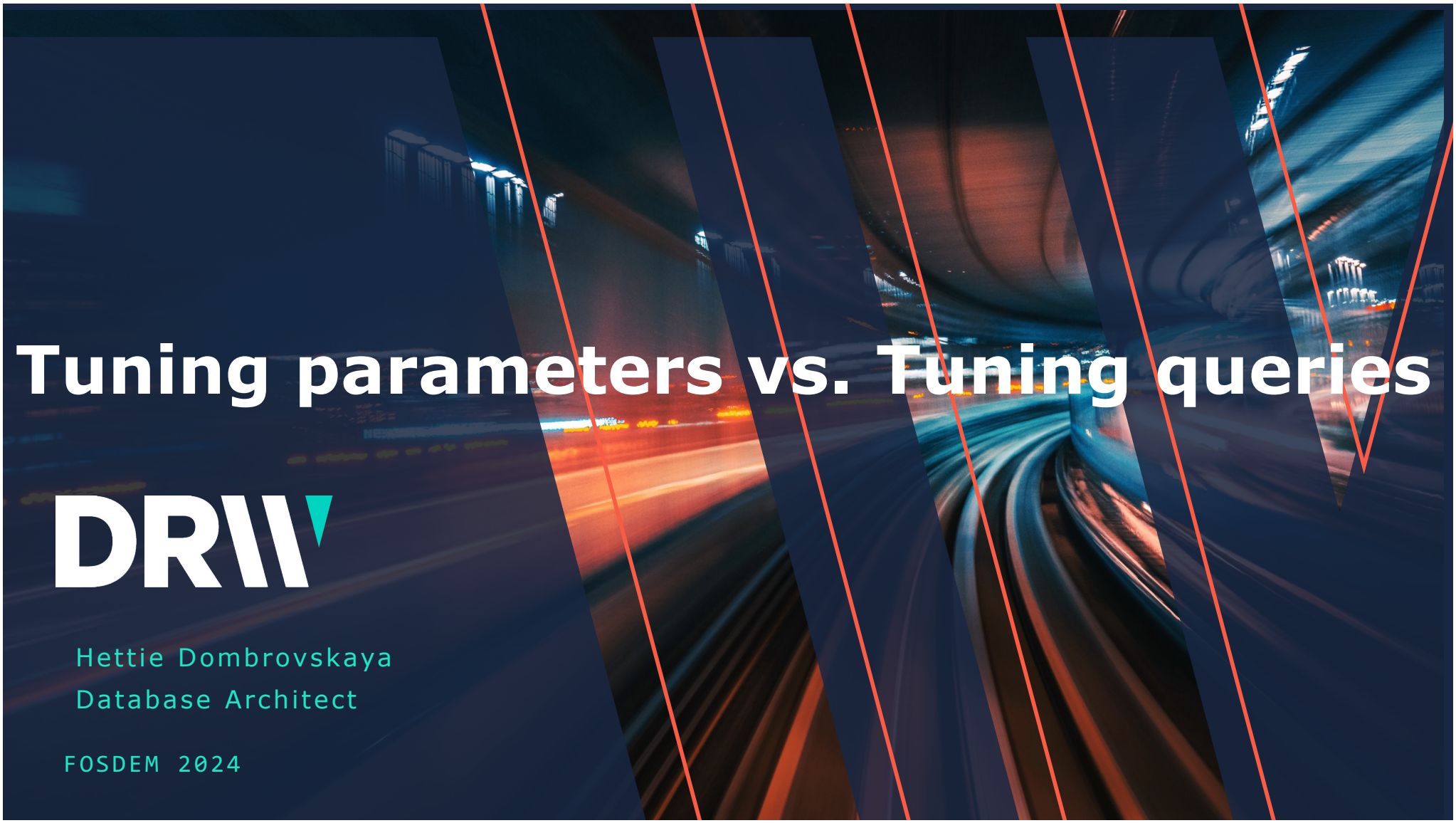


Tuning parameters vs. Tuning queries

DRW

Hettie Dombrovskaya
Database Architect

FOSDEM 2024

Who Am I

Database Architect at DRW
Local Organizer of Chicago PostgreSQL User Group

PG Day Chicago is on April 26, 2024!

