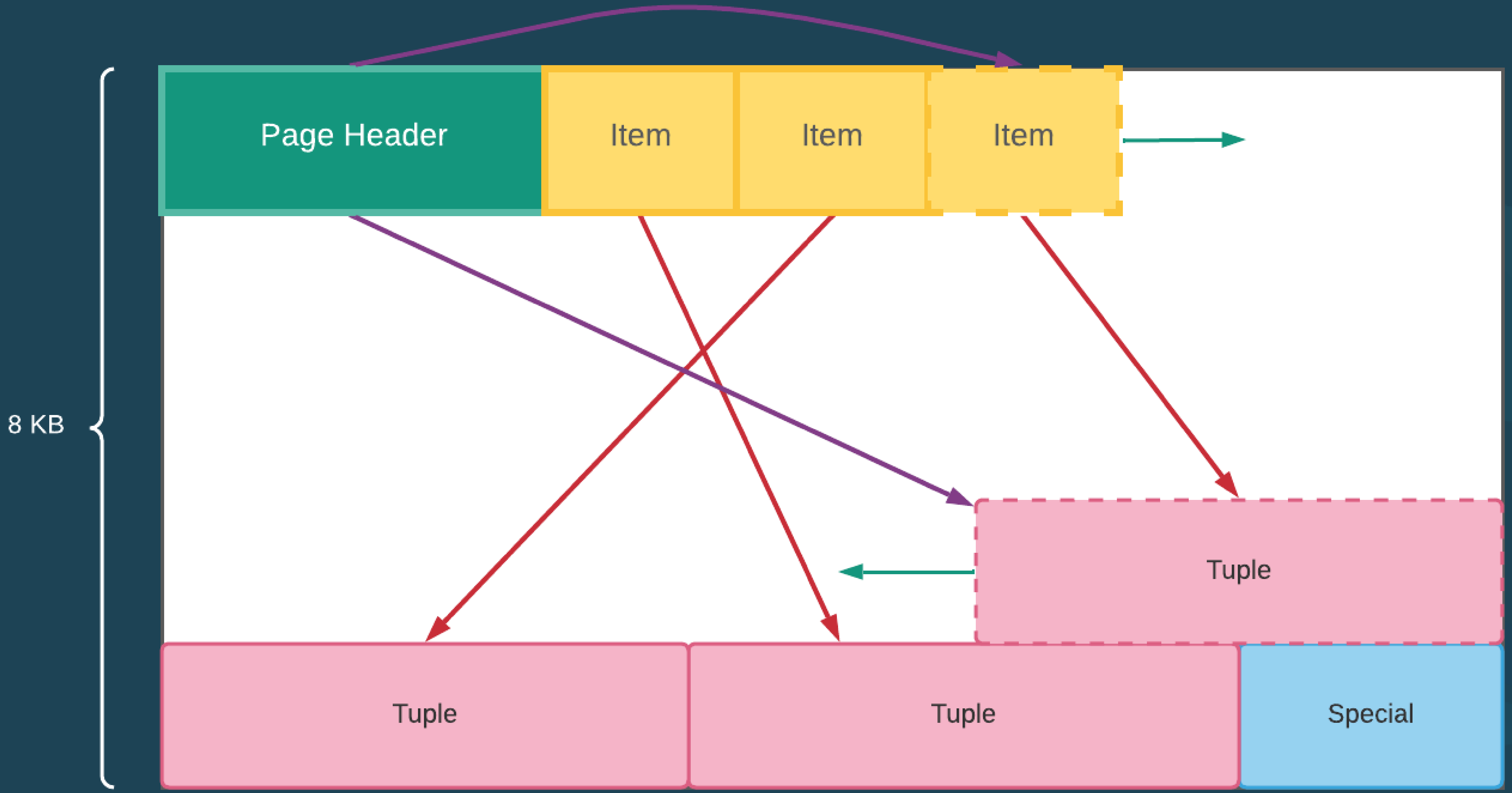
Before all! We need to understand basis

1. PostgreSQL basis
2. What is EXPLAIN
    a. ANALYZE
3. Scan Methods
4. Join Methods
5. Overview other elements
    a. Tools

Keep calm! We are trying to understand EXPLAIN, but first we need to divide and conquer

PostgreSQL Basis

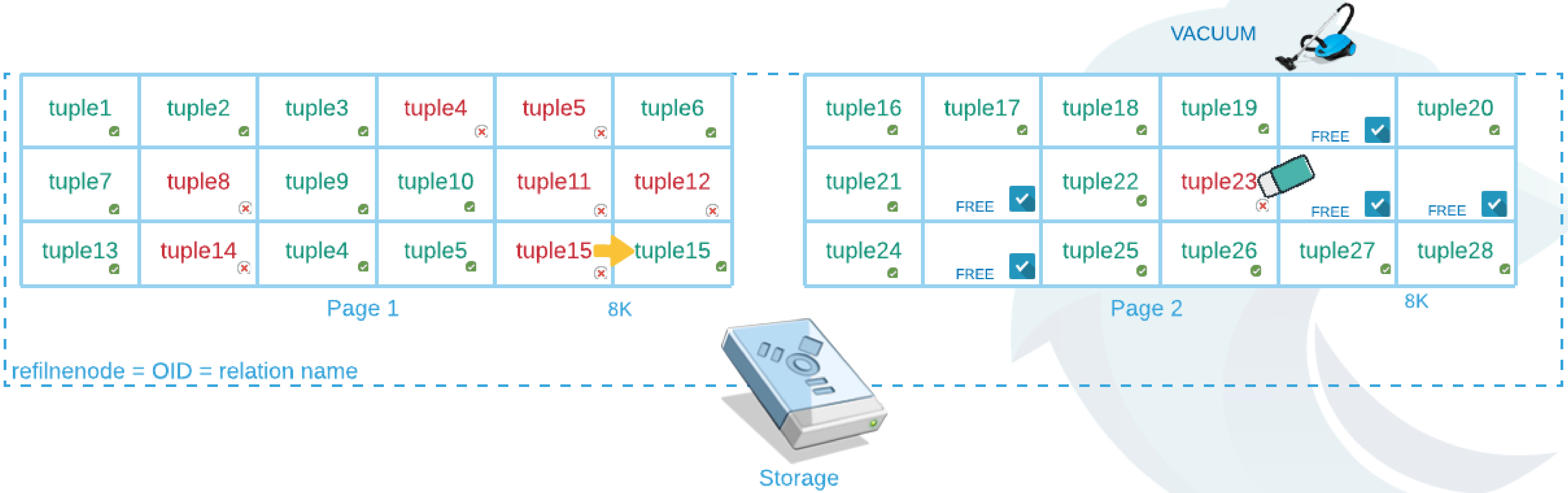
Where is data stored and scanned?



Wolfgres

How does relation storage look like?

1. Relation is a table or index
2. MVCC generates tuples versions when the data changes
3. VACUUM clean space (erase old versions of tuples) in the pages

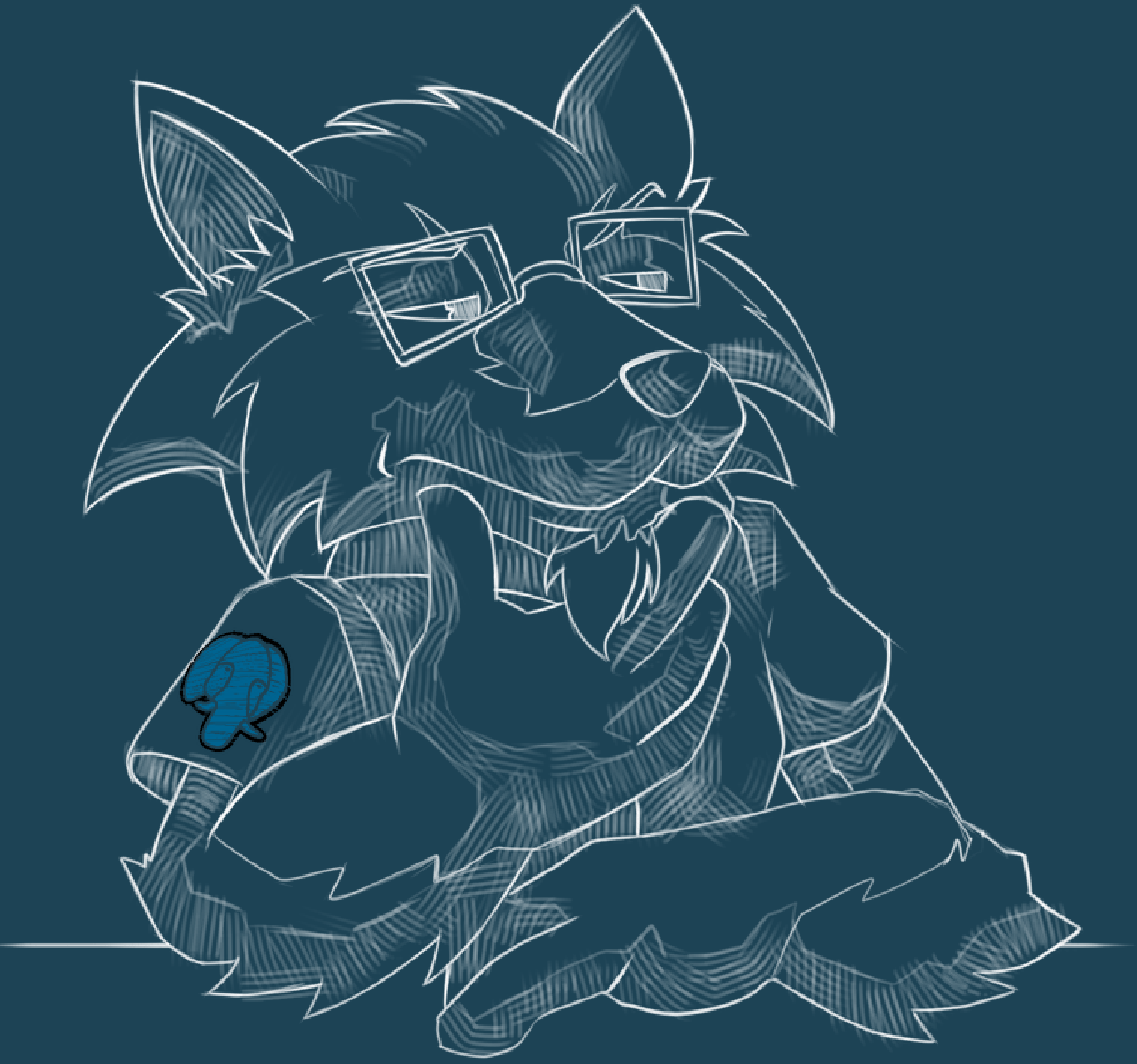


EXPLAIN Sentence

What is EXPLAIN?

