(Ab)using 4D indexing in PostGIS 2.2 with PostgreSQL 9.5 to give you the perfect match

Victor Blomqvist

vb@viblo.se

blomqvist@tantanapp.com

Tantan (探探)

March 19, pgDay Asia 2016 in Singapore

探探

At Tantan we use PostgreSQL & PostGIS for everything!

探探

Suggesting users is difficult

- How should we rank them?
- How can it execute quickly?
  (At Tantan we need to do 1000 ranking queries per second at peak!)

探探

The exciting new feature in PostGIS is this:

<<->>

Today I will show how to take advantage of the 4th dimension!

1. Use double the amount of dimensions from 2
2. …
3. Profit!

探探

We will look at 3 different properties to help our suggestion SELECT

1. Popularity
2. Age
3. Activity

探探

Let’s begin with 2 dimensions

探探

Table and index

```
CREATE TABLE users (
    id serial PRIMARY KEY,
    birthdate date,
    location geometry,
    active_time timestamp,
    popularity double precision
);

CREATE INDEX users_location_gix
    ON users USING GIST (location);
```

探探

Selecting

```
SELECT * FROM users
   ORDER BY location <-> ST_MakePoint(103.8, 1.3)
   /* Singapore */
   LIMIT 10;
```

探探

Lets look at our first case

1. **Popularity** <--
2. Age
3. Activity

探探

The popularity formula

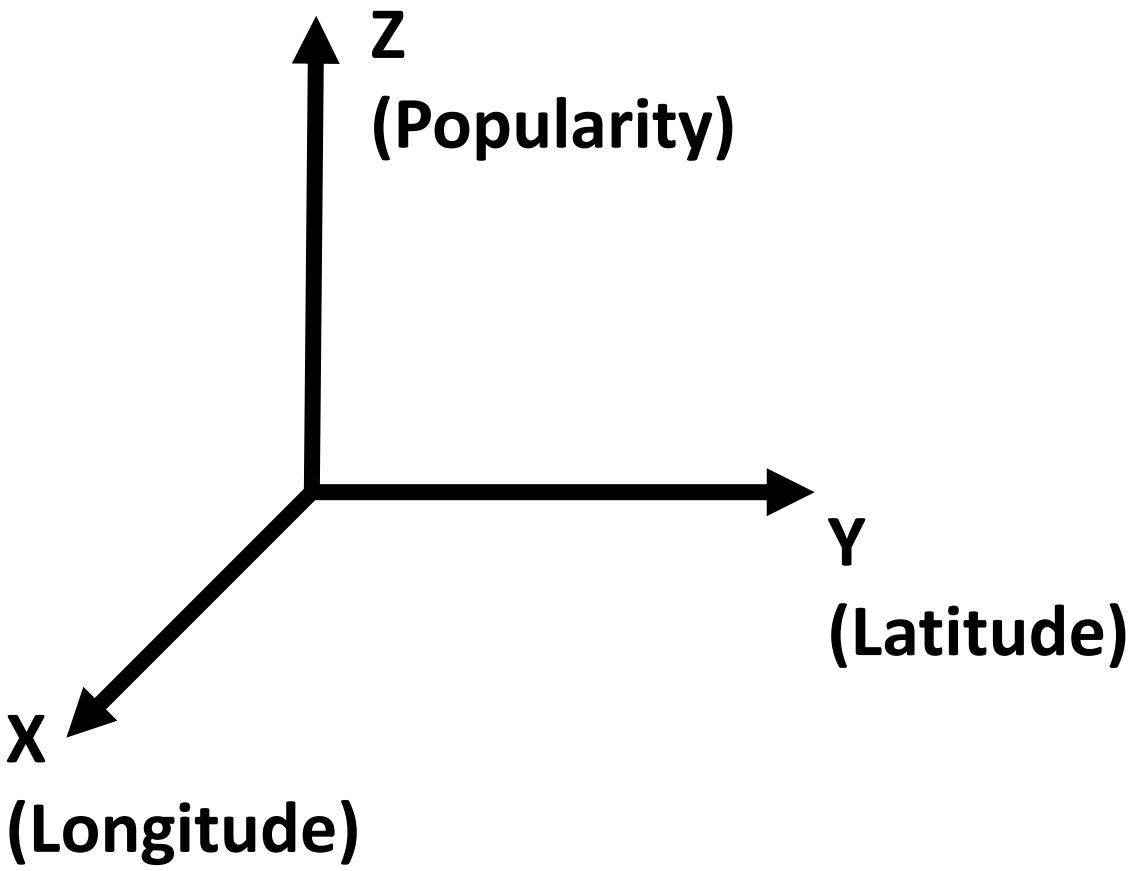
Popularity = likes / (likes + dislikes)