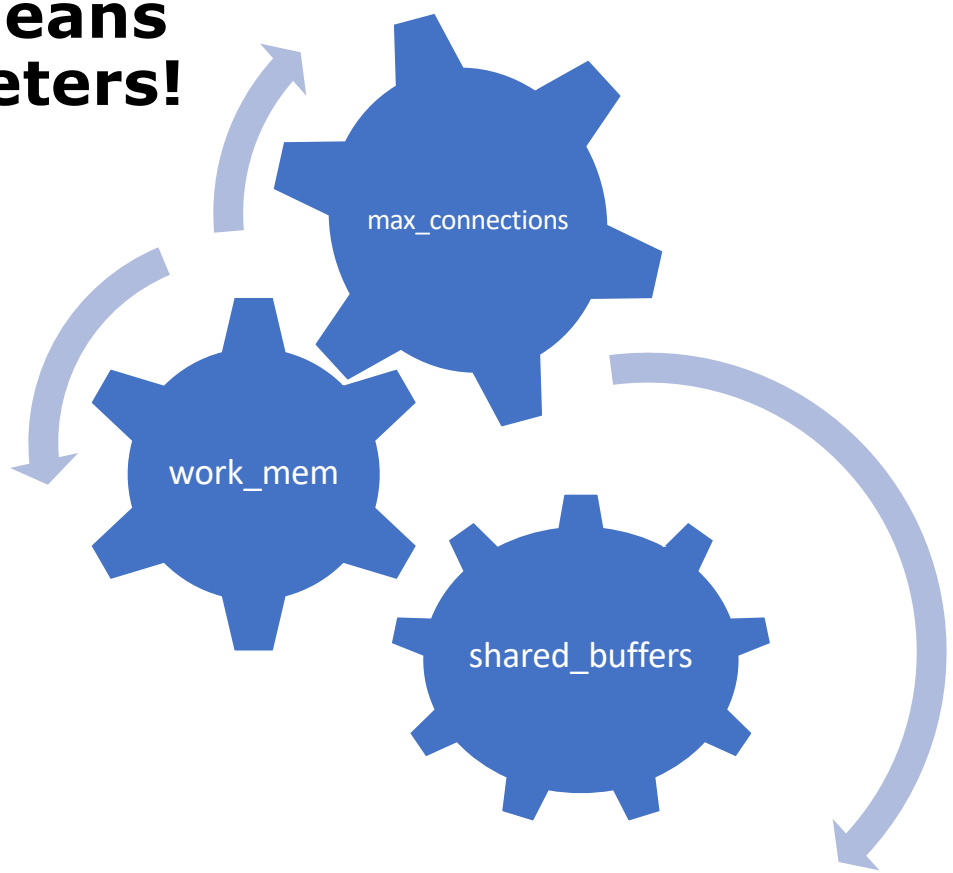
I never presented any talk about tuning parameters -why now?!

Tuning your database – what does it mean?

I thought I knew…

Until I started working for EDB!

And then it turned out that *tuning* means tuning parameters!



Why I never did it before?

Why people believe in the magic of parameters?

My goal today is to show
Why it ~~almost~~ does not matter

Why it is difficult to show?

- Tuning individual queries performance vs improving throughput
- It’s difficult to model a real-life workload
- It’s difficult to model a real-life concurrency

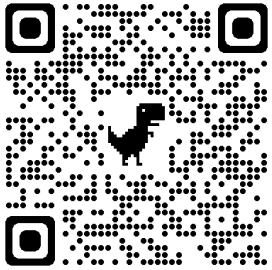
Facts

Tuning parameters can improve performance 10%, 20%, in some cases 50%

Tuning queries can improve performance several (tens) times

Tuning application can improve performance up to hundreds of time!