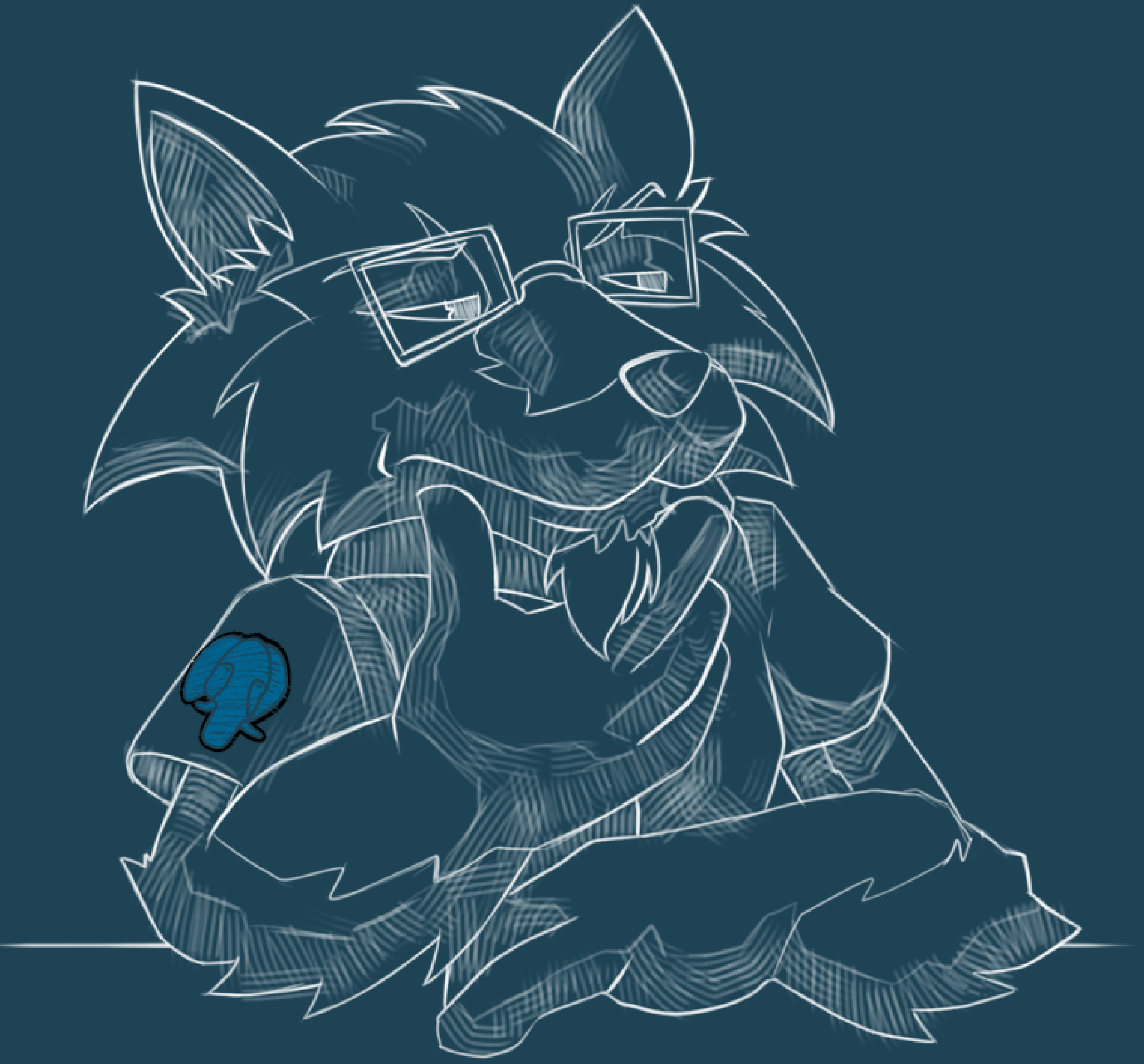
- This command displays the execution plan
  - Plan tree
- Table(s) referenced by the statement will be scanned
  - Scan methods
  - Join methods

```
EXPLAIN SELECT * FROM foo;
                              QUERY PLAN
---------------------------------------------------------
 Seq Scan on foo  (cost=0.00..155.00 rows=10000 width=4)
(1 row)
```



Wolfgres

What is EXPLAIN ANALYZE?



- Executes the query plan too
- Show plan and more
  - Actual time
  - Real rows
  - loops
- It shows how was done because it executes query
- Remember to execute the query (if you run UPDATE, it will update data)

EXPLAIN Output

Wolfgres

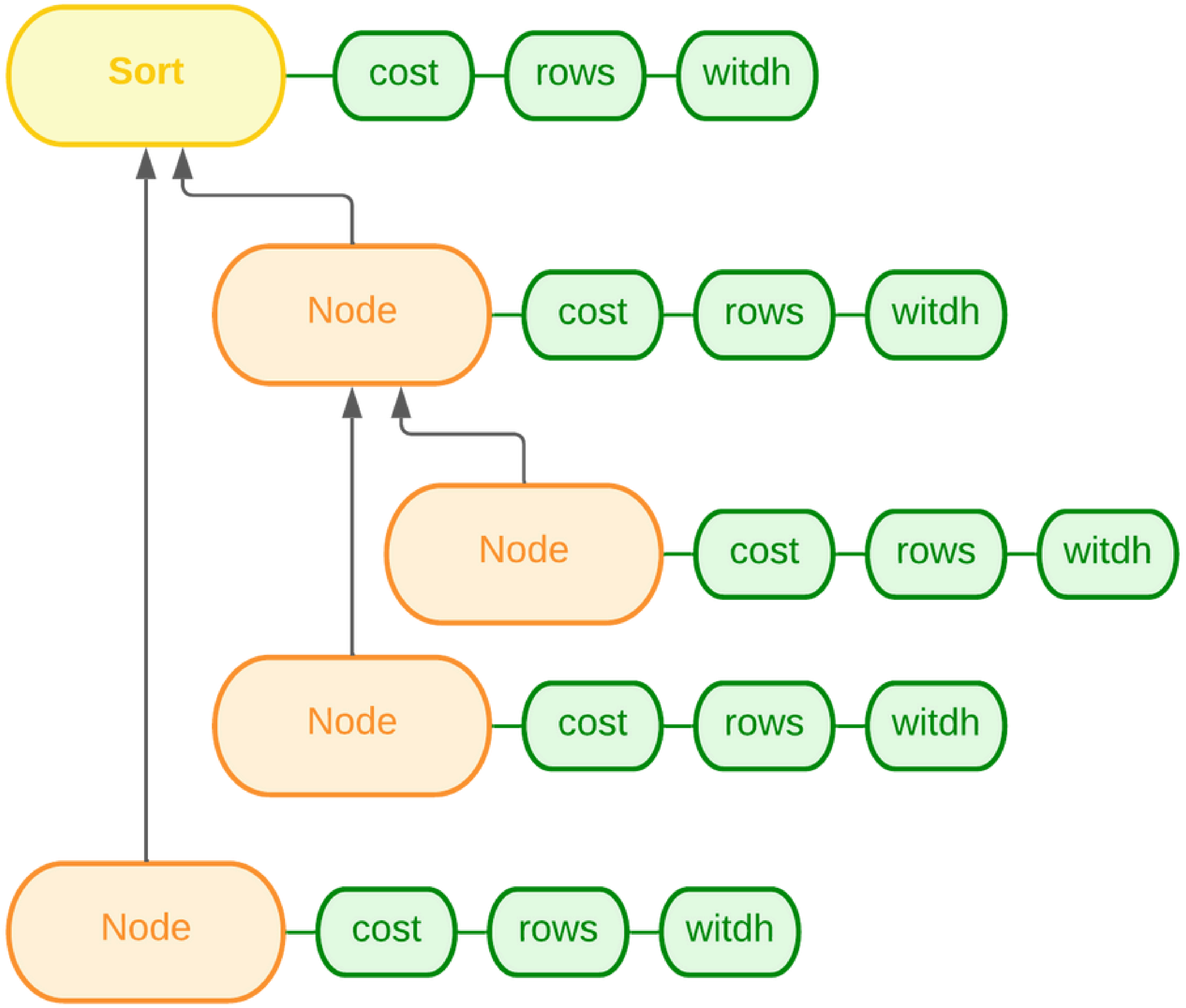
How to read plan EXPLAIN output?

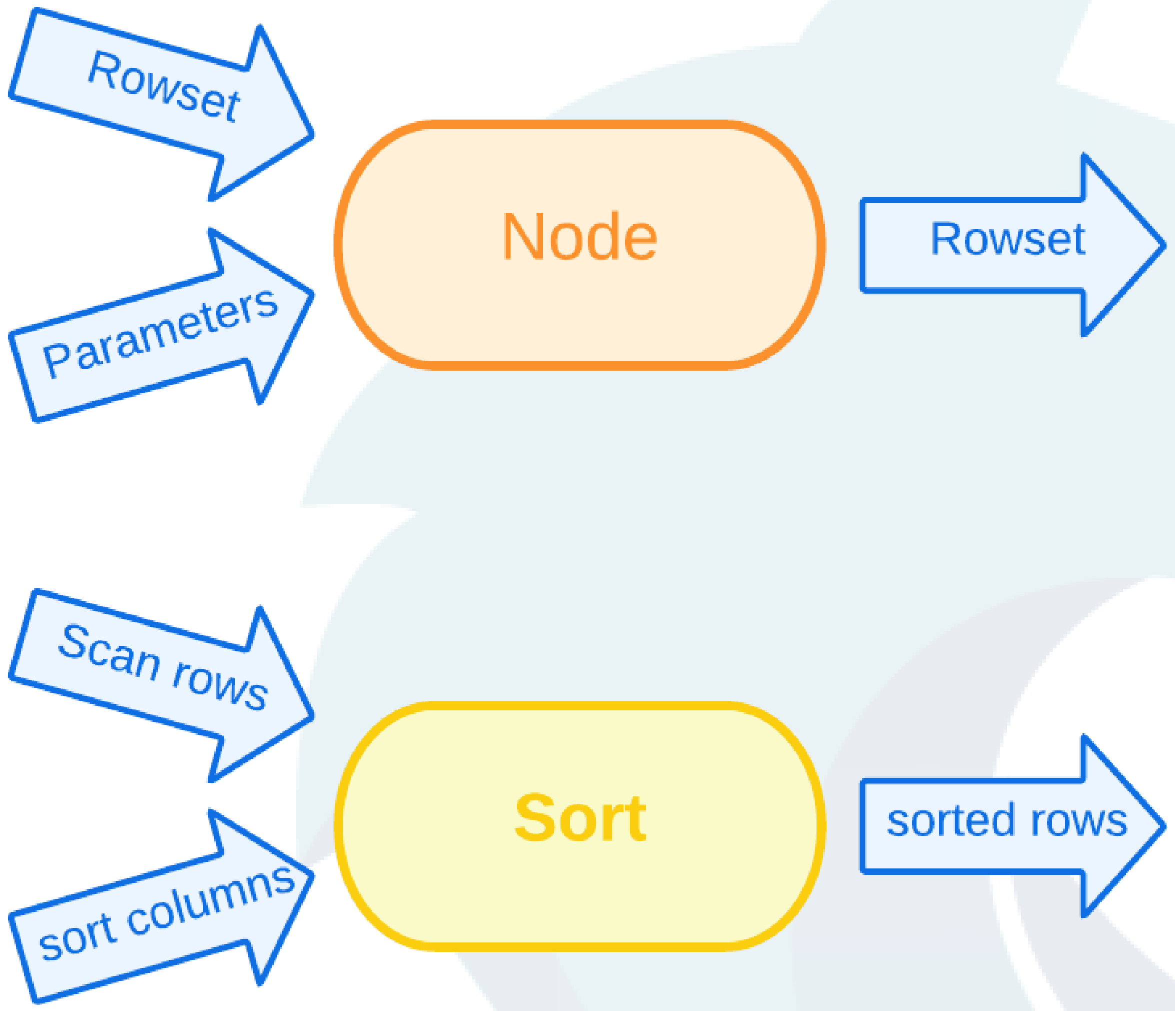
- Plan break query down in atomic “nodes”
- Inverted tree
- Read inside-out
  - Each node have a Resulset or do something after execute another node

```
wolfgres_db=# EXPLAIN SELECT * FROM customer c ORDER BY c.name;
                              QUERY PLAN
--------------------------------------------------------------------
 Sort  (cost=33.41..34.66 rows=500 width=49)
   Sort Key: name
   ->  Seq Scan on customer c  (cost=0.00..11.00 rows=500 width=49)
```