Order by location and popularity

```
WITH x AS (SELECT * FROM users
   ORDER BY location <-> ST_MakePoint(103.8, 1.3)
   LIMIT 100)
SELECT * FROM x
   ORDER BY
      ST_Distance(location::geography, ST_MakePoint(103.8, 1.3))
      * 1 / (popularity+1)
   LIMIT 10;
```

探探

Picture of x-y-z axis



Adding a 3rd dimension!

```
ALTER TABLE users ADD COLUMN loc_pop geometry;

UPDATE users
        SET loc_pop = ST_makepoint(ST_X(location),
    ST_Y(location), 0.01 * 1 / (popularity+1));

CREATE INDEX users_loc_pop_gix
   ON users USING GIST (loc_pop gist_geometry_ops_nd);
```

探探

To use our new 3D index we need to use the new <<->> operator

<<->> — Returns the n-D distance between the centroids of A and B bounding boxes.

This operand will make use of n-D GiST indexes that may be available on the geometries. It is different from other operators that use spatial indexes in that the spatial index is only used when the operator is in the ORDER BY clause.

* http://postgis.net/docs/manual-2.2/geometry_distance_centroid_nd.html

探探

With <<->> our query becomes

```
SELECT * FROM users
    ORDER BY loc_pop <<->> ST_MakePoint(103.8, 1.3, 0)
    LIMIT 10;
```

探探

Second case

1. Popularity
2. **Age** <--
3. Activity

探探

A quick review of our table:

```
CREATE TABLE users (
    id serial PRIMARY KEY,
    birthdate date,
    location geometry,
    active_time timestamp,
    popularity double precision
);
```

探探

Selecting with age filter

```
SELECT * FROM users
   WHERE age(birthdate) between '20 years' AND '30 years'
   ORDER BY location <-> ST_MakePoint(103.8, 1.3)
   LIMIT 10;
```

探探

A very selective user might only want to look at 74 year olds

```
SELECT * FROM users
   WHERE age(birthdate) between '74 years' AND '75 years'
   ORDER BY location <-> ST_MakePoint(103.8, 1.3)
   LIMIT 10;
```

探探

Explain analyze of the query

Limit (cost=0.41..62011.93 rows=10 width=96) (actual time=49.456..551.955 rows=10 loops=1)

-> Index Scan using users_location_gix on users (cost=0.41..1103805.51 rows=178 width=96) (actual time=49.398..546.631 rows=10 loops=1)

Order By: (location <-> 'XXX'::geometry)

Filter: ((birthdate >= (now() - '75 years'::interval)) AND (birthdate <= (now() - '74 years'::interval)))

