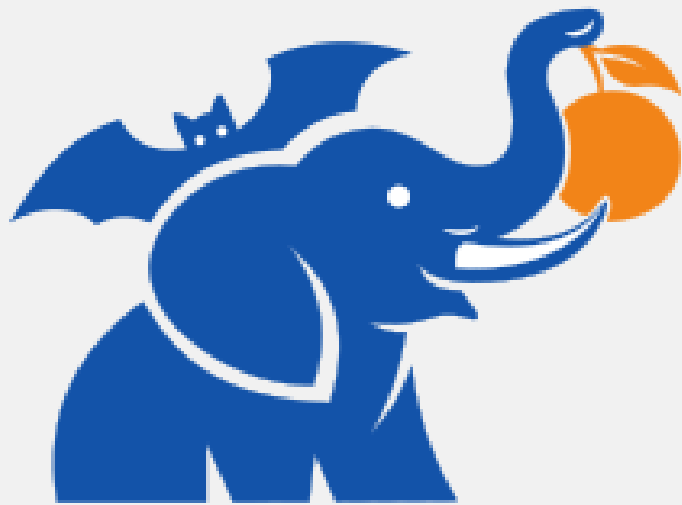




PostgreSQL Conference Europe Valencia, 20-23 October 2026

<https://2026.pgconf.eu/>



A composition of short items (or: a talk for the tiktok generation)

Ellert van Koperen

Audit 1

Thursday, October 22 at 13:15 - 13:45

Ego stuff



Ellert van Koperen

- Freelancing since 2014
- Dedicated database nerd since 2019, specialized in **PostgreSQL** as ellert@database.one
- Find me on linkedin or here at the conference



Dealing with GEO-Spatial data

Data of those objects is stored in a table possibly like so:

```
create table things (  
  id          bigserial primary key,  
  name        text,  
  ...  
  latitude    numeric,  
  longitude    numeric  
);
```

Retrieving data naively

```
select * from things
where
    latitude between 52.155 and 52.29583
and longitude between 5.3875 and 5.1625;
```

And you'll get nothing

Retrieving data naively

```
select * from things
where
    latitude between 52.155 and 52.29583
and longitude between 5.1625 and 5.3875;
```

This still sucks

Let's do it ourselves

```
create table things (  
  id          bigserial primary key,  
  ...         ...,  
  latitude    numeric,  
  longitude   numeric  
);
```

Coordinates
stored here



```
alter table things add column coordinates point;  
alter table things alter column coordinates  
  set data type point using point(latitude, longitude);
```

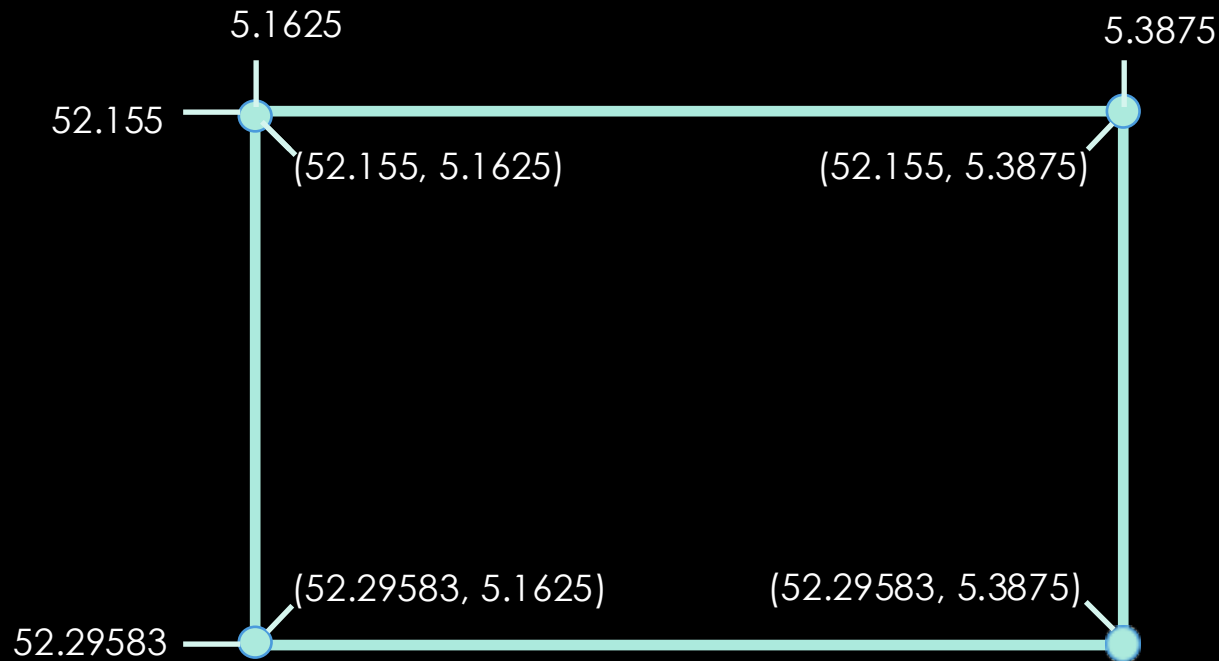
```
create index concurrently things_coordinates_idx  
  on things using GIST(coordinates);
```