Query example



https://github.com/hettie-d/postgres_air

```
SELECT f.flight_no,
       f.actual_departure,
       count(passenger_id) passengers
  FROM flight f
       JOIN booking_leg bl ON bl.flight_id = f.flight_id
       JOIN passenger p ON p.booking_id=bl.booking_id
 WHERE f.departure_airport = 'JFK'
   AND f.arrival_airport = 'ORD'
   AND f.actual_departure BETWEEN
        '2023-08-08' and '2023-08-12'
GROUP BY f.flight_id, f.actual_departure;
```

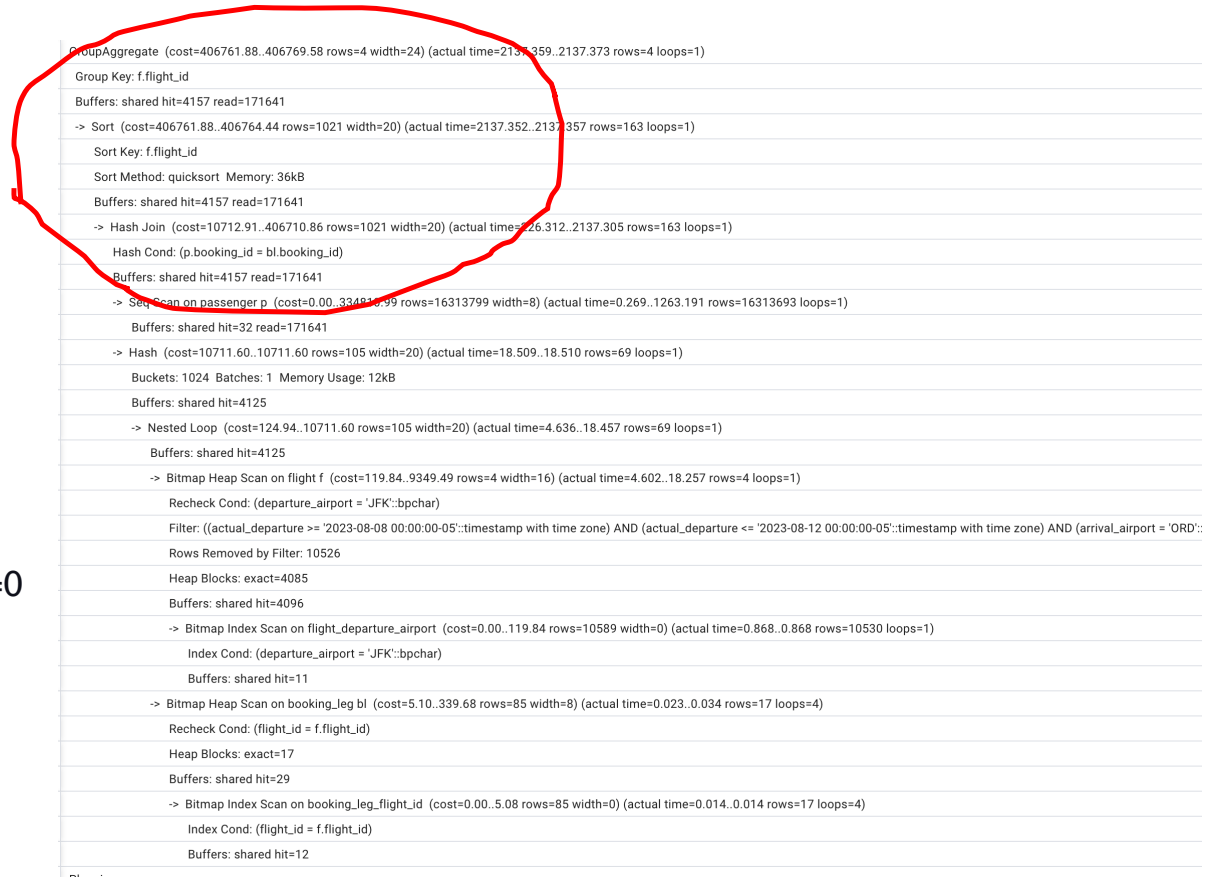
Execution plan with default memory allocation