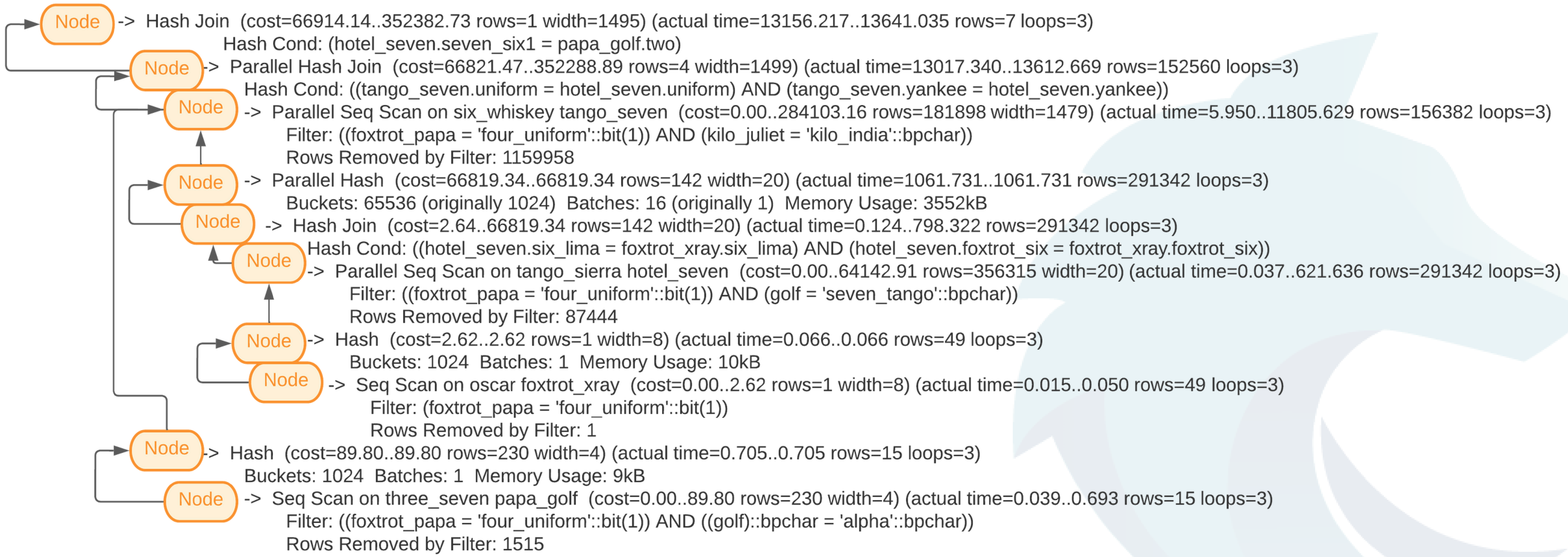
Wolfgres

How to read plan EXPLAIN output?



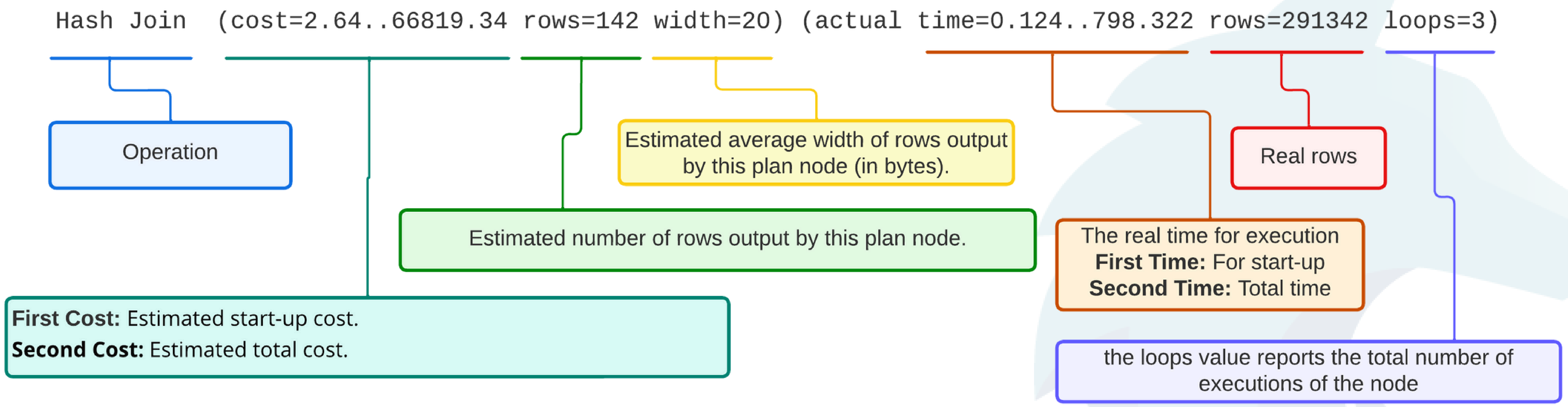


How to read plan EXPLAIN output?



```
->  Hash Join  (cost=66914.14..352382.73 rows=1 width=1495) (actual time=13156.217..13641.035 rows=7 loops=3)
          Hash Cond: (hotel_seven.seven_six1 = papa_golf.two)
      ->  Parallel Hash Join  (cost=66821.47..352288.89 rows=4 width=1499) (actual time=13017.340..13612.669 rows=152560 loops=3)
            Hash Cond: ((tango_seven.uniform = hotel_seven.uniform) AND (tango_seven.yankee = hotel_seven.yankee))
            ->  Parallel Seq Scan on six_whiskey tango_seven  (cost=0.00..284103.16 rows=181898 width=1479) (actual time=5.950..11805.629 rows=156382 loops=3)
                  Filter: ((foxtrot_papa = 'four_uniform'::bit(1)) AND (kilo_juliet = 'kilo_india'::bpchar))
                  Rows Removed by Filter: 1159958
            ->  Parallel Hash  (cost=66819.34..66819.34 rows=142 width=20) (actual time=1061.731..1061.731 rows=291342 loops=3)
                  Buckets: 65536 (originally 1024)  Batches: 16 (originally 1)  Memory Usage: 3552kB
                  ->  Hash Join  (cost=2.64..66819.34 rows=142 width=20) (actual time=0.124..798.322 rows=291342 loops=3)
                        Hash Cond: ((hotel_seven.six_lima = foxtrot_xray.six_lima) AND (hotel_seven.foxtrot_six = foxtrot_xray.foxtrot_six))
                        ->  Parallel Seq Scan on tango_sierra hotel_seven  (cost=0.00..64142.91 rows=356315 width=20) (actual time=0.037..621.636 rows=291342 loops=3)
                              Filter: ((foxtrot_papa = 'four_uniform'::bit(1)) AND (golf = 'seven_tango'::bpchar))
                              Rows Removed by Filter: 87444
                        ->  Hash  (cost=2.62..2.62 rows=1 width=8) (actual time=0.066..0.066 rows=49 loops=3)
                              Buckets: 1024  Batches: 1  Memory Usage: 10kB
                              ->  Seq Scan on oscar foxtrot_xray  (cost=0.00..2.62 rows=1 width=8) (actual time=0.015..0.050 rows=49 loops=3)
                                    Filter: (foxtrot_papa = 'four_uniform'::bit(1))
                                    Rows Removed by Filter: 1
      ->  Hash  (cost=89.80..89.80 rows=230 width=4) (actual time=0.705..0.705 rows=15 loops=3)
            Buckets: 1024  Batches: 1  Memory Usage: 9kB
            ->  Seq Scan on three_seven papa_golf  (cost=0.00..89.80 rows=230 width=4) (actual time=0.039..0.693 rows=15 loops=3)
                  Filter: ((foxtrot_papa = 'four_uniform'::bit(1)) AND ((golf)::bpchar = 'alpha'::bpchar))
                  Rows Removed by Filter: 1515
```

Information per each node



```
Hash Join  (cost=2.64..66819.34 rows=142 width=20) (actual time=0.124..798.322 rows=291342 loops=3)
```

Operation

First Cost: Estimated start-up cost.
Second Cost: Estimated total cost.

Estimated number of rows output by this plan node.

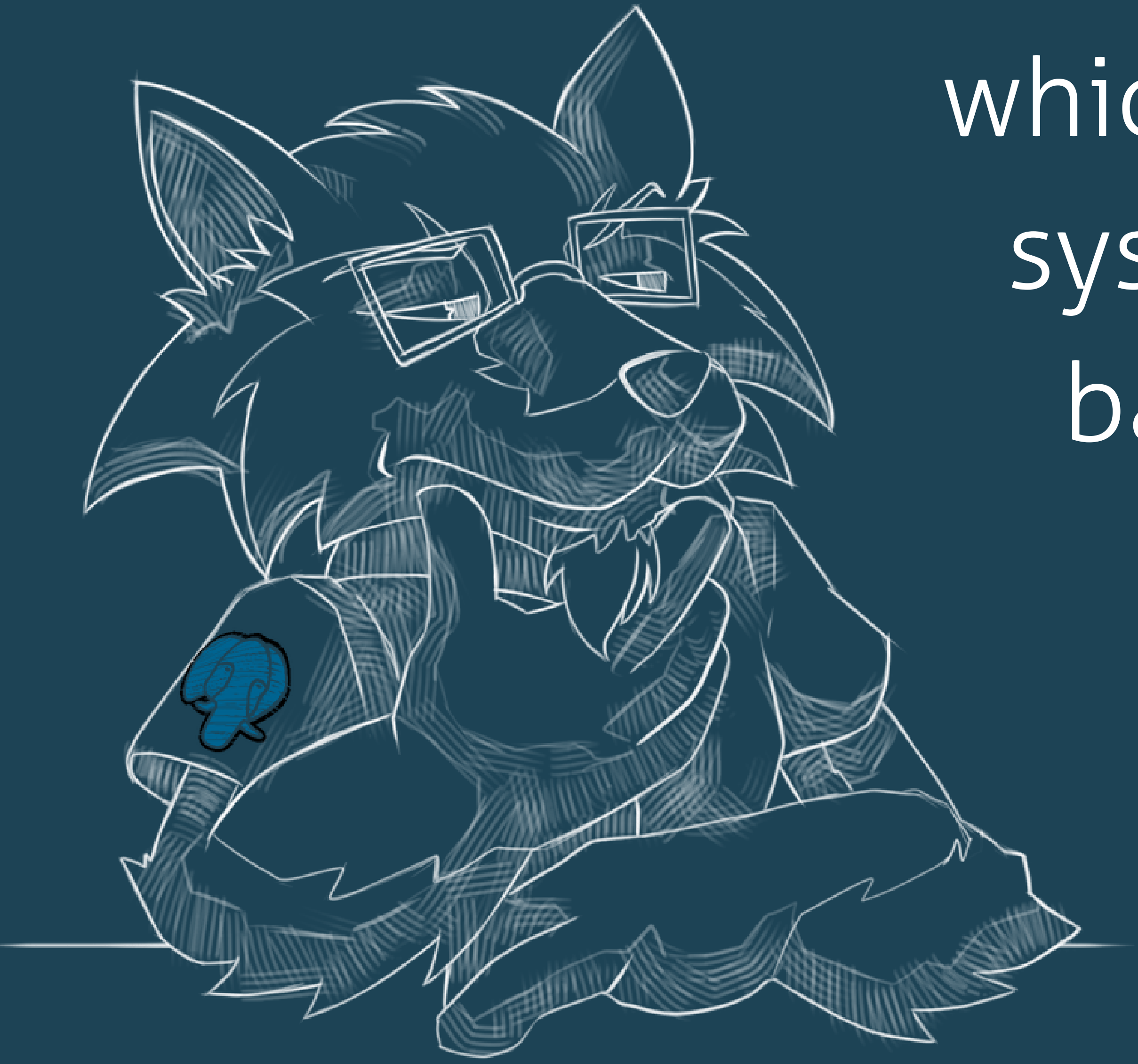
Estimated average width of rows output by this plan node (in bytes).

The real time for execution
First Time: For start-up
Second Time: Total time

Real rows

the loops value reports the total number of executions of the node

What does it mean cost?