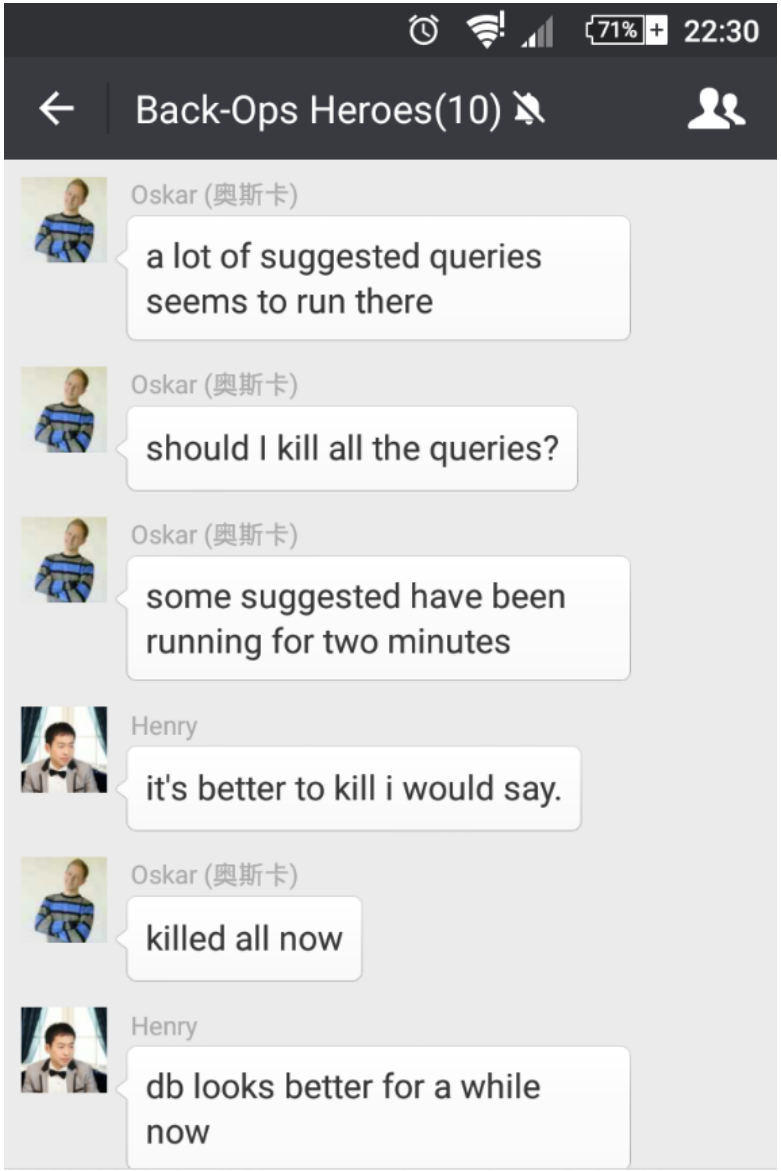
Rows Removed by Filter: 192609

Planning time: 0.157 ms

Execution time: 553.151 ms

探探

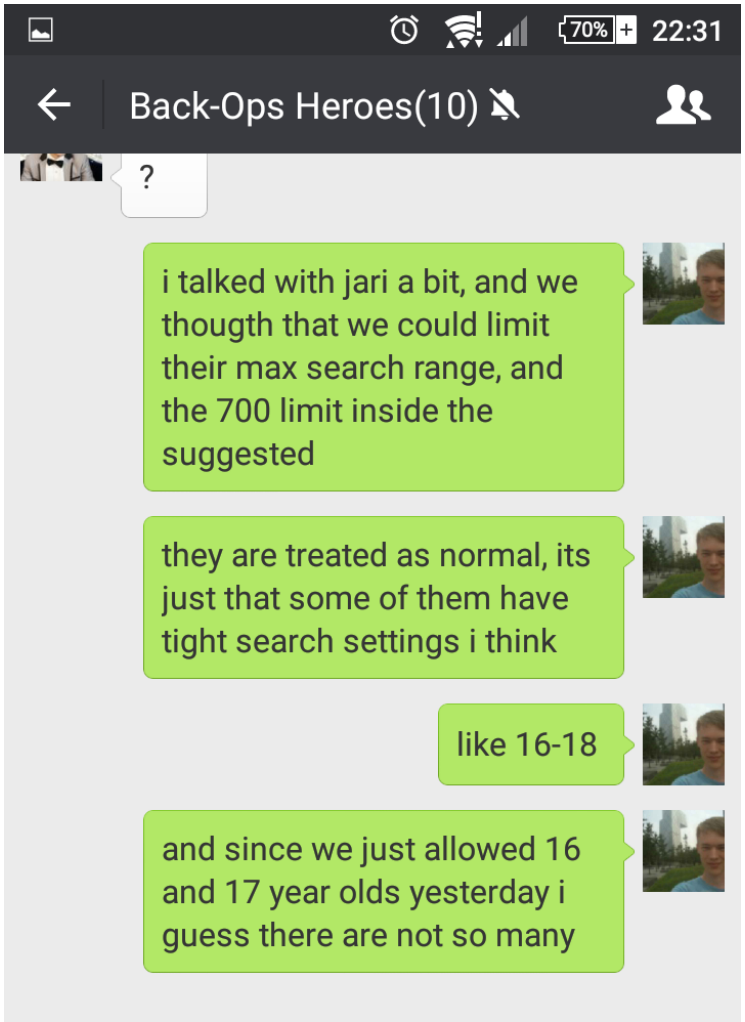
This is a real problem



探探

Possible solutions?

- Prevent searches of restricted ages
- Add a distance restriction
- Add age to the geo index



探探

Adding age to the geo index

```
ALTER TABLE users ADD COLUMN loc_age geometry;

UPDATE users
        SET loc_age = ST_makepoint(ST_X(location), ST_Y(location),
Extract('year' FROM birthdate)/100000);

CREATE INDEX users_loc_age_gix ON users USING GIST (loc_age
gist_geometry_ops_nd);
```

探探

Updated query

```
SELECT * FROM users
   WHERE loc_age &&&
      ST_MakeLine(
         ST_MakePoint(180, 90, Extract('year' FROM Now() - interval '74 years')/100000),
         ST_MakePoint(-180, -90, Extract('year' FROM Now() - interval '75 years')/100000))
   ORDER BY loc_age <<->> ST_MakePoint(103.8, 1.3, 0)
   LIMIT 10;
```

探探

Looking at the execution plan we can see that all is good

Limit (cost=0.41..8.43 rows=1 width=136) (actual time=15.294..16.082 rows=10 loops=1)

  -> Index Scan using users_loc_age_gix on users (cost=0.41..8.43 rows=1 width=136) (actual time=14.685..14.948 rows=10 loops=1)

        Index Cond: (loc_age &&& 'XXX'::geometry)

        Order By: (loc_age <<->> 'XXX'::geometry)