`shared_buffers=128MB`

`work_mem=4MB`

`max_parallel_workers_per_gather=0`

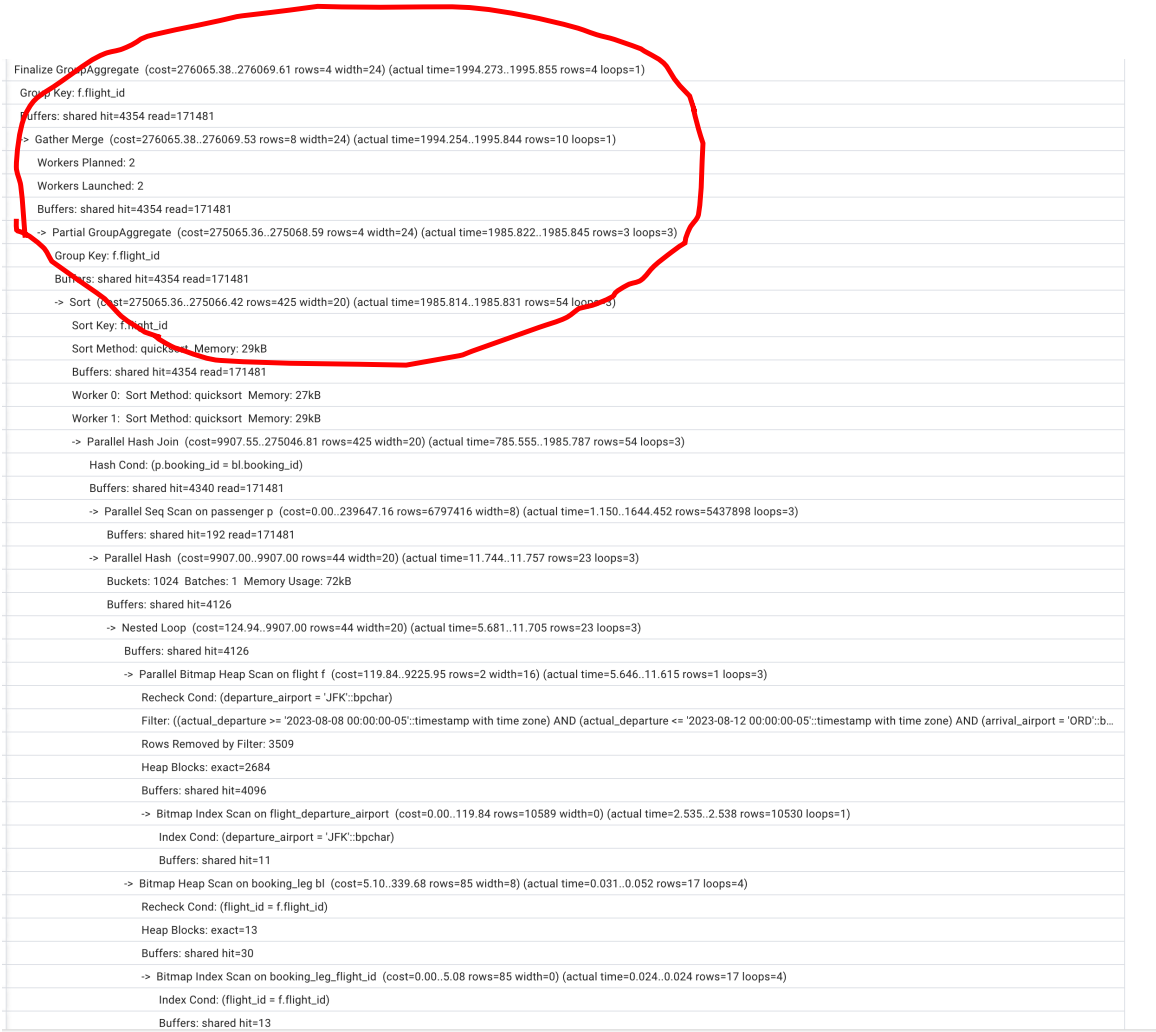
Execution time: 2.4 s



Execution plan with default memory allocation

```
shared_buffers=128MB
work_mem=4MB
max_parallel_workers_per_gather=2
```