"A figure in completely arbitrary cost units (float) which is intended to represent estimated time and system resources required to execute the query based on dubious theory and proven practice (some people adjust seq_page_cost)"
- Josh Berkus

Wolfgres

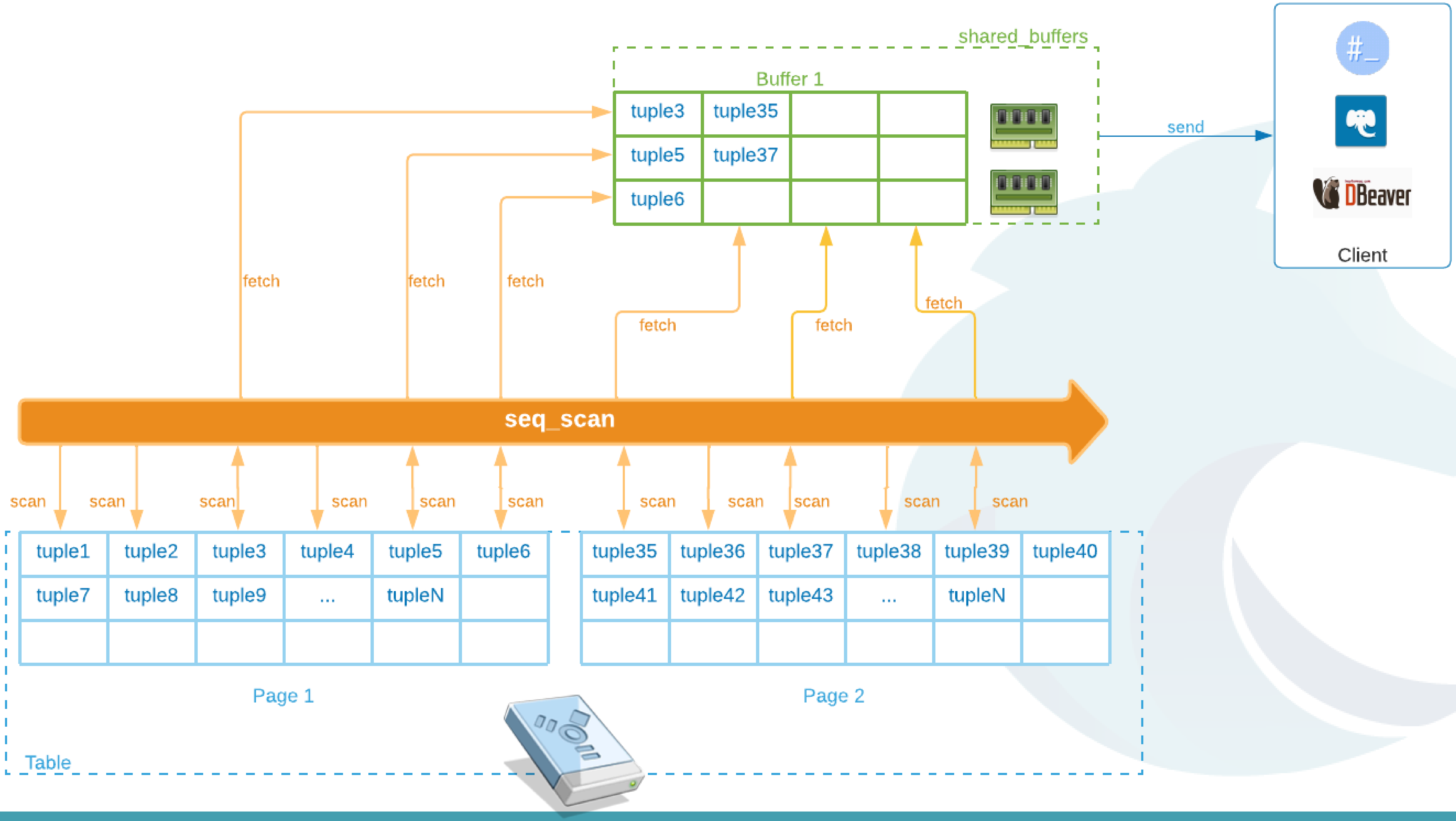
Cost

- Remember: “measured in cost units are arbitrary”
- It’s just a reference in your plan query
- Don’t compare cost between different queries, only the cost is a reference in the same one
- Select a plan with the lowest cost (the same query)

Scan Methods

Wolfgres

Seq Scan


shared_buffers
Buffer 1
tuple3
tuple35
tuple5
tuple37
tuple6
send
Client
DBeaver
fetch
fetch
fetch
fetch
fetch
fetch
seq_scan
scan
scan
scan
scan
scan
scan
scan
scan
scan
scan
scan
tuple1
tuple2
tuple3
tuple4
tuple5
tuple6
tuple7
tuple8
tuple9
...
tupleN
tuple35
tuple36
tuple37
tuple38
tuple39
tuple40
tuple41
tuple42
tuple43
...
tupleN
Page 1
Page 2
Table

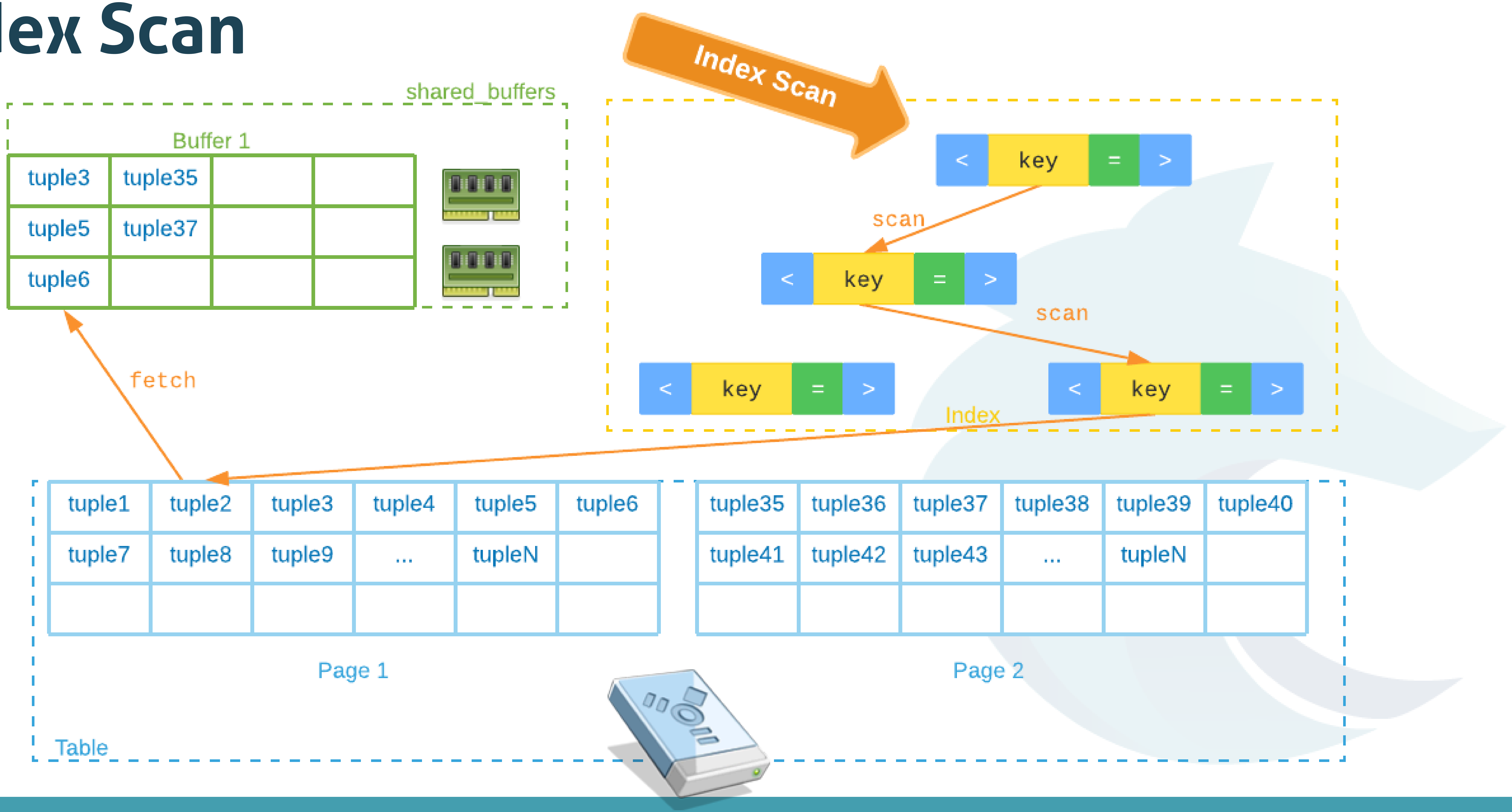
Seq Scan

- If necessary scan almost the complete table
- In some cases, Seq Scan is more cheap than Index

```
wolfgres_db=# EXPLAIN SELECT * FROM customer c;
                          QUERY PLAN
--------------------------------------------------------------
 Seq Scan on customer c  (cost=0.00..11.00 rows=500 width=49)
(1 fila)

wolfgres_db=# EXPLAIN SELECT * FROM customer c WHERE c.customer_id >= 1 AND c.customer_id <= 400;
                          QUERY PLAN
--------------------------------------------------------------
 Seq Scan on customer c  (cost=0.00..13.50 rows=400 width=49)
   Filter: ((customer_id >= 1) AND (customer_id <= 400))
(2 filas)
```

