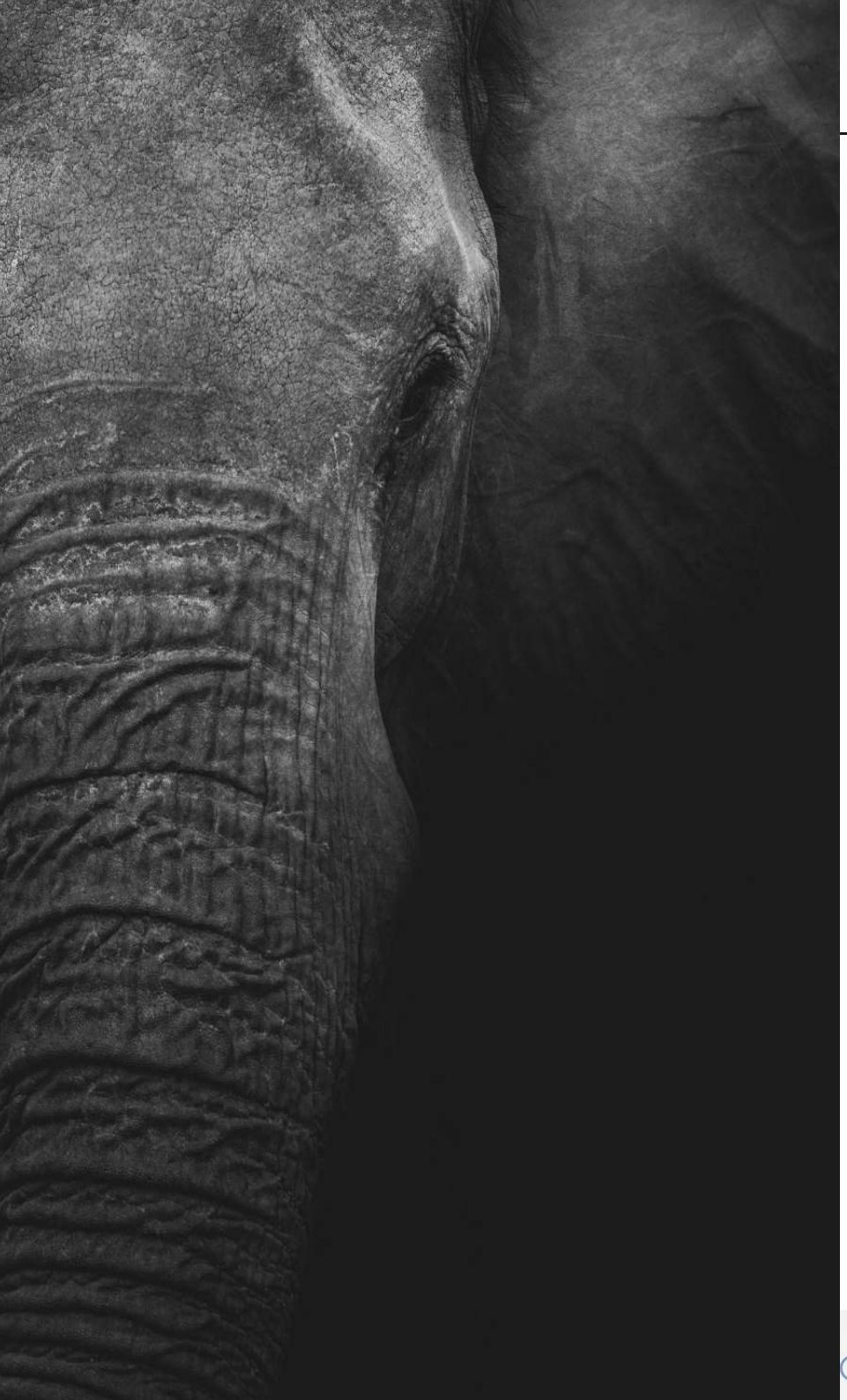
Manipulating text with PostgreSQL
- lesser known PG jewels -