Execution time 2.1 s



```
-> Gather Merge  (cost=276065.38..276069.53 rows=8 width=24) (actual time=1726.858..1727.739 rows=11 loops=1)
    Workers Planned: 2
    Workers Launched: 2
    Buffers: shared hit=4546 read=171289
    -> Partial GroupAggregate  (cost=275065.36..275068.59 rows=4 width=24) (actual time=1722.810..1722.816 rows=4 loops=3)
        Group Key: f.flight_id
        Buffers: shared hit=4546 read=171289
        -> Sort  (cost=275065.36..275066.42 rows=425 width=20) (actual time=1722.803..1722.806 rows=54 loops=3)
            Sort Key: f.flight_id
```

Increasing work_mem

```
shared_buffers=128MB
max_parallel_workers_per_gather=2
```

work_mem =500MB

1.8 sec

work_mem =1GB

1.8 sec

Increasing shared buffers (requires restart)

```
shared_buffers=1GB
work_mem=4MB/100MB/500MB

1s


shared_buffers=2GB
work_mem=500MB

1.1s
```

Let’s try something different!

Let’s take a closer look at the execution plans we have so far

Heap scan when looking for the departure dates between August 8 and 12.

Build the index

```
CREATE INDEX
flight_actual_departure
    ON postgres_air.flight
    (actual_departure);
```