Index Scan



Wolfgres

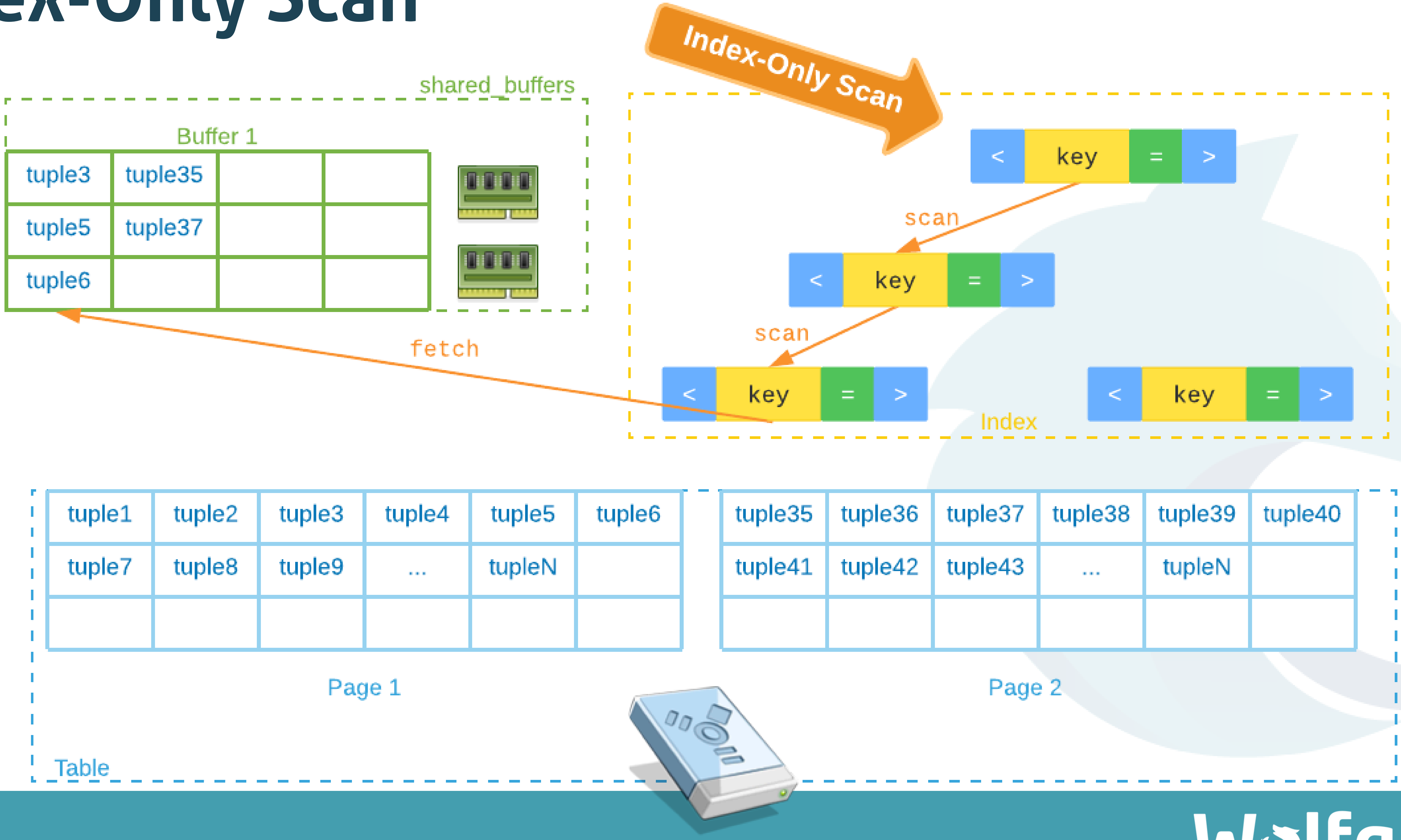
Index Scan

- First scan index to fetch data from page.

```
wolfgres_db=# EXPLAIN SELECT * FROM customer c WHERE c.customer_id = 35;
                                    QUERY PLAN
---------------------------------------------------------------------------------
 Index Scan using customer_pkey on customer c  (cost=0.27..8.29 rows=1 width=49)
   Index Cond: (customer_id = 35)
(2 filas)

wolfgres_db=# EXPLAIN SELECT * FROM customer c WHERE c.customer_id >= 35 AND c.customer_id <= 47;
                                     QUERY PLAN
----------------------------------------------------------------------------------
 Index Scan using customer_pkey on customer c  (cost=0.27..8.53 rows=13 width=49)
   Index Cond: ((customer_id >= 35) AND (customer_id <= 47))
(2 filas)
```

Index-Only Scan



Wolfgres

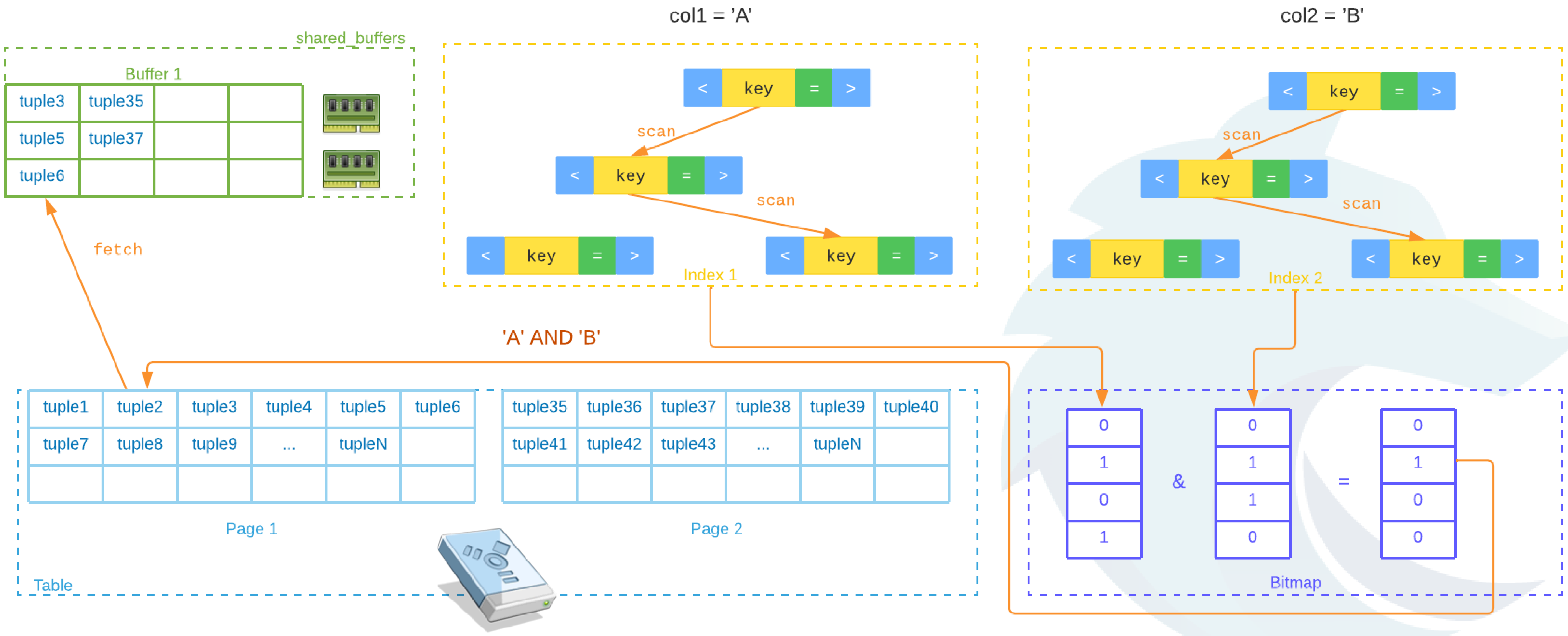
Index-Only Scan

- The data storage in index and fetch it.

```
       wolfgres_db=# EXPLAIN SELECT c.customer_id  FROM customer c WHERE c.customer_id = 35;
                                  QUERY PLAN
--------------------------------------------------------------------------------
 Index Only Scan using customer_pkey on customer c  (cost=0.27..8.29 rows=1 width=49)
   Index Cond: (customer_id = 35)
(2 filas)

wolfgres_db=# EXPLAIN SELECT c.customer_id FROM customer c WHERE c.customer_id >= 35 AND
c.customer_id <= 47;
                                   QUERY PLAN
---------------------------------------------------------------------------------
 Index Only Scan using customer_pkey on customer c  (cost=0.27..8.53 rows=13 width=49)
   Index Cond: ((customer_id >= 35) AND (customer_id <= 47))
(2 filas)
```

Bitmap Index Scan



https://dba.stackexchange.com/questions/119386/understanding-bitmap-heap-scan-and-bitmap-index-scan

Bitmap Index Scan

- Generate Bitmap from index to scan data

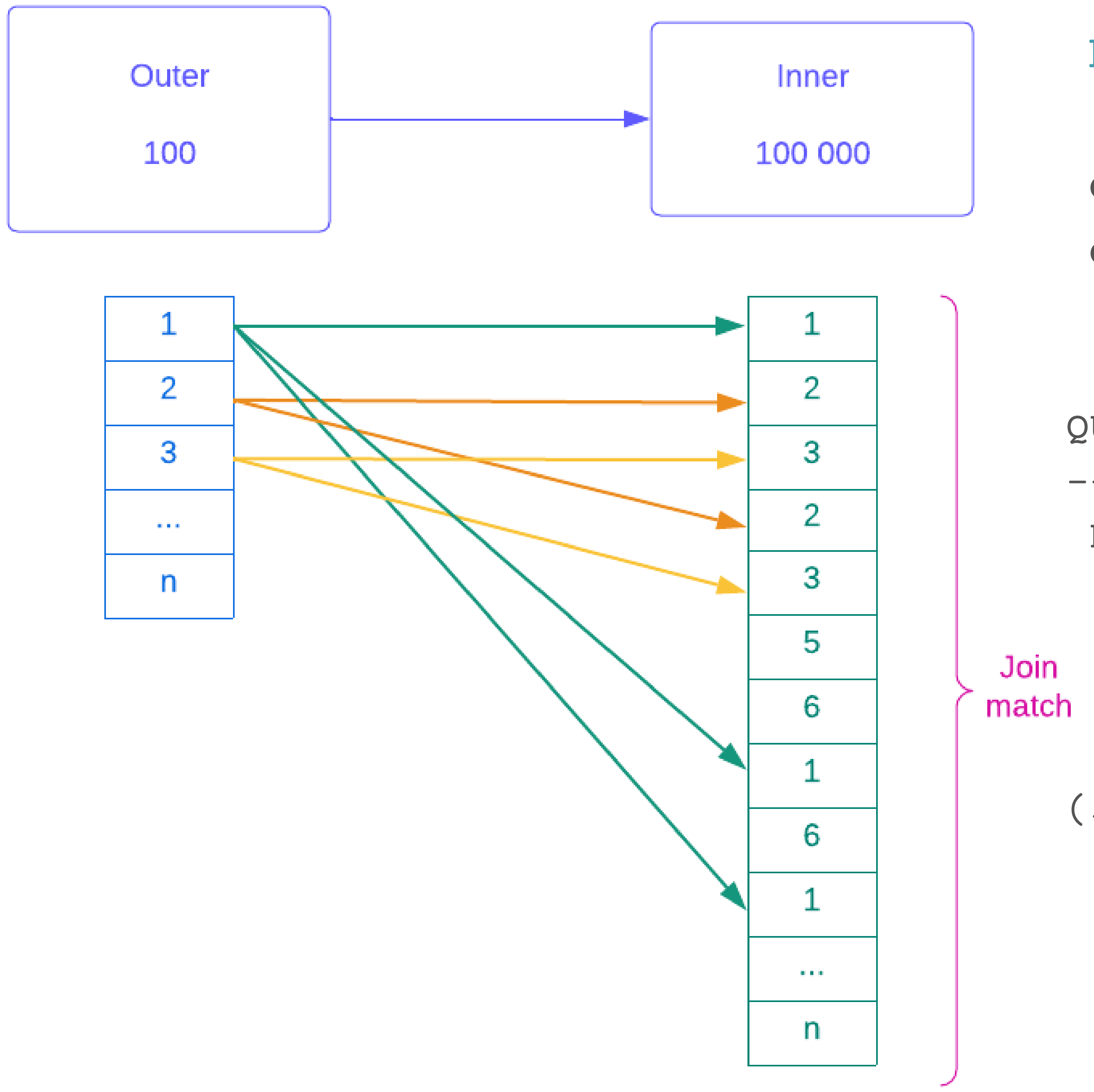
```
wolfgres_db=# EXPLAIN SELECT * FROM employee WHERE employee_id = 1 OR employee_id = 55;
                                   QUERY PLAN
---------------------------------------------------------------------------------
 Bitmap Heap Scan on employee  (cost=8.57..14.41 rows=2 width=84)
   Recheck Cond: ((employee_id = 1) OR (employee_id = 55))
   ->  BitmapOr  (cost=8.57..8.57 rows=2 width=0)
         ->  Bitmap Index Scan on employee_pkey  (cost=0.00..4.28 rows=1 width=0)
               Index Cond: (employee_id = 1)
         ->  Bitmap Index Scan on employee_pkey  (cost=0.00..4.28 rows=1 width=0)
               Index Cond: (employee_id = 55)

wolfgres_db=# CREATE INDEX first_name_idx ON wfg.employee(first_name);
wolfgres_db=# CREATE INDEX last_name_idx ON wfg.employee(last_name);
wolfgres_db=# EXPLAIN SELECT * FROM employee e WHERE e.first_name = 'Paul' AND e.last_name = 'Cole';
                                QUERY PLAN
----------------------------------------------------------------------------
 Bitmap Heap Scan on employee e  (cost=4.30..12.32 rows=1 width=84)
   Recheck Cond: ((first_name)::text = 'Paul'::text)
   Filter: ((last_name)::text = 'Cole'::text)
   ->  Bitmap Index Scan on first_name_idx  (cost=0.00..4.30 rows=3 width=0)
         Index Cond: ((first_name)::text = 'Paul'::text)
(5 filas)
```