Vincent Picavet – Oslandia
vincent.picavet@oslandia.com

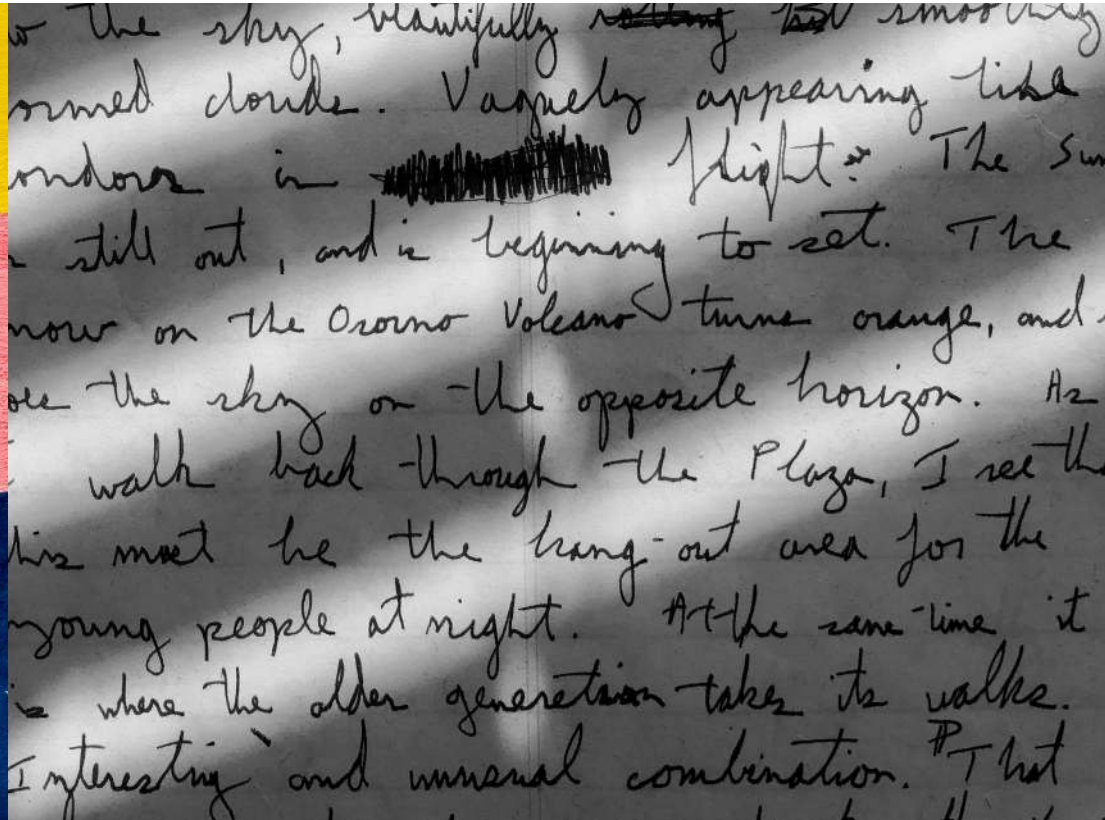


PostgreSQL / PostGIS

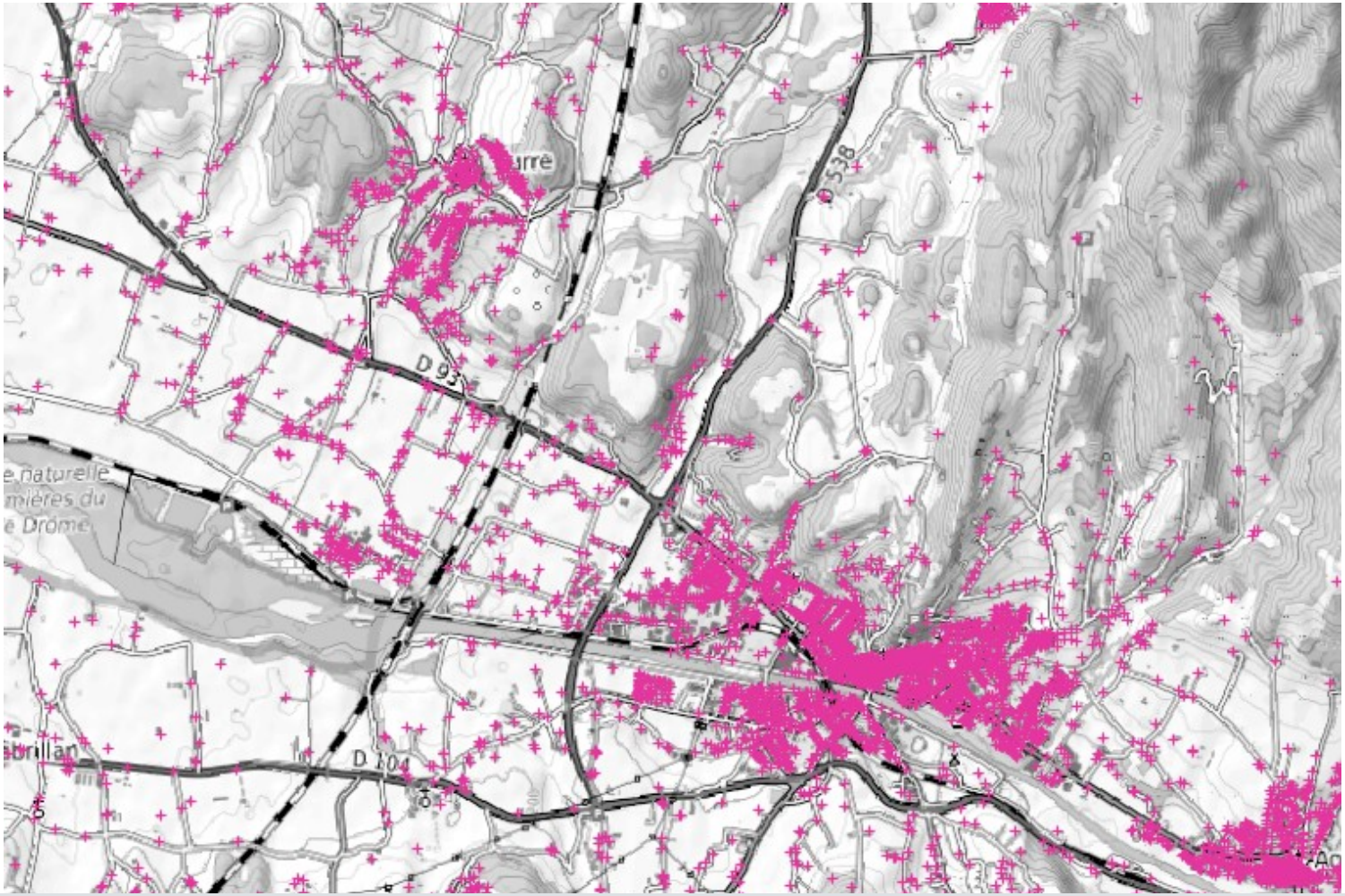
- PostgreSQL
  - The World's Most Advanced OpenSource Relational Database
- PostGIS
  - Spatial extension for PG

Text in GIS ?