Convert to point
type and maintain
with e.g. trigger

Add
multidimensional-
enabled index

And we can finally get rid of the betweens

```
select * from things where coordinates <@  
box( point(52.155, 5.3875), point(52.29583, 5.1625));
```



Dealing with GEO-Spatial data

More juicy tidbits on database.one/blog
or talk to me directly



One troublesome query

One troublesome query

Pg_stat_statements on production: it takes about **8.656 milliseconds** for an average execution of this query.

It is not too complex and on a relatively small table (about 13.000 rows), but it runs quite often (**359.513.795** in one year), and is causing a considerable amount of the system's overall load.

The query can not be changed.

```
SELECT IDX FROM PARAMETERINFO
WHERE (NAME = $1 OR
      (LOWER(NAME) = LOWER($1) AND FLAGS & x'80'::int4 > 0)
)
AND FLAGS & x'20'::int4 > 0;
```

The queryplan (on local test)

```
Seq Scan on parameterinfo (cost=0.00..535.30 rows=8 width=4) (actual  
time=7.324..7.324 rows=0 loops=1)  
  Filter: (((flags & '32'::bigint) > 0) AND (((name)::text = 'Blah'::text) OR  
((lower((name)::text) = 'Blah'::text) AND ((flags & '128'::bigint) > 0))))  
    Rows Removed by Filter: 13029  
Planning Time: 0.090 ms  
Execution Time: 7.337 ms
```

Better readable

```
SELECT IDX FROM PARAMETERINFO
WHERE (NAME = $1 OR
      (LOWER(NAME) = LOWER($1) AND FLAGS & x'80'::int4 > 0))
      AND FLAGS & x'20'::int4 > 0;
```

```
SELECT IDX FROM PARAMETERINFO
WHERE NAME = $1 AND FLAGS & x'20'::int4 > 0
```

UNION

```
SELECT IDX FROM PARAMETERINFO
WHERE LOWER(NAME) = LOWER($1) AND FLAGS & x'80'::int4 > 0
      AND FLAGS & x'20'::int4 > 0;
```

First index : functional or expression

```
create index concurrently on PARAMETERINFO (NAME);  
create index concurrently on PARAMETERINFO (LOWER(NAME));  
analyze table PARAMETERINFO;
```

```
SELECT IDX FROM PARAMETERINFO  
WHERE NAME = $1 AND FLAGS & x'20'::int4 > 0
```

UNION

```
SELECT IDX FROM PARAMETERINFO  
WHERE LOWER(NAME) = LOWER($1) AND FLAGS & x'80'::int4 > 0  
    AND FLAGS & x'20'::int4 > 0;
```

Second index : partial

```
create index concurrently on PARAMETERINFO (name) where (FLAGS & x'20'::int4) > 0 ;
```

```
create index concurrently on PARAMETERINFO (LOWER(NAME)) where (FLAGS & x'80'::int4) > 0 and (FLAGS & x'20'::int4) > 0 ;
```

```
analyze table PARAMETERINFO;
```

```
SELECT IDX FROM PARAMETERINFO  
WHERE NAME = $1 AND FLAGS & x'20'::int4 > 0  
UNION  
SELECT IDX FROM PARAMETERINFO  
WHERE LOWER(NAME) = LOWER($1) AND FLAGS & x'80'::int4 > 0  
AND FLAGS & x'20'::int4 > 0;
```