Join Methods

Wolfgres

Nested Loop

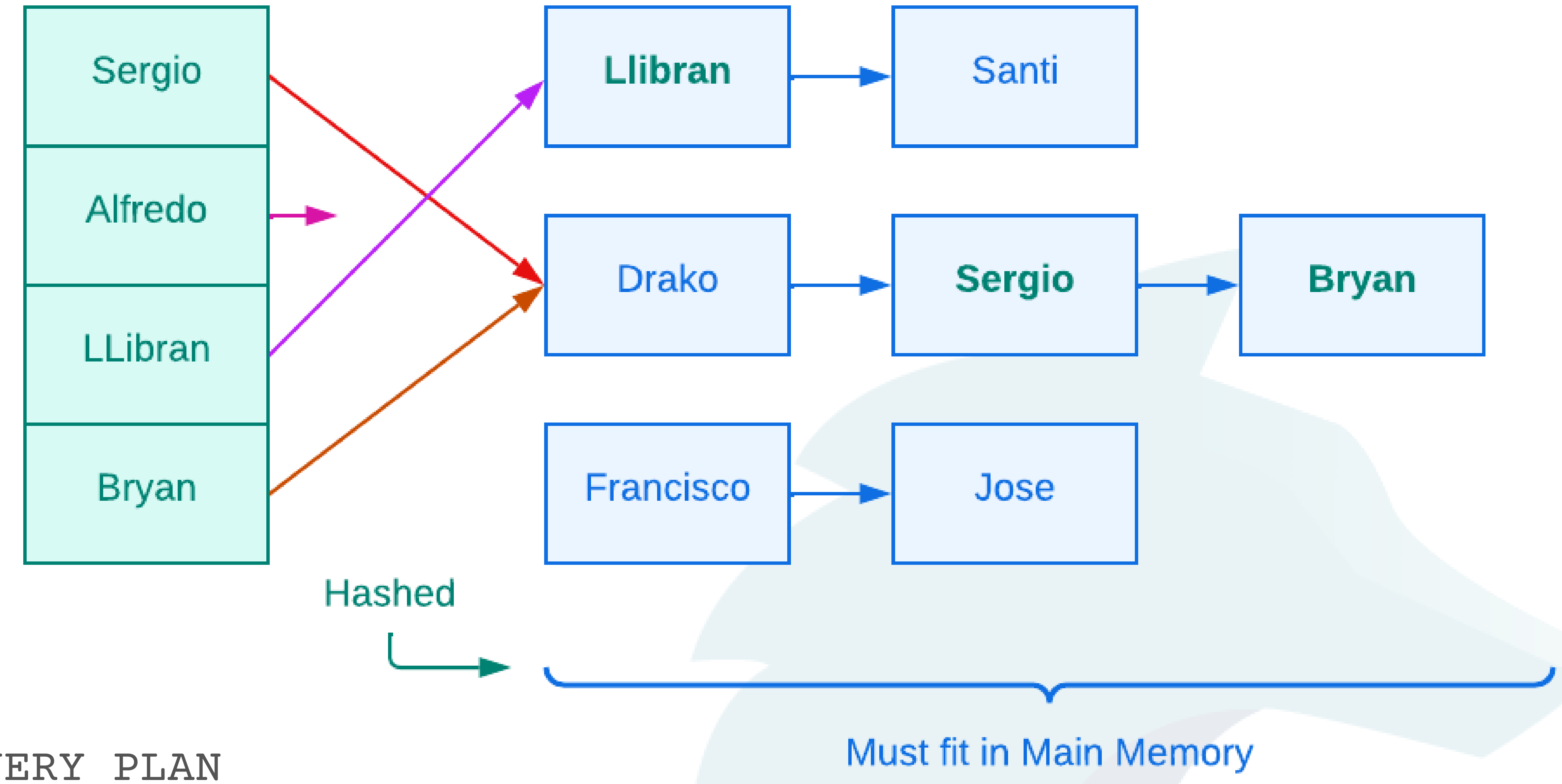


```
EXPLAIN
 SELECT * FROM product inner_table INNER JOIN
category outer_table ON   inner_table.category_id =
outer_table.category_id WHERE outer_table.category_id = 1;
```

```
QUERY PLAN
--------------------------------------------------------------------------------
 Nested Loop  (cost=0.00..590.94 rows=1684 width=406)
   ->  Seq Scan on category outer_table  (cost=0.00..1.09 rows=1 width=87)
         Filter: (category_id = 1)
   ->  Seq Scan on product inner_table  (cost=0.00..573.01 rows=1684 width=319)
         Filter: (category_id = 1)
(5 filas)
```

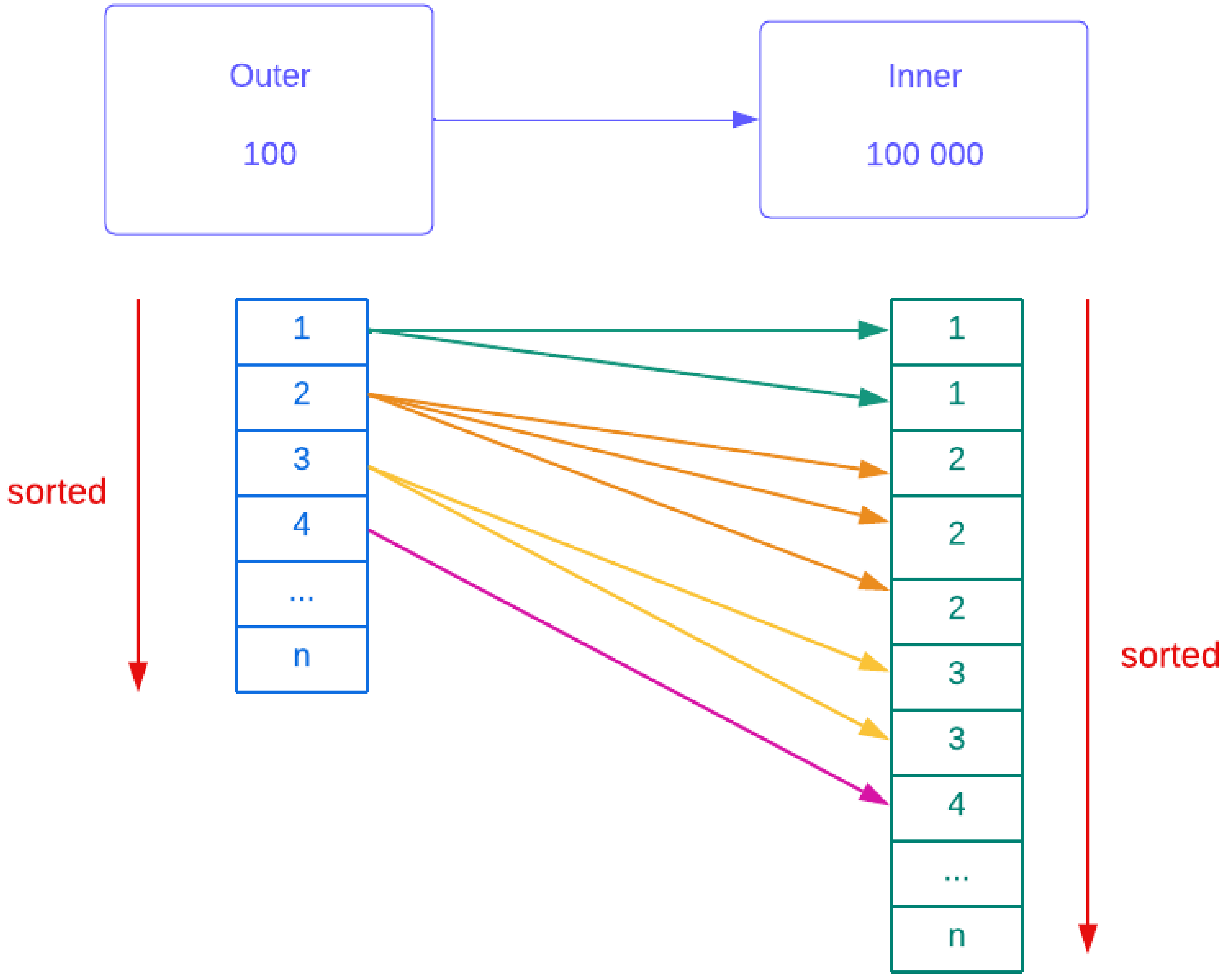
Hash join

```
EXPLAIN SELECT * FROM product p
        INNER JOIN category c ON
p.category_id = c.category_id;
-- or
EXPLAIN SELECT * FROM category c
        INNER JOIN product p ON
c.category_id = p.category_id;
-- same plan::
```



```
                              QUERY PLAN
-----------------------------------------------------------------------
 Hash Join  (cost=1.16..591.32 rows=10001 width=406)
   Hash Cond: (p.category_id = c.category_id)
   ->  Seq Scan on product p  (cost=0.00..548.01 rows=10001 width=319)
   ->  Hash  (cost=1.07..1.07 rows=7 width=87)
         ->  Seq Scan on category c  (cost=0.00..1.07 rows=7 width=87)
```

