Redpill Linpro

Find your neighbours

Open Source Days 2012
Copenhagen, Denmark

Magnus Hagander
magnus@hagander.net

What's a neighbour

- Closest location by distance
  - (the obvious)
- Closest by similarity
  - Define similarity?
- Closest by anything we can call a distance
  - Define distance?

K-nearest-neighbour

- The theory
  - Collection of n objects
  - Compute distance to point x
  - Find the k closest points to x
- That's KNN

KNN-GiST

- PostgreSQL has GiST indexes
  - Generalized Indexed Search Trees
- In 9.1+, can be used to resolve KNN-style queries fast!
- Uses the LIMIT keyword in SQL to determine query style

Scenarios

- PostgreSQL builtin geometric datatypes
- PostGIS
- Trigrams and other non-location data

Scenarios

- PostgreSQL builtin geometric datatypes
- **PostGIS**
- Trigrams and other non-location data

Typical query scenario

- “Give me the closest 10 <something>”
  - Train stations
  - Bus stops
  - Coffee shops
  - Bars
  - …

PostGIS 2.0

- CREATE EXTENSION postgis
- Currently beta2

Some basic data

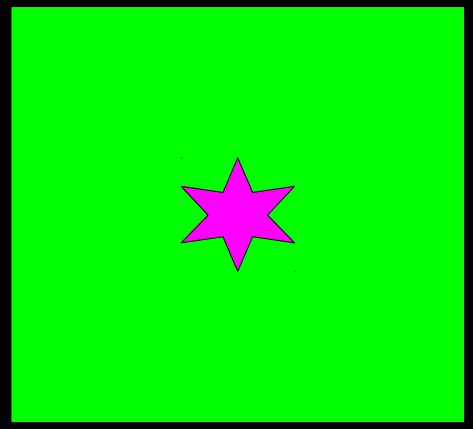
```
CREATE TABLE locations (
    locationid int NOT NULL PRIMARY KEY,
    locationname text NOT NULL,
    geo geometry NOT NULL
);

knn=# select count(*) from locations;
37712
```

Find the closest points

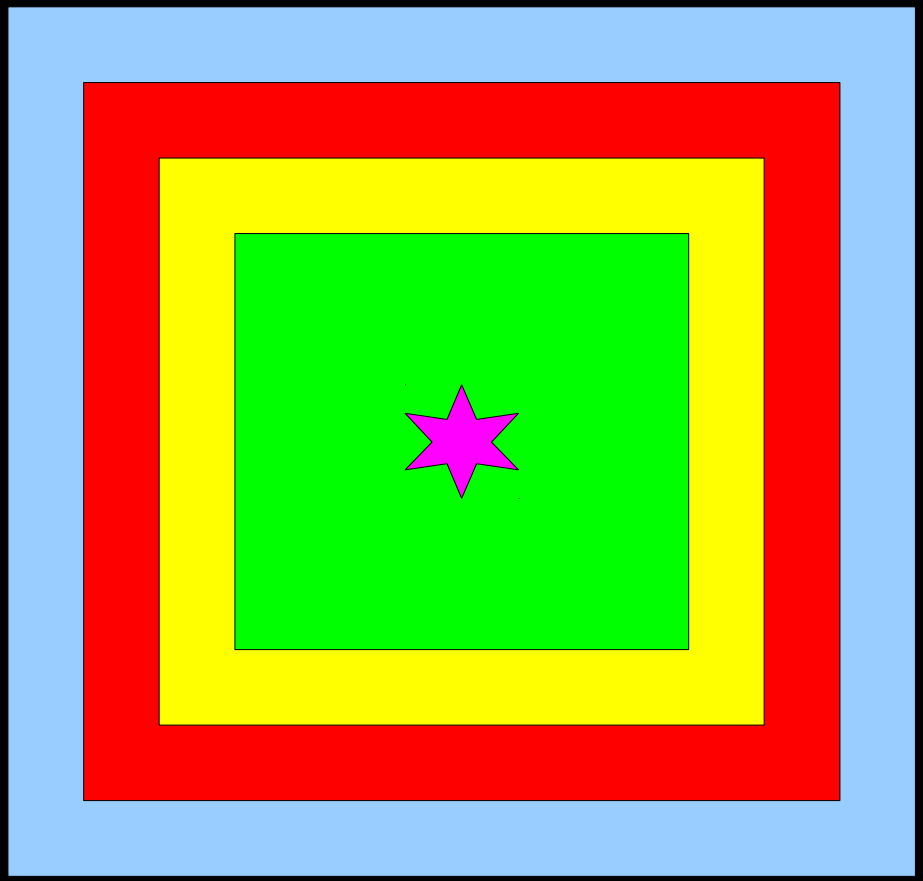
- In a way that scales to large datasets...