- GIS deals with "features" = geometry + attributes
- Most attributes are textual data
- We focus today on some **text search** features in PG



Dataset : addresses in Drôme, FR



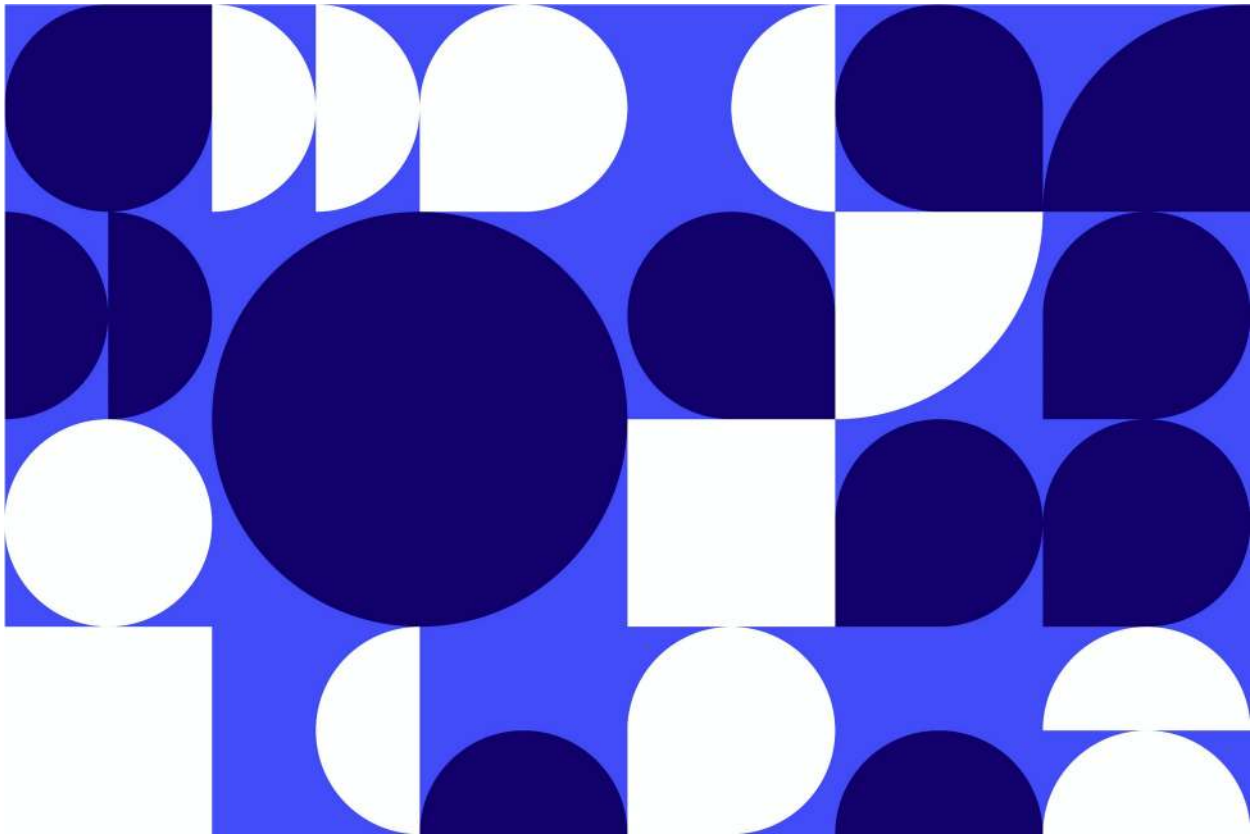
drome — Total des entités: 247699, Filtrées: 247699, Sélectionnées: 0