Merge join

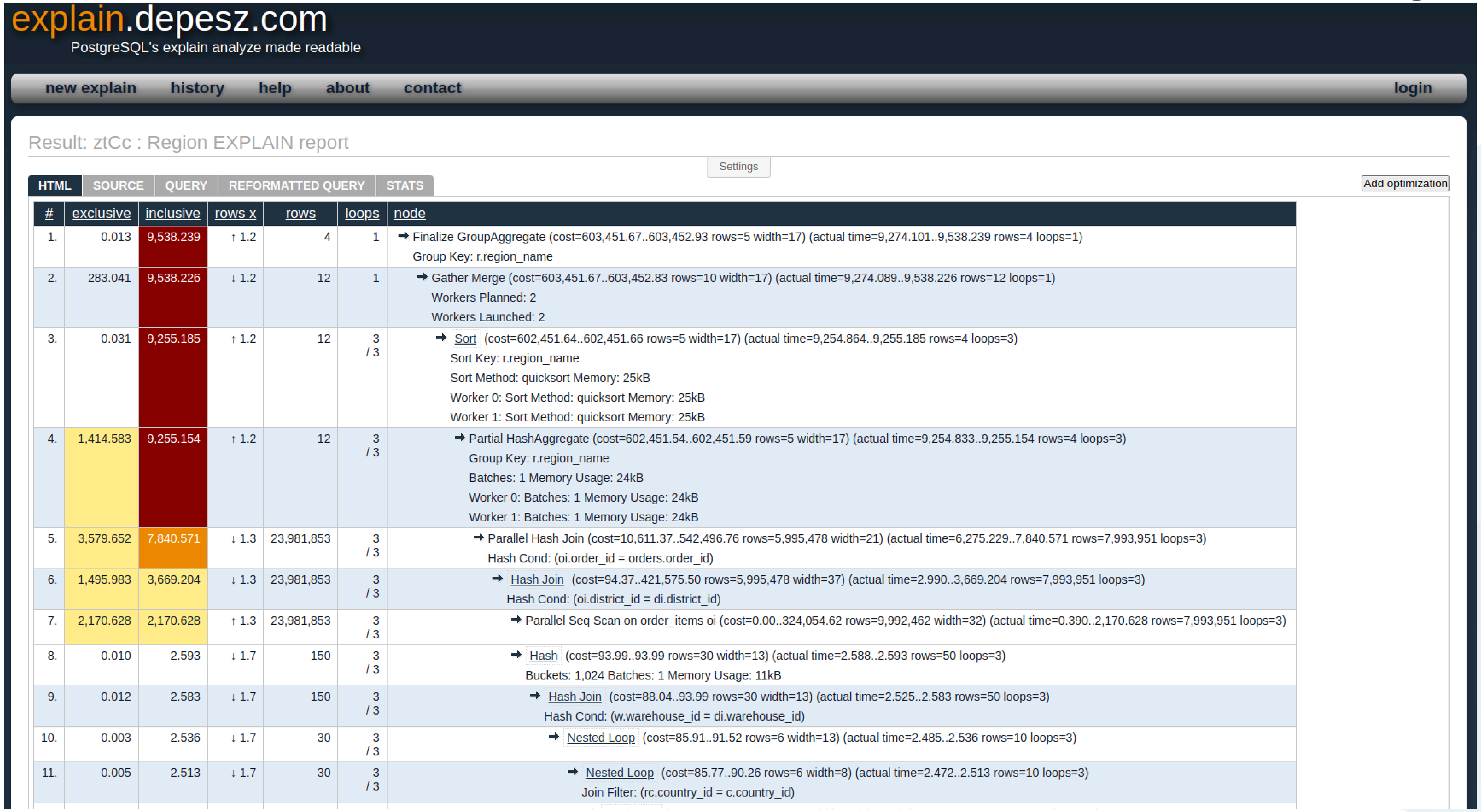


- Before comparing, sorted two list

EXPLAIN Tools

Wolfgres

explain.depesz.com → expert-friendly

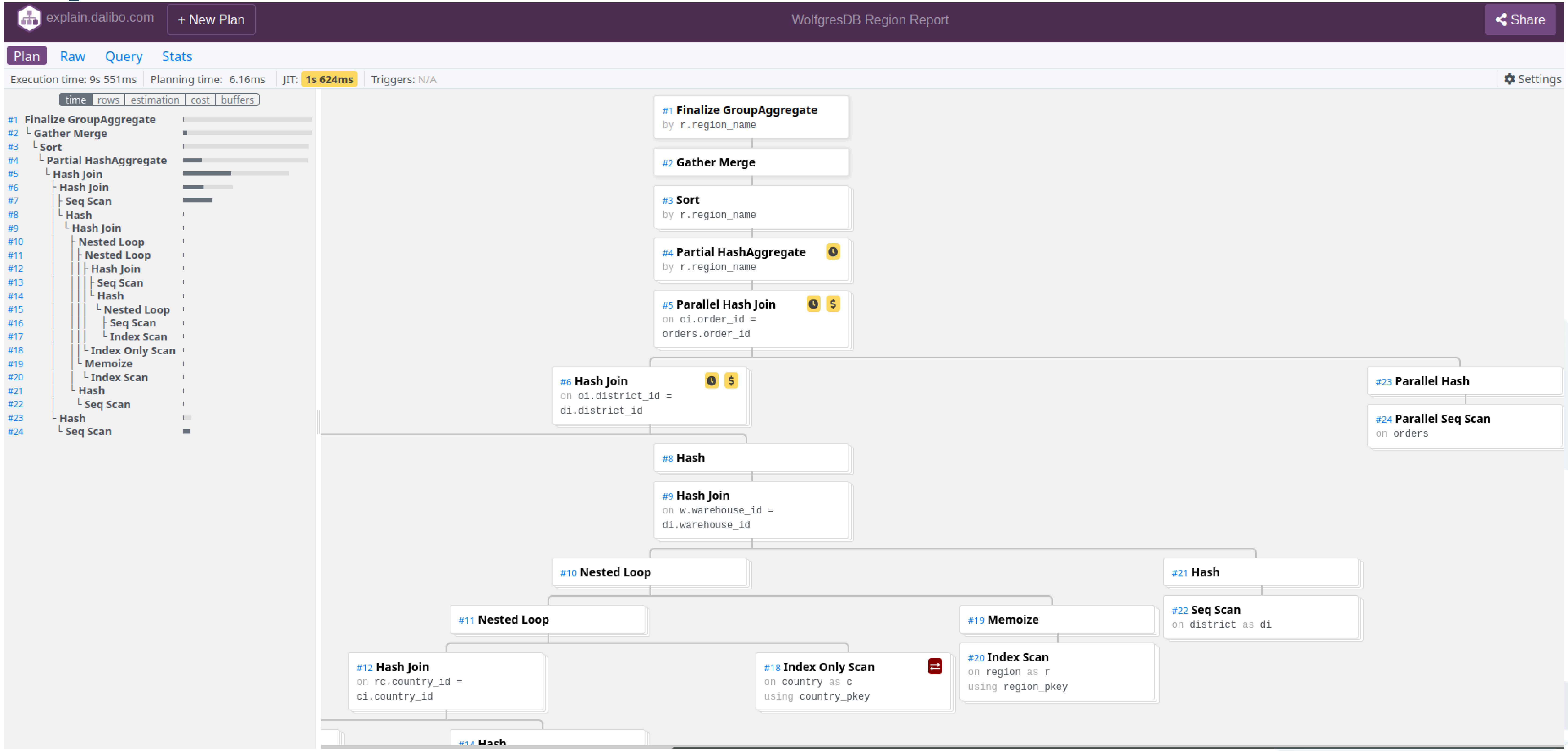


explain.depesz.com
PostgreSQL's explain analyze made readable

Result: ztCc : Region EXPLAIN report

#	exclusive	inclusive	rows x	rows	loops	node
1.	0.013	9,538.239	↑ 1.2	4	1	→ Finalize GroupAggregate (cost=603,451.67..603,452.93 rows=5 width=17) (actual time=9,274.101..9,538.239 rows=4 loops=1) Group Key: r.region_name
2.	283.041	9,538.226	↓ 1.2	12	1	→ Gather Merge (cost=603,451.67..603,452.83 rows=10 width=17) (actual time=9,274.089..9,538.226 rows=12 loops=1) Workers Planned: 2 Workers Launched: 2
3.	0.031	9,255.185	↑ 1.2	12	3 / 3	→ Sort (cost=602,451.64..602,451.66 rows=5 width=17) (actual time=9,254.864..9,255.185 rows=4 loops=3) Sort Key: r.region_name Sort Method: quicksort Memory: 25kB Worker 0: Sort Method: quicksort Memory: 25kB Worker 1: Sort Method: quicksort Memory: 25kB
4.	1,414.583	9,255.154	↑ 1.2	12	3 / 3	→ Partial HashAggregate (cost=602,451.54..602,451.59 rows=5 width=17) (actual time=9,254.833..9,255.154 rows=4 loops=3) Group Key: r.region_name Batches: 1 Memory Usage: 24kB Worker 0: Batches: 1 Memory Usage: 24kB Worker 1: Batches: 1 Memory Usage: 24kB
5.	3,579.652	7,840.571	↓ 1.3	23,981,853	3 / 3	→ Parallel Hash Join (cost=10,611.37..542,496.76 rows=5,995,478 width=21) (actual time=6,275.229..7,840.571 rows=7,993,951 loops=3) Hash Cond: (oi.order_id = orders.order_id)
6.	1,495.983	3,669.204	↓ 1.3	23,981,853	3 / 3	→ Hash Join (cost=94.37..421,575.50 rows=5,995,478 width=37) (actual time=2.990..3,669.204 rows=7,993,951 loops=3) Hash Cond: (oi.district_id = di.district_id)
7.	2,170.628	2,170.628	↑ 1.3	23,981,853	3 / 3	→ Parallel Seq Scan on order_items oi (cost=0.00..324,054.62 rows=9,992,462 width=32) (actual time=0.390..2,170.628 rows=7,993,951 loops=3)
8.	0.010	2.593	↓ 1.7	150	3 / 3	→ Hash (cost=93.99..93.99 rows=30 width=13) (actual time=2.588..2.593 rows=50 loops=3) Buckets: 1,024 Batches: 1 Memory Usage: 11kB
9.	0.012	2.583	↓ 1.7	150	3 / 3	→ Hash Join (cost=88.04..93.99 rows=30 width=13) (actual time=2.525..2.583 rows=50 loops=3) Hash Cond: (w.warehouse_id = di.warehouse_id)
10.	0.003	2.536	↓ 1.7	30	3 / 3	→ Nested Loop (cost=85.91..91.52 rows=6 width=13) (actual time=2.485..2.536 rows=10 loops=3)
11.	0.005	2.513	↓ 1.7	30	3 / 3	→ Nested Loop (cost=85.77..90.26 rows=6 width=8) (actual time=2.472..2.513 rows=10 loops=3) Join Filter: (rc.country_id = c.country_id)

explain.dalibo.com → visual, based on PEV



Execution time: 9s 551ms | Planning time: 6.16ms | JIT: 1s 624ms | Triggers: N/A

```
#1  Finalize GroupAggregate
#2  └ Gather Merge
#3    └ Sort
#4      └ Partial HashAggregate
#5        └ Hash Join
#6          ├ Hash Join
#7          │ ├ Seq Scan
#8          │ └ Hash
#9          │   └ Hash Join
#10         │     ├ Nested Loop
#11         │     │ ├ Nested Loop
#12         │     │ │ ├ Hash Join
#13         │     │ │ │ ├ Seq Scan
#14         │     │ │ │ └ Hash
#15         │     │ │ │   └ Nested Loop
#16         │     │ │ │     ├ Seq Scan
#17         │     │ │ │     └ Index Scan
#18         │     │ │ └ Index Only Scan
#19         │     │ └ Memoize
#20         │     │   └ Index Scan
#21         │     └ Hash
#22         │       └ Seq Scan
#23         └ Hash
#24           └ Seq Scan
```

#1 **Finalize GroupAggregate**
by `r.region_name`

#2 **Gather Merge**

#3 **Sort**
by `r.region_name`

#4 **Partial HashAggregate**
by `r.region_name`

#5 **Parallel Hash Join**
on `oi.order_id = orders.order_id`

#6 **Hash Join**
on `oi.district_id = di.district_id`

#23 **Parallel Hash**

#24 **Parallel Seq Scan**
on `orders`

#8 **Hash**

#9 **Hash Join**
on `w.warehouse_id = di.warehouse_id`

#10 **Nested Loop**

#21 **Hash**

#11 **Nested Loop**

#19 **Memoize**

#22 **Seq Scan**
on `district as di`

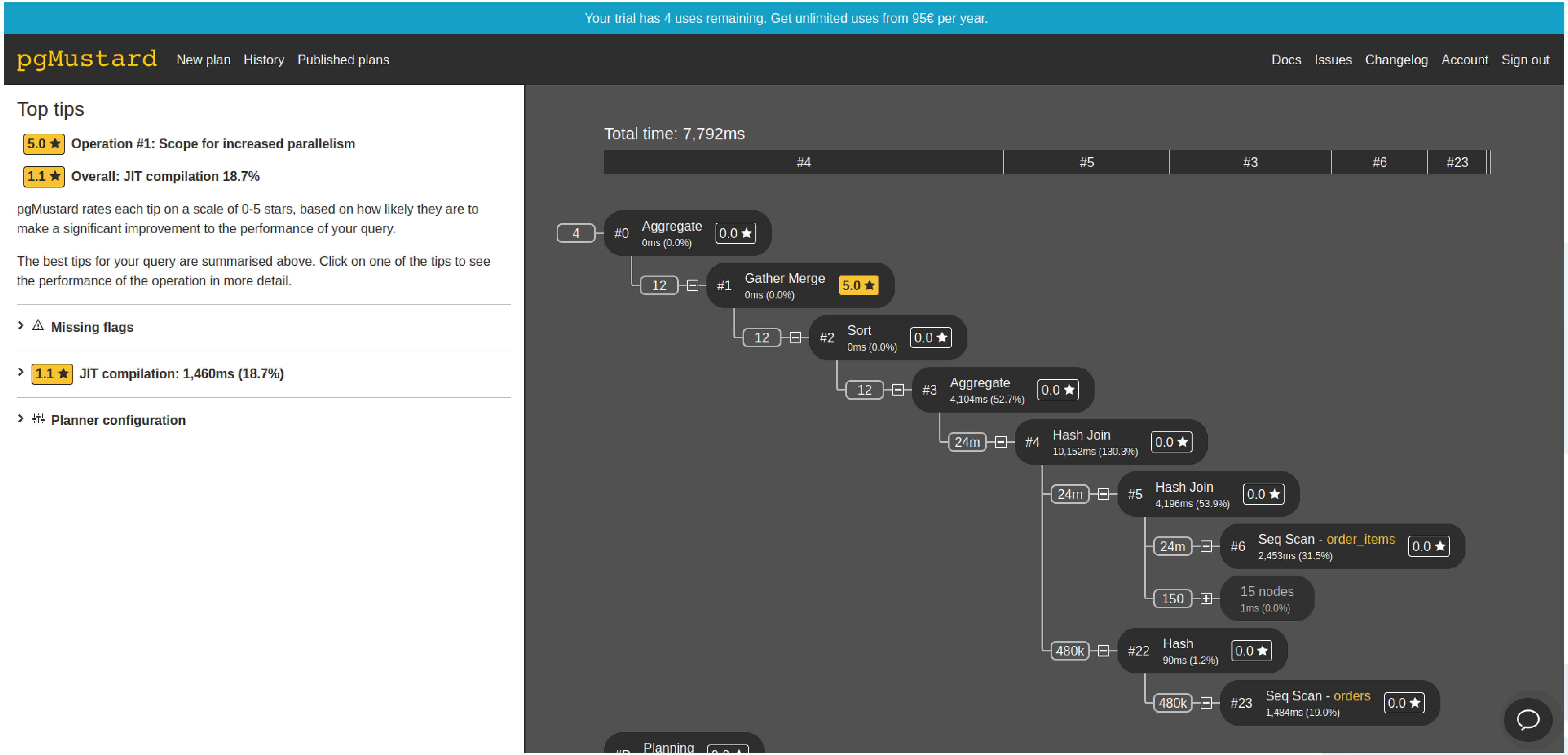
#12 **Hash Join**
on `rc.country_id = ci.country_id`