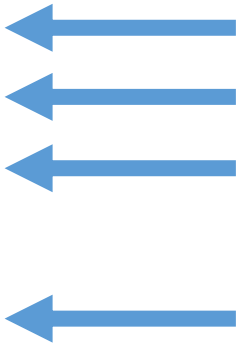
Planning time: 0.332 ms

Execution time: 19.053 ms

探探

Looking at the result we see that something is wrong

UserID	Age
6827677	74 years 11 mons 7 days
1281456	75 years 15 days
1269119	73 years 7 mons 27 days
5791734	73 years 7 mons 8 days
3875002	74 years 7 mons 14 days
6373179	73 years 5 mons 8 days
3727434	74 years 7 mons 21 days
5214330	74 years 10 days
3127049	74 years 8 mons 22 days
6390900	74 years 21 days



Solution: Keep the non-geometry where statement

```
SELECT * FROM users
   WHERE age(birthdate) BETWEEN '74 years' AND '75 years'
   AND loc_age &&&
      ST_MakeLine(
         ST_MakePoint(180, 90, Extract('year' FROM Now() - interval '74 years')/100000),
         ST_MakePoint(-180, -90, Extract('year' FROM Now() - interval '75 years')/100000))
   ORDER BY loc_age <<->> ST_MakePoint(103.8, 1.3, 0)
   LIMIT 10;
```

探探

Finally we look at Activity

1. Popularity
2. Age
3. **Activity** <--

探探

Order by last active time

```
WITH x AS (SELECT * FROM users
   ORDER BY location <-> ST_MakePoint(103.8, 1.3)
   LIMIT 100)
SELECT * FROM x
   ORDER BY
      ST_Distance(location::geography, ST_MakePoint(103.8, 1.3))
      * Extract(Now() - active_time)
   LIMIT 10;
```

探探

What is the difference between time, popularity and age?