Execution time: 0.7 s

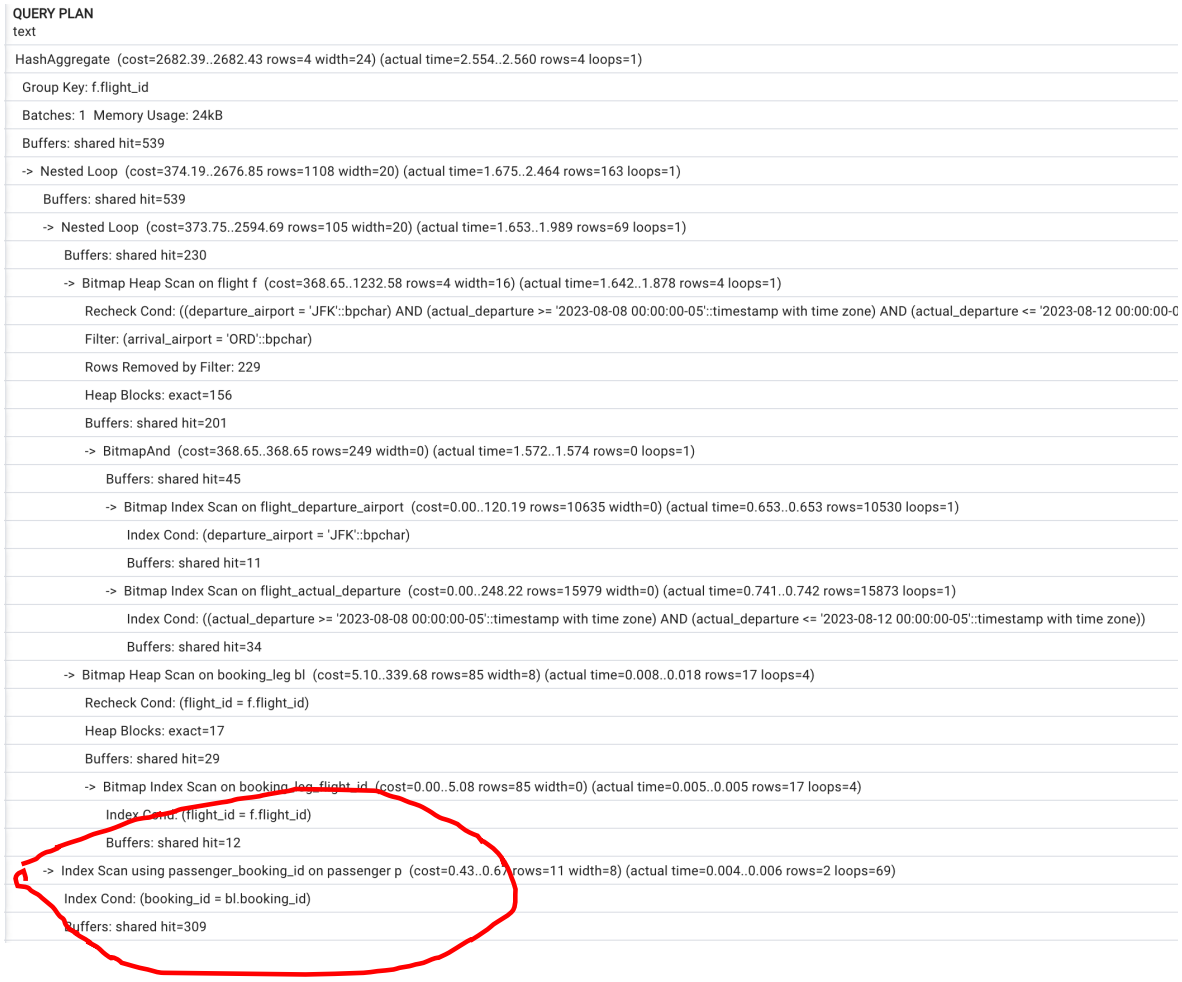
```
-> Bitmap Heap Scan on flight f  (cost=368.65..1232.58 rows=4 width=16) (actual time=2.200..2.483 rows=4 loops=3)
    Recheck Cond: ((departure_airport = 'JFK'::bpchar) AND (actual_departure >= '2023-08-08 00:00:00-05'::timestamp with time zone) AND (actual_departure <= '2023-08-12 00:00:00-05'::tim
    Filter: (arrival_airport = 'ORD'::bpchar)
    Rows Removed by Filter: 229
    Heap Blocks: exact=156
    Buffers: shared hit=607
    -> BitmapAnd  (cost=368.65..368.65 rows=249 width=0) (actual time=2.092..2.093 rows=0 loops=3)
        Buffers: shared hit=139
        -> Bitmap Index Scan on flight_departure_airport  (cost=0.00..120.19 rows=10635 width=0) (actual time=0.973..0.973 rows=10530 loops=3)
            Index Cond: (departure_airport = 'JFK'::bpchar)
            Buffers: shared hit=35
        -> Bitmap Index Scan on flight_actual_departure  (cost=0.00..248.22 rows=15979 width=0) (actual time=0.905..0.905 rows=15873 loops=3)
            Index Cond: ((actual_departure >= '2023-08-08 00:00:00-05'::timestamp with time zone) AND (actual_departure <= '2023-08-12 00:00:00-05'::timestamp with time zone))
```

```
-> Hash Join  (cost=2243.76..267385.35 rows=473 width=20) (actual time=490.297..2186.655 rows=54 loops=3)
    Hash Cond: (p.booking_id = bl.booking_id)
    Buffers: shared hit=498 read=171481
    -> Parallel Seq Scan on passenger p  (cost=0.00..239646.72 rows=6797372 width=8) (actual time=0.521..1941.157 rows=5437898 loops=3)
        Buffers: shared hit=192 read=171481
    -> Hash  (cost=2242.12..2242.12 rows=131 width=20) (actual time=4.722..4.725 rows=69 loops=3)
        Buckets: 1024  Batches: 1  Memory Usage: 12kB
        Buffers: shared hit=284
```