Previous method



Bounding box search

Previous method



Previous method

```
CREATE OR REPLACE FUNCTION nearest_locations(
        point geometry,
        pointcount int,
        out locationid integer,
        out locationname text,
        out distance float)
....
```

Previous method

- Function does looping
- Essentially re-running the same query with a larger bounding box
  - PostGIS gives bounding box indexing
- Re-run until we have enough rows
- *Don't forget overflow check!*

Basic PostGIS indexing

```
CREATE INDEX locations_geom_idx
ON locations
USING gist(geo)
```

Previous method

```
EXPLAIN ANALYZE
SELECT
 locationid,locationname,distance
FROM nearest_locations(
  ST_MakePoint(17.97235, 59.35437),
  10)
ORDER BY distance
```

Previous method

Sort (cost=62.58..65.08 rows=1000 width=76) (actual time=9.622..9.622 rows=10 loops=1)

Sort Key: distance

Sort Method: quicksort Memory: 26kB

-> Function Scan on nearest_locations (cost=0.25..12.75 rows=1000 width=76) (actual time=9.507..9.531 rows=10 loops=1)

Total runtime: 9.722 ms

KNN-GiST method

- Requires datatypes that implement “distance” operator
- PostGIS does from 2.0
  - <-> = centroid distance
  - <#> = bounding box distance
  - Equivalent for points

KNN-GiST method

```
EXPLAIN ANALYZE
SELECT locationid, locationname
FROM locations
ORDER BY geo <->
 ST_SetSRID(
  ST_MakePoint(17.97235, 59.35437),
 4326)
LIMIT 10
```

KNN-GiST method

```
 Limit  (cost=0.00..0.85 rows=10 width=148)
(actual time=0.155..0.163 rows=10 loops=1)

   ->  Index Scan using locations_geom_idx on
locations  (cost=0.00..3202.50 rows=37712
width=148) (actual time=0.154..0.161 rows=10
loops=1)

         Order By: (geo <->
'0101000020E6100000FE43FAEDEBF831407E5704FF5BA
D4D40'::geometry)

 Total runtime: 0.198 ms
```

What was that query?

locationid	locationname
21671	Vreten T-bana
773	Sundbyberg station
21668	Näckrosen T-bana
21670	Huvudsta T-bana
21675	Hallonbergen T-bana
21673	Duvbo T-bana
21642	Solna centrum T-bana
20755	Alvik T-bana
21690	Stora mossen T-bana
21689	Abrahamsberg T-bana

Scenarios

- PostgreSQL builtin geometric datatypes
- PostGIS
- **Trigrams and other non-location data**

What's a trigram

- “A trigram is a group of three consecutive characters taken from a string”
- Number of trigrams matching between strings used for similarity matching
  - (in natural languages)

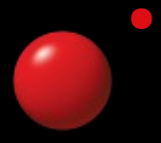
pg_trgm

- CREATE EXTENSION pg_trgm

- In PostgreSQL since 8.3
- Shines in combination with KNN-GiST

Trigram examples

```
knn=# SELECT show_trgm('Vreten');
 {"  v"," vr","en ",ete,ret,ten,vre}
knn=# SELECT show_trgm('Greten');
 {"  g"," gr","en ",ete,gre,ret,ten}
knn=# SELECT similarity('Vreten','Vreten');
        1
knn=# SELECT similarity('Vreten','Greten');
       0.4
```



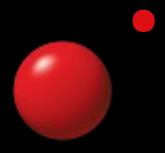
Similarity!