Third index : hash

```
create index concurrently on PARAMETERINFO using hash(NAME) where  
(FLAGS & x'20'::int4) > 0 ;  
  
create index concurrently on parameterinfo using hash(LOWER(NAME))  
where (FLAGS & x'80'::int4) > 0 and (FLAGS & x'20'::int4) > 0 ;  
  
analyze table PARAMETERINFO;
```

```
SELECT IDX FROM PARAMETERINFO  
WHERE NAME = $1 AND FLAGS & x'20'::int4 > 0  
UNION  
SELECT IDX FROM PARAMETERINFO  
WHERE LOWER(NAME) = LOWER($1) AND FLAGS & x'80'::int4 > 0  
AND FLAGS & x'20'::int4 > 0;
```

Performance

Bitmap Heap Scan on parameterinfo (cost=8.06..35.18 rows=8 width=4) (actual time=0.013..0.014 rows=0 loops=1)

Recheck Cond: (((name)::text = 'Blah'::text) AND ((flags & '32'::bigint) > 0)) OR ((lower((name)::text) = 'blah'::text) AND ((flags & '128'::bigint) > 0) AND ((flags & '32'::bigint) > 0))

-> BitmapOr (cost=8.06..8.06 rows=8 width=0) (actual time=0.011..0.011 rows=0 loops=1)

-> Bitmap Index Scan on parameterinfo_name_idx (cost=0.00..4.01 rows=1 width=0) (actual time=0.008..0.008 rows=0 loops=1)

Index Cond: ((name)::text = 'Blah'::text)

-> Bitmap Index Scan on parameterinfo_lower_idx1 (cost=0.00..4.05 rows=7 width=0) (actual time=0.003..0.003 rows=0 loops=1)

Index Cond: (lower((name)::text) = 'blah'::text)

Planning Time: 0.101 ms

Execution Time: 0.035 ms

Planning Time: 0.090 ms

Execution Time: 7.337 ms

← Original was 55x slower



One little index

More juicy tidbits on database.one/blog
or talk to me directly



PAGING

Method 3

Get a **chunk**
per request



```
select * from postgres_air.boarding_pass limit 1;
```

pass_id	passenger_id	booking_leg_id	seat	boarding_time
27467390	18250564	18114010	10E	2024-05-28 07:25:00+00

```
select count(1) from postgres_air.boarding_pass;
```

30195223 = ± 30M

```
select boarding_time from postgres_air.boarding_pass order by
boarding_time asc limit 1;
```

2024-05-26 18:55:00.000000 +00

```
select boarding_time from postgres_air.boarding_pass order by
boarding_time desc limit 1;
```

2024-08-13 14:55:00.000000 +00

BEFORE WE START

```
create index concurrently if not exists  
boarding_pass_boarding_time_idx on  
postgres_air.boarding_pass(boarding_time);
```

```
select count(1) from postgres_air.boarding_pass  
where date(boarding_time) = '2024-06-01'::date;  
-- 393588 (0.4M) , count took 7.3s
```

GET A CHUNK

```
select * from postgres_air.boarding_pass
  where date(boarding_time) = '2024-06-01'::date
 limit 100;
-- get first chunk, this is fast-ish, execution 50ms
```

GET ANOTHER CHUNK

```
select * from postgres_air.boarding_pass
  where date(boarding_time) = '2024-06-01'::date
 limit 100;
-- get first chunk, this is fast-ish, execution 50ms
```

```
select * from postgres_air.boarding_pass
  where date(boarding_time) = '2024-06-01'::date
 offset 346600 limit 100;
-- get way further chunk, that is much much slower
execution at 3s 882ms
```

GET ANOTHER CHUNK

```
select * from postgres_air.boarding_pass
  where date(boarding_time) = '2024-06-01'::date
 limit 100;
```

-- get first chunk, this is fast-ish, execution 50ms

```
select * from postgres_air.boarding_pass
  where date(boarding_time) = '2024-06-01'::date
 offset 346600 limit 100;
```

-- get way further chunk, that is much much slower
execution at 3s 882ms

YOU IDIOT!