#18 **Index Only Scan**
on `country as c`
using `country_pkey`

#20 **Index Scan**
on `region as r`
using `region_pkey`

#14 **Hash**

pgmustard.com → beginner-friendly



Your trial has 4 uses remaining. Get unlimited uses from 95€ per year.

pgMustard   New plan   History   Published plans   Docs   Issues   Changelog   Account   Sign out

Top tips

5.0 ★ **Operation #1: Scope for increased parallelism**

1.1 ★ **Overall: JIT compilation 18.7%**

pgMustard rates each tip on a scale of 0-5 stars, based on how likely they are to make a significant improvement to the performance of your query.

The best tips for your query are summarised above. Click on one of the tips to see the performance of the operation in more detail.

› ⚠ **Missing flags**

› 1.1 ★ **JIT compilation: 1,460ms (18.7%)**

› **Planner configuration**

Total time: 7,792ms

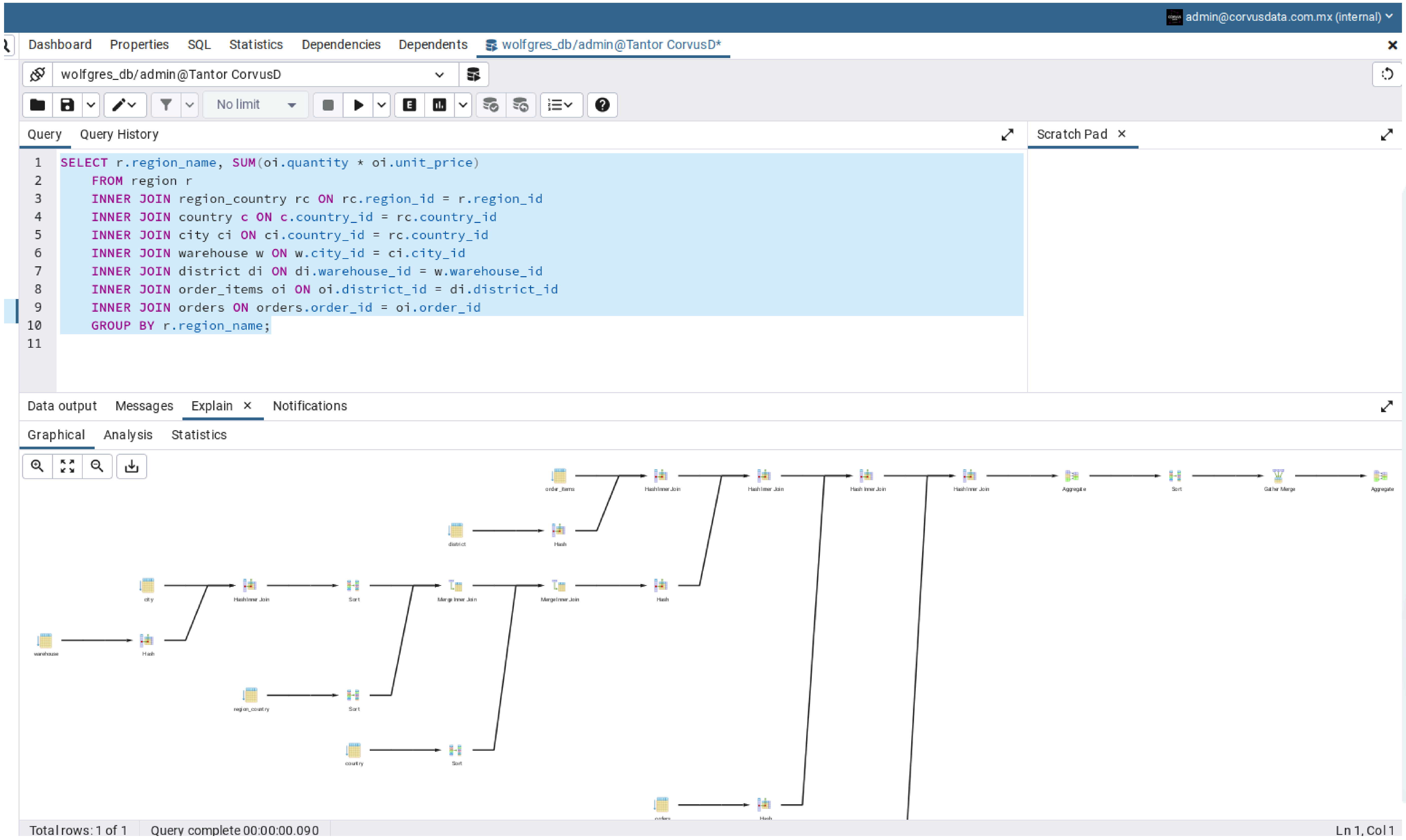
#4 | #5 | #3 | #6 | #23

```
4    #0  Aggregate 0ms (0.0%)                             0.0 ★
  12   #1  Gather Merge 0ms (0.0%)                        5.0 ★
    12   #2  Sort 0ms (0.0%)                              0.0 ★
      12   #3  Aggregate 4,104ms (52.7%)                  0.0 ★
        24m  #4  Hash Join 10,152ms (130.3%)              0.0 ★
          24m  #5  Hash Join 4,196ms (53.9%)              0.0 ★
            24m  #6  Seq Scan - order_items 2,453ms (31.5%)  0.0 ★
            150  15 nodes 1ms (0.0%)
          480k #22 Hash 90ms (1.2%)                       0.0 ★
            480k #23 Seq Scan - orders 1,484ms (19.0%)    0.0 ★
Planning
```

https://app.pgmustard.com#/explore/46dcf0f4-264f-4b1f-9115-32521992dc89

Wolfgres

pgAdmin → I think you need expert



```
SELECT r.region_name, SUM(oi.quantity * oi.unit_price)
    FROM region r
    INNER JOIN region_country rc ON rc.region_id = r.region_id
    INNER JOIN country c ON c.country_id = rc.country_id
    INNER JOIN city ci ON ci.country_id = rc.country_id
    INNER JOIN warehouse w ON w.city_id = ci.city_id
    INNER JOIN district di ON di.warehouse_id = w.warehouse_id
    INNER JOIN order_items oi ON oi.district_id = di.district_id
    INNER JOIN orders ON orders.order_id = oi.order_id
    GROUP BY r.region_name;
```

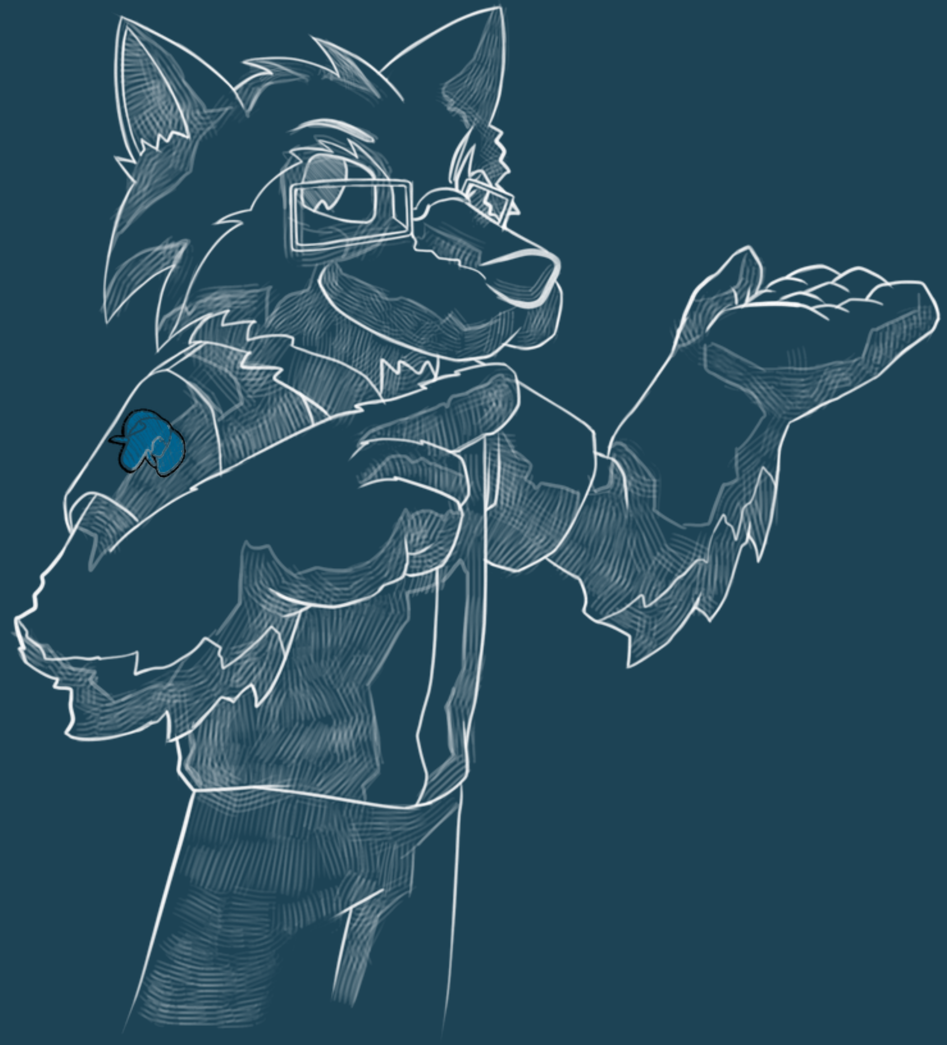
Total rows: 1 of 1   Query complete 00:00:00.090   Ln 1, Col 1

Learn more...

- Explaining EXPLAIN - Josh Berkus
  - https://www.youtube.com/watch?v=ZOZglRUjfFI
- A beginners guide to EXPLAIN ANALYZE – Michael Christofide
  - https://www.youtube.com/watch?v=31EmOKBP1PY
- Explaining the Postgres Query Optimizer – Bruce Momjian -
  - https://www.youtube.com/watch?v=wLpcVM9qxV0
- Just follow to @samokhvalov share a lot great hacks in PostgreSQL
  - https://x.com/samokhvalov/status/1767642882419368411

Wolfgres

More links!



- https://www.pgmustard.com/docs/explain
- https://www.cybertec-postgresql.com/en/how-to-interpret-postgresql-explain-analyze-output/
- https://docs.gitlab.com/ee/development/understanding_explain_plans.html
- https://www.depesz.com/2013/04/16/explaining-the-unexplainable/

Questions

All questions will be answer by email... ;D

THANK YOU

Wolfgres

All the drawings were made by Uriel Vazquez.
di.uriel.vs@gmail.com

THANK YOU

Wolfgres

How to understand EXPLAIN without dying in the attempt

Alfredo Rodríguez

@AlfredoDrakoRod

Wolfgres