Build another index!

```
CREATE INDEX
passenger_booking_id
    ON postgres_air.passenger
    (booking_id);
```

Execution time: 60 ms



```
QUERY PLAN
text
HashAggregate  (cost=2682.39..2682.43 rows=4 width=24) (actual time=2.554..2.560 rows=4 loops=1)
  Group Key: f.flight_id
  Batches: 1  Memory Usage: 24kB
  Buffers: shared hit=539
  ->  Nested Loop  (cost=374.19..2676.85 rows=1108 width=20) (actual time=1.675..2.464 rows=163 loops=1)
        Buffers: shared hit=539
        ->  Nested Loop  (cost=373.75..2594.69 rows=105 width=20) (actual time=1.653..1.989 rows=69 loops=1)
              Buffers: shared hit=230
              ->  Bitmap Heap Scan on flight f  (cost=368.65..1232.58 rows=4 width=16) (actual time=1.642..1.878 rows=4 loops=1)
                    Recheck Cond: ((departure_airport = 'JFK'::bpchar) AND (actual_departure >= '2023-08-08 00:00:00-05'::timestamp with time zone) AND (actual_departure <= '2023-08-12 00:00:00-0
                    Filter: (arrival_airport = 'ORD'::bpchar)
                    Rows Removed by Filter: 229
                    Heap Blocks: exact=156
                    Buffers: shared hit=201
                    ->  BitmapAnd  (cost=368.65..368.65 rows=249 width=0) (actual time=1.572..1.574 rows=0 loops=1)
                          Buffers: shared hit=45
                          ->  Bitmap Index Scan on flight_departure_airport  (cost=0.00..120.19 rows=10635 width=0) (actual time=0.653..0.653 rows=10530 loops=1)
                                Index Cond: (departure_airport = 'JFK'::bpchar)
                                Buffers: shared hit=11
                          ->  Bitmap Index Scan on flight_actual_departure  (cost=0.00..248.22 rows=15979 width=0) (actual time=0.741..0.742 rows=15873 loops=1)
                                Index Cond: ((actual_departure >= '2023-08-08 00:00:00-05'::timestamp with time zone) AND (actual_departure <= '2023-08-12 00:00:00-05'::timestamp with time zone))
                                Buffers: shared hit=34
              ->  Bitmap Heap Scan on booking_leg bl  (cost=5.10..339.68 rows=85 width=8) (actual time=0.008..0.018 rows=17 loops=4)
                    Recheck Cond: (flight_id = f.flight_id)
                    Heap Blocks: exact=17
                    Buffers: shared hit=29
                    ->  Bitmap Index Scan on booking_leg_flight_id  (cost=0.00..5.08 rows=85 width=0) (actual time=0.005..0.005 rows=17 loops=4)
                          Index Cond: (flight_id = f.flight_id)
                          Buffers: shared hit=12
        ->  Index Scan using passenger_booking_id on passenger p  (cost=0.43..0.67 rows=11 width=8) (actual time=0.004..0.006 rows=2 loops=69)
              Index Cond: (booking_id = bl.booking_id)
              Buffers: shared hit=309
```

What will happen if we return parameters back to default?

DRW

The execution plan will remain the same (and the execution speed as well!)

Understanding the role of parameters

Communicating the hardware characteristics to PostgreSQL

Examples:

- RAM 16 GB/ shared_buffers 128MB
- RAM 16 GB/shared_buffers 4 GB/ work_mem 200MB/max_connecitons 1000
- random_page_cost 4

Application changes

Not necessarily NORM!

Examples:

- Using ‘=‘ instead of ‘~’
- Column transform:
  trunc(created_dt)=CURRENT_DATE
- Committing each record
- Not committing until the batch end

Q&A

Hettie Dombrovskaya
Database Architect DRW

hdombrovska@drwholdings.com

www.drw.com