	ogc_fid	id	id_fantoir	numero	repetition	nom_voie	code_postal	code_insee	om_commun	e_ancienne_c	ncienne_com	alias	nom_ld	e_acheminer	nom_afnor	ource_positio	urce_nom_vc
1	86150	26259_001...	26259_0015	230		Chemin de...	26330	26259	Ratières					RATIERES	CHEMIN D...	commune	commune
2	86151	26259_001...	26259_0016	100		Chemin de...	26330	26259	Ratières					RATIERES	CHEMIN D...	commune	commune
3	86152	26259_001...	26259_0016	110		Chemin de...	26330	26259	Ratières					RATIERES	CHEMIN D...	commune	commune
4	86153	26259_001...	26259_0016	120		Chemin de...	26330	26259	Ratières					RATIERES	CHEMIN D...	commune	commune
5	86154	26259_001...	26259_0018	305		Chemin de...	26330	26259	Ratières					RATIERES	CHEMIN D...	commune	commune
6	86155	26259_001...	26259_0018	415		Chemin de...	26330	26259	Ratières					RATIERES	CHEMIN D...	commune	commune
7	86156	26259_001...	26259_0018	475		Chemin de...	26330	26259	Ratières					RATIERES	CHEMIN D...	commune	commune
8	86157	26259_001...	26259_0018	695		Chemin de...	26330	26259	Ratières					RATIERES	CHEMIN D...	commune	commune
9	86158	26259_001...	26259_0018	840		Chemin de...	26330	26259	Ratières					RATIERES	CHEMIN D...	commune	commune

Lesser known PG jewels

- Standard pattern matching
- Fuzzy matching
- Trigrams
- Full text search
- Collations

Standard pattern matching



Standard pattern matching : LIKE

- LIKE or ~~ : pattern matching
  - 'hello' LIKE '**h_l%**' → true
  - '**_**' → any single character
  - '**%**' → zero or more characters
- ILIKE or ~~* : case insensitive
- NOT LIKE / NOT ILIKE
- /!\ limited indexing

```
select distinct(nom_commune) from drome where nom_commune ilike 'b%';
```

	nom_commune
1	Beaurières
2	Bellecombe-Tarendol
3	Buis-les-Baronnies
4	Barbières
5	Bouvières
6	Beaumont-en-Diois
7	Bésignan
8	Ballons

Standard pattern matching : SIMILAR TO

- SIMILAR TO : regexp pattern matching
  - 'Hello' SIMILAR TO '%(H|h)%' → true
  - SQL standard definition of regular expressions (not POSIX)
  - Match entire string
  - Uses '_' and '%', also | * + ? {m} {m,n} () []

```
select distinct(nom_commune) from drome where nom_commune similar to '(C|B)_a%';
```

	nom_commune
1	Beaurières
2	Charens
3	Beaumont-en-Diois
4	Chastel-Arnaud
5	Chabeuil
6	Charols
7	Chamaloc
8	Chatuzange-le-Goubet

Standard pattern matching : ~ POSIX regexp

- ~ operator : Regular expression matching
  - text ~ text → boolean
  - 'Hello' ~ '(H|h)e.*' → true
  - POSIX regular expressions
  - True if part of text matches
- Other operators
  - ~* : match, case insensitive
  - !~ : does not match
  - !~* : does not match, case insensitive
- Limitations
  - Indexing as for LIKE
  - Using trigram indexing can help

```
select distinct(nom_commune) from drome where nom_commune ~ '(C|B).a.*';
```

Fuzzy matching

Fuzzy matching : Levenshtein

- CREATE EXTENSION fuzzystrmatch;
- Levenshtein distance
  - How many characters to change from one string to another

$$\operatorname{lev}(a,b) = \begin{cases} |a| & \text{if } |b| = 0, \\ |b| & \text{if } |a| = 0, \\ \operatorname{lev}\left(\operatorname{tail}(a), \operatorname{tail}(b)\right) & \text{if } a[0] = b[0] \\ 1 + \min \begin{cases} \operatorname{lev}\left(\operatorname{tail}(a), b\right) \\ \operatorname{lev}\left(a, \operatorname{tail}(b)\right) \\ \operatorname{lev}\left(\operatorname{tail}(a), \operatorname{tail}(b)\right) \end{cases} & \text{otherwise,} \end{cases}$$

- Indexing
  - No indexing possible
  - combine with other means (e.g. trigram)

```
foss4g=> select levenshtein('Firenze', 'Firense');
 levenshtein
-------------
           1
(1 row)
foss4g=> select levenshtein('Firenze', 'Florence');
 levenshtein
-------------
           3
(1 row)
```