GET ANOTHER CHUNK

```
select * from postgres_air.boarding_pass
  where boarding_time >= '2024-06-01'::timestamptz
        and boarding_time < '2024-06-02'::timestamptz
 limit 100;
-- get first chunk, this is still ok, execution 14ms
```

```
select * from postgres_air.boarding_pass
  where boarding_time >= '2024-06-01'::timestamptz
        and boarding_time < '2024-06-02'::timestamptz
 offset 346600 limit 100;
-- get way further chunk, better but still much slower
execution at 438ms
```

GET ANOTHER CHUNK

```
select * from postgres_air.boarding_pass
  where boarding_time >= '2024-06-01'::timestamptz
        and boarding_time < '2024-06-02'::timestamptz
 order by boarding_time, pass_id limit 100;
-- get first chunk, this is still ok, execution 17ms
```

```
select * from postgres_air.boarding_pass
  where boarding_time >= '2024-06-01'::timestamptz
        and boarding_time < '2024-06-02'::timestamptz
 order by boarding_time, pass_id offset 346600 limit 100;
-- get way further chunk, but meh with offset, execution
at 232ms
```

WHY IS THIS STILL SLOW ?

```
select * from postgres_air.boarding_pass
where boarding_time >= '2024-06-01'::timestamptz
      and boarding_time < '2024-06-02'::timestamptz
order by boarding_time, pass_id offset 346600 limit 100;
```

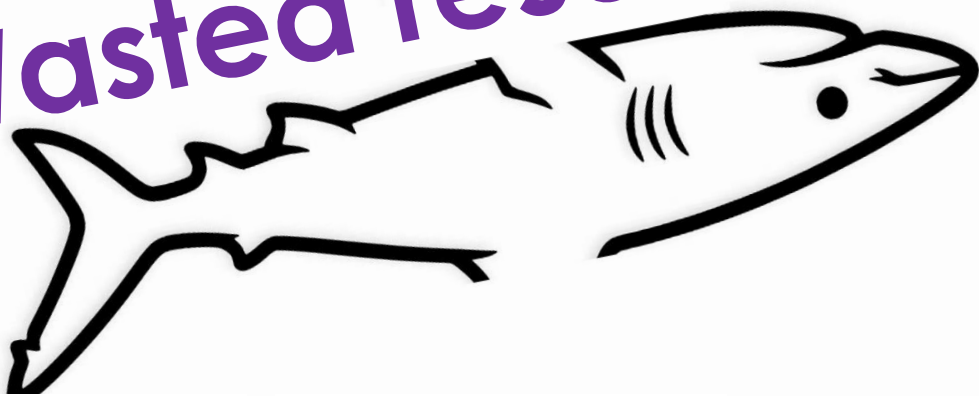
WHY IS THIS STILL SLOW ?



```
select * from  
  where boardin  
    and boarding_time < '2024-06-02'::timestampz  
  order by boarding_time, pass_id offset 346600 limit 100;
```

WHY IS THIS STILL SLOW ?

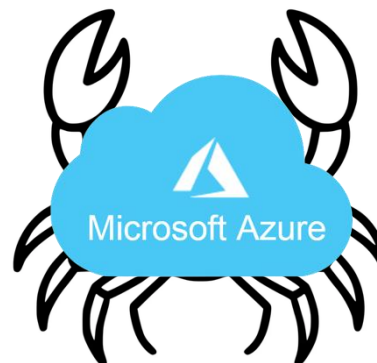
Wasted resources



Page



offset



GET PREVIOUS CHUNK

```
select * from postgres_air.boarding_pass
where boarding_time >= '2024-06-01'::timestamptz
      and boarding_time < '2024-06-02'::timestamptz
order by boarding_time, pass_id offset 346500 limit 100;
-- Last row of data from this chunk:
```

boarding_time	pass_id
2024-06-01 20:30:00.000000 +00:00	28810614

GET CHUNK RIGHT AFTER

boarding_time	pass_id
2024-06-01 20:30:00.000000 +00:00	28810614

```
select * from postgres_air.boarding_pass
where boarding_time >= '2024-06-01'::timestampz
and boarding_time < '2024-06-02'::timestampz
and boarding_time >= '2024-06-01 20:30:00.000000 +00:00'::timestampz
and pass_id > 28810614
order by boarding_time, pass_id limit 100;
-- There you go, execution: 15ms
```

FIX LOGIC

```
select * from postgres_air.boarding_pass
where boarding_time >= '2024-06-01'::timestamptz
and boarding_time < '2024-06-02'::timestamptz
and (
    (boarding_time > '2024-06-01 20:30:00.000000 +00:00'::timestamptz)
or
    (boarding_time = '2024-06-01 20:30:00.000000 +00:00'::timestamptz
    and pass_id > 28810614)
)
order by boarding_time, pass_id limit 100;
-- correct but execution: 145ms
```