- Popularity is bounded, a users popularity can range between 0 and 1.
- Age is also bounded, and static. All our users are between 16 and 100 years old and their birthdate never change.
- Time increase infinitely in one direction.

Adding time to the geo column

```
ALTER TABLE users ADD COLUMN loc_active geometry;

UPDATE users
   SET loc_active = ST_MakePoint(ST_X(location), ST_Y(location),
      Extract('epoch' FROM active_time) / 60 / 10000);

CREATE INDEX users_loc_active_gix
   ON users USING GIST (loc_active gist_geometry_ops_nd);
```

探探

Select with time becomes

```
SELECT * FROM users
   ORDER BY loc_active <<->> ST_MakePoint(103.8, 1.3,
         Extract('epoch' FROM Now()) / 60 / 10000)
   LIMIT 10;
```

探探

What we talked about so far

1. Location X and Y

2. Popularity
3. Age
4. Activity

探探

Adding a 4th dimension

```
ALTER TABLE users ADD COLUMN loc_pop_age geometry;

UPDATE users
        SET loc_pop_age =
                ST_MakePoint(ST_X(location), ST_Y(location),
                0.01 * 1 / (popularity+1),
                Extract('year' FROM birthdate)/100000);

CREATE INDEX users_loc_pop_age_gix
   ON users USING GIST (loc_pop_age gist_geometry_ops_nd);
```