- Also expressed as an operator
  - 1-similarity()
- Rank words by “closest to”

```
SELECT locationname
FROM locations
ORDER BY locationname <-> 'Vreten'
LIMIT 10
```

Similarity results

Vreten T-bana
Vretstorp
Vrena
Vreta kloster
Ven
Anten
Viken
Van Etten Rd
Ålsten
Vreta Strandväg södra



Trigram query plan

```
 Limit  (cost=1677.34..1677.37 rows=10 width=11) (actual time=113.428..113.429 rows=10 loops=1)
   ->  Sort  (cost=1677.34..1771.62 rows=37712 width=11) (actual time=113.425..113.425 rows=10 loops=1)
         Sort Key: ((locationname <-> 'Vreten'::text))
         Sort Method: top-N heapsort  Memory: 25kB
         ->  Seq Scan on locations  (cost=0.00..862.40 rows=37712 width=11) (actual time=0.027..104.345 rows=37712 loops=1)
 Total runtime: 113.456 ms
```

Trigram indexing

```
CREATE INDEX
locationname_trgm
ON locations
USING gist(locationname gist_trgm_ops)
```

Indexed query plan

```
 Limit  (cost=0.00..0.99 rows=10 width=11)
(actual time=12.539..12.608 rows=10 loops=1)
   ->  Index Scan using locationname_trgm on
locations  (cost=0.00..3750.50 rows=37712
width=11) (actual time=12.538..12.604 rows=10
loops=1)
         Order By: (locationname <-> 'Vreten')
 Total runtime: 12.900 ms
```

Trigram index drawbacks

```
locations           | 3128 kB
locationname_idx    | 1128 kB
locations_geom_idx | 1768 kB
locationname_trgm   | 2864 kB
```

Trigram index drawbacks

- Large index size
  - Cache effects
- Expensive to update
- Issues with non-utf8 characters

Bonus!

```
knn=# EXPLAIN SELECT locationname FROM locations
      WHERE locationname LIKE 'Vre%';
                                  QUERY PLAN

-------------------------------------------------
-
 Index Scan using locationname_idx on locations
(cost=0.00..8.27 rows=3 width=11)
   Index Cond: ((locationname ~>=~ 'Vre'::text) A
(locationname ~<~ 'Vrf'::text))
   Filter: (locationname ~~ 'Vre%'::text)
(3 rows)
```

Bonus!

```
knn=# EXPLAIN SELECT locationname FROM locations
      WHERE locationname LIKE '%rete%';
                          QUERY PLAN
-----------------------------------------------------------
 Seq Scan on locations  (cost=0.00..862.40 rows=3 wid
   Filter: (locationname ~~ '%rete%'::text)
(2 rows)
```

Bonus with trigrams!

```
knn=# EXPLAIN SELECT locationname FROM locations
      WHERE locationname like '%rete%';
                                  QUERY PLAN

-------------------------------------------------------------------------
 Bitmap Heap Scan on locations  (cost=4.29..15.54 rows=3 width=11)
   Recheck Cond: (locationname ~~ '%rete%'::text)
   ->  Bitmap Index Scan on locationname_trgm  (cost=0.00..4.29 rows=3 width=0)
         Index Cond: (locationname ~~ '%rete%'::text)
(4 rows)
```

Thank you!

Questions?

Twitter: @magnushagander
http://blog.hagander.net/
magnus@hagander.net

Thanks to Oleg Bartunov, Teodor Sigaev for code and theory,
Jonathan Katz for presentation inspiration