NOW REALLY FIX LOGIC USING BRAIN

```
select * from postgres_air.boarding_pass
where boarding_time >= '2024-06-01'::timestamptz
and boarding_time < '2024-06-02'::timestamptz
and (boarding_time, pass_id)
    > ('2024-06-01 20:30:00.000000 +00:00'::timestamptz, 28810614)
)
order by boarding_time, pass_id limit 100;
-- correct and execution: 5ms (!!)
```



... mostly

Paging

More juicy tidbits on database.one/blog
or talk to me directly



ACK:

Mathias van der Meent for spotting the monotonicity trap



Something about:

Selectivity

Something about: selectivity



meaning selectivity database

Something about: selectivity



meaning selectivity database

◆ AI Overview

Database selectivity is the measure of how effectively a query can filter out rows, determined by the ratio of matching rows to the total number of rows. A highly selective query returns a small fraction of rows, making it faster because the database does less work. Conversely, a low-selectivity query returns a large number of rows, which is less efficient. This concept is crucial for database performance optimization, especially when choosing to index a column. [↗](#)

High vs. low selectivity

- **High selectivity:** A query with high selectivity targets a small number of rows from a large dataset.

OR IS IT ?



Something about: selectivity



meaning selectivity

AI Overview

Database selectivity is the measure of how determined by the ratio of matching rows. A selective query returns a small fraction of rows, which does less work. Conversely, a low-selectivity query returns a large fraction of rows, which is less efficient. This concept is crucial for query optimization, especially when choosing to index a column.

High vs. low selectivity

- **High selectivity:** A query with high selectivity returns a small fraction of rows from a large dataset.



Slideshare



Cardinality and Selectivity

- **Cardinality:** Number of distinct values
- **Selectivity:** Cardinality / total number of rows
- What values are better
- Optimize → Stats Update



MySQL: Indexing for Better Performance

Jehad Keriaki 2014

10

Something about: selectivity

Selectivity

Selectivity of an index - the ratio of the number of distinct values in the indexed column(s) to the number of records. The ideal selectivity is one. Such a selectivity can be reached only by unique indexes on NOT NULL columns (UNIQUE or PRIMARY KEY columns).

Example:

Query A select count (*) from users;

Query B select count(distinct username) from users;

B/A = selectivity

A higher selectivity (closer to one) is more desirable. If selectivity is too low the query optimizer won't use it.

Cardinality and Selectivity

Cardinality: Number of distinct values

Selectivity: Cardinality / total number of rows

Low cardinality values are better

Optimize → Stats Update

Indexing for Better Performance

Jehad Keriaki 2014

Something about: selectivity

Most queries retrieve only a fraction of the rows in a table, due to `WHERE` clauses that restrict the rows to be examined. The planner thus needs to make an estimate of the *selectivity* of `WHERE` clauses, that is, the fraction of rows that match each condition in the `WHERE` clause. The information used for this task is stored in the `pg_statistic` system catalog. Entries in `pg_statistic` are updated by the `ANALYZE` and `VACUUM ANALYZE` commands, and are always approximate even when freshly updated.

Something about: selectivity

The planner thus needs to make an estimate of the *selectivity* of `WHERE` clauses, that is, the fraction of rows that match each condition in the `WHERE` clause.

Something about: selectivity

In human language
high selectivity = very picky

In Postgres language
high selectivity = many results, not very picky

Something about: selectivity

Maybe we should not use selectivity

But use some other term instead.

Perhaps **selected-fraction**
or **predicate-fraction**

Something about: selectivity

More juicy tidbits on database.one/blog
or talk to me directly



ACK:

Hetty Dombrovskaya, for in-depth discussion about selectivity

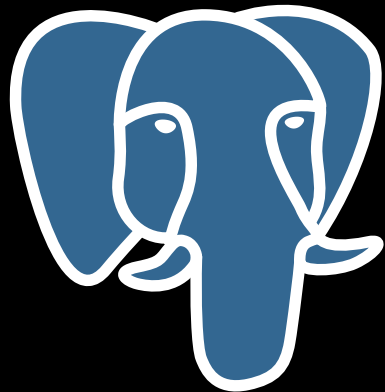
Floor Drees, for pushing me to give this talk the first time



That is all I could cram into this short timeslot today

More juicy tidbits on database.one/blog or talk to me directly or find me on LinkedIn

Leave feedback



That is all I could cram into this short timeslot today

More juicy tidbits on database.one/blog or talk to me directly or find me on LinkedIn

Leave feedback 