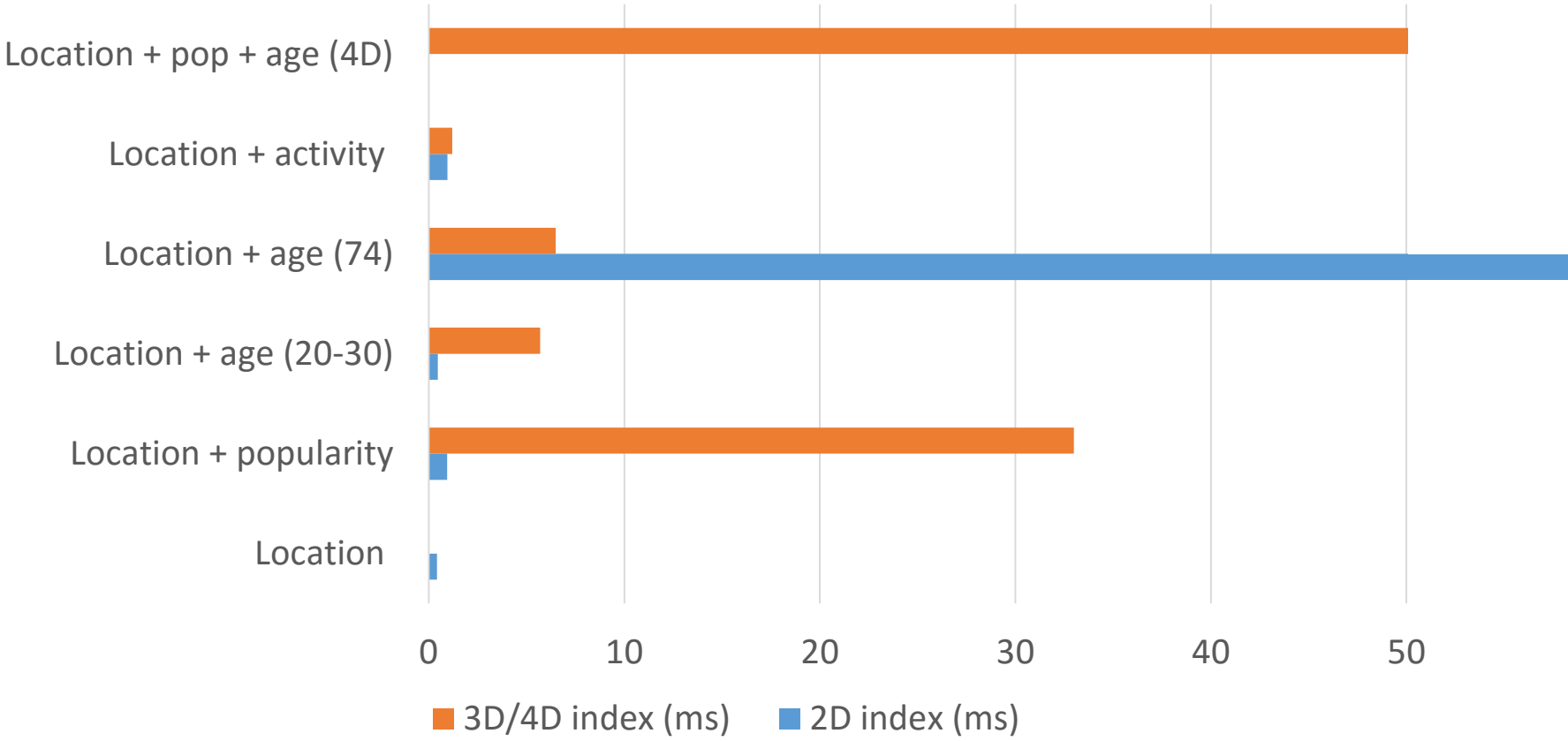
探探

Adding a 4th dimension - Query

```
SELECT * FROM users
   WHERE loc_pop_age &&&
      ST_MakeLine(
         ST_MakePoint(180, 90, -100, Extract('year' FROM Now() - interval
'20 years')/100000),
         ST_MakePoint(-180, -90, 100, Extract('year' FROM Now() - interval
'30 years')/100000))
   ORDER BY loc_pop_age <<->> ST_MakePoint(103.8, 1.3, 0, 0)
   LIMIT 10;
```

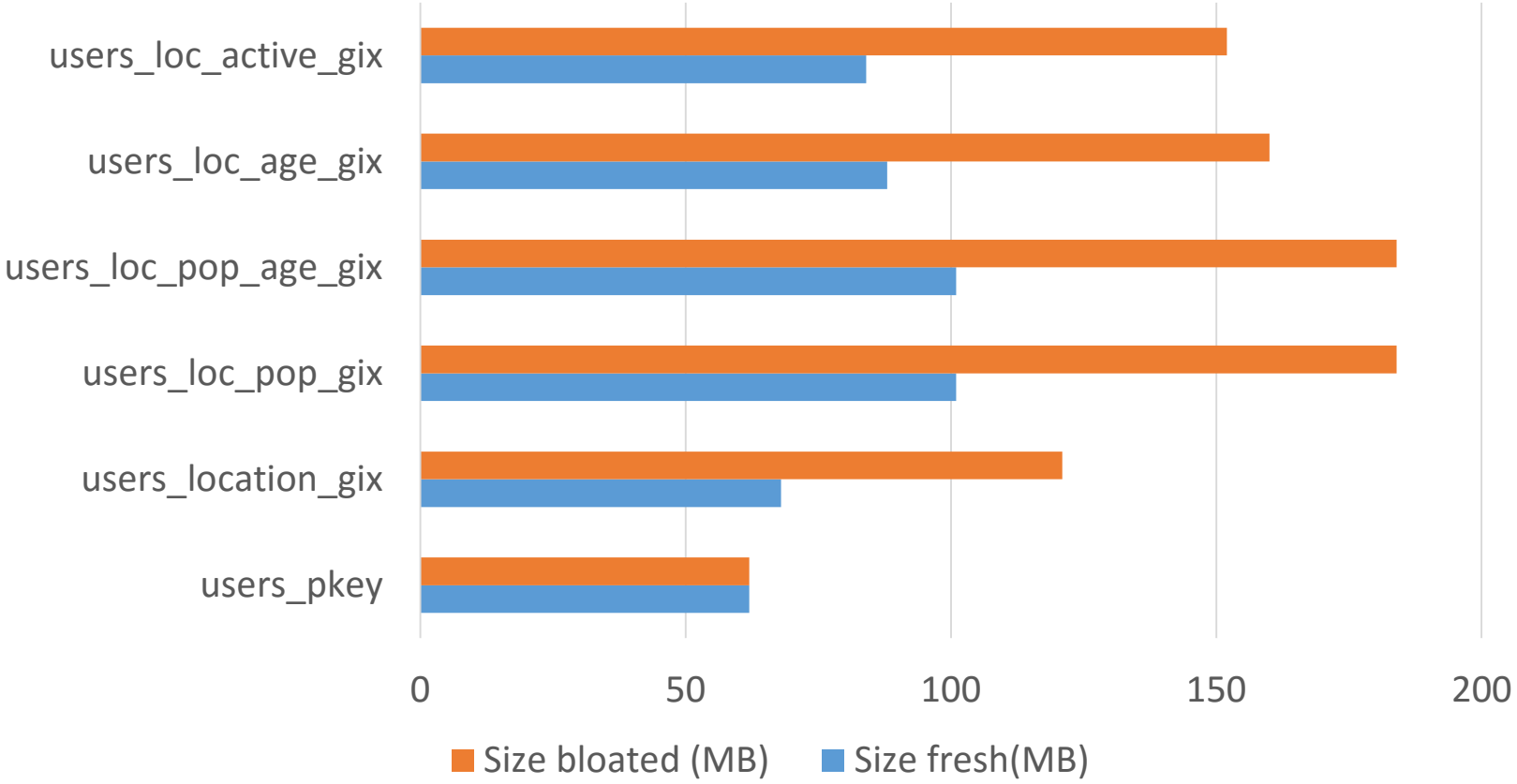
探探

Query runtimes 2D vs 3D vs 4D



探探

Query times did not impress. What about index sizes?



探探

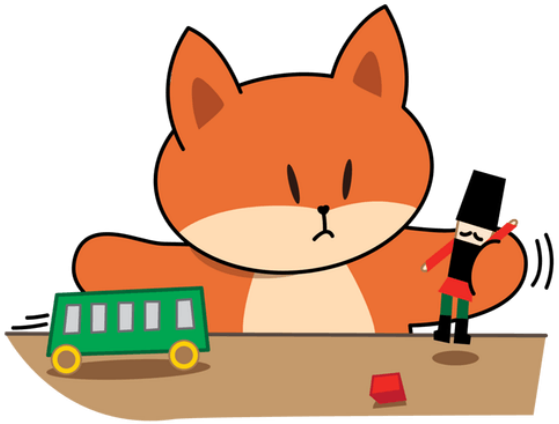
Conclusion

1. Popularity
   - Pro: Improves ranking?
   - Con: Makes it more difficult to reason about the ranking formula
   - Con: Makes the query slower
2. Age
   - Pro: Fixes query time of outlier queries
   - Con: Make the average query time longer
   - Con: Have to take special care to not disturb the normal ranking
3. Time
   - Pro: Improves ranking?
   - Con: Much more difficult to reason about the ranking formula
   - Con: Makes the query slower

探探

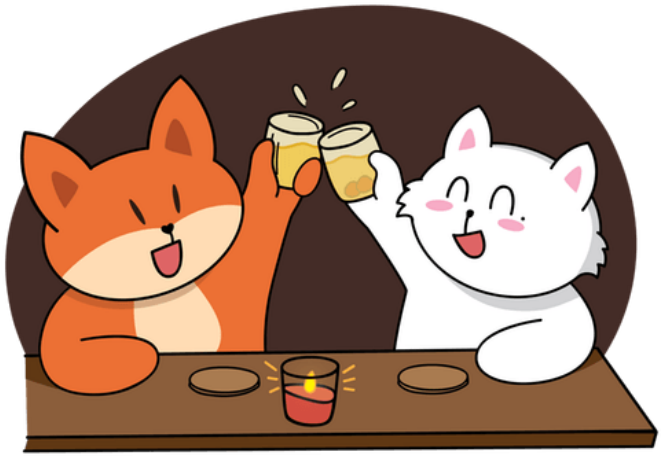
Conclusion

Its more difficult to use the 3rd and 4th dimension than what a quick glance reveals. Test extensively!



探探

Questions?



Thank You!

blomqvist@tantanapp.com
vb@viblo.se

探探