Fuzzy matching : soundex

- CREATE EXTENSION fuzzystrmatch;
- **soundex** : sound similarity of english text
  - Convert text to 4 char code
- **difference** : number of common code #
- Specific to english text
- Indexing : use expression index

```
foss4g=> select soundex('Florence'), soundex('Firenze'), soundex('Firense');
 soundex | soundex | soundex
---------+---------+---------
 F465    | F652    | F652
(1 row)
```

```
foss4g=> select difference('Firense', 'Firenze'), difference('Firenze', 'Florence');
 difference | difference
------------+------------
          4 |          1
(1 row)
```

Fuzzy matching : metaphone

- CREATE EXTENSION fuzzystrmatch;
- Metaphone : sound similarity of text
  - Convert text to sound code
- Double metaphone : sound similarity of text
  - Convert text to sound code + alt code
  - Better for non-english texts
  - Alt code can differ according to pronounciation
- Indexing : use expression index

```
foss4g=> select distinct(nom_commune), dmetaphone(nom_commune) from drome where dmetaphone(nom_commune) = dmetaphone('allex');
 nom_commune | dmetaphone
-------------+------------
 Alixan      | ALKS
 Allex       | ALKS
(2 rows)
```

Trigrams

Trigrams : pg_trgm

- CREATE EXTENSION pg_trgm;
- Compare text based on 3-letters decomposition
- Functions
  - similarity(string, string) = <->
  - word_similarity(string, string) = <<->
  - strict_word_similarity(string, string) = <<<->
  - True / false versions : <% % <<%
  - Configurable by GCU

```
foss4g=> select similarity('Welcome to FOSS4G', 'Welcome to Firenze');
 similarity
------------
       0.48
(1 row)

foss4g=> select show_trgm('Welcome to Firenze');
                                                  show_trgm
-------------------------------------------------------------------------------------------------------------
 {"  f","  t","  w"," fi"," to"," we",com,elc,enz,fir,ire,lco,"me ",nze,ome,ren,"to ",wel,"ze "}
(1 row)

foss4g=>
```

Trigrams : pg_trgm

```
create index idx_nom_commune_trgm on drome using gist(nom_commune git_trgm_ops);

select * from drome where nom_commune % 'bordeaux';
```

ogc_fid	id	id_fantoir	numero	repetition	nom_voie	code_postal	code_insee	nom_commune
226 918	26056_e2vfao_0019		190		Crovens	26460	26056	Bourdeaux
226 919	26056_i347lq_0000{		6		la Vialle	26460	26056	Bourdeaux
226 920	26056_i347lq_00211		211		la Vialle	26460	26056	Bourdeaux
226 728	26056_d8u1lk_0010		100		Chemin de la Montagne	26460	26056	Bourdeaux
226 729	26056_d8u1lk_0029		295		Chemin de la Montagne	26460	26056	Bourdeaux
226 730	26056_d8u1lk_0029		297		Chemin de la Montagne	26460	26056	Bourdeaux

Trigrams : indexing

- GIST or GIN, GIST preferred
- … using gist(text_col gist_trgm_ops);
- Trigram index also used for LIKE, ILIKE, Regexp matching !
  - Including non-left anchor

```
explain analyze select * from drome where nom_commune ilike '%ourde%';
                                                    QUERY PLAN

------------------------------------------------------------------------------------------------------------------------------------
--
 Bitmap Heap Scan on drome  (cost=76.71..3177.74 rows=1088 width=196) (actual time=4.852..5.693 rows=1019 loops=1)
   Recheck Cond: ((nom_commune)::text ~~* '%ourde%'::text)
   Heap Blocks: exact=40
   ->  Bitmap Index Scan on idx_nom_commune_trgm  (cost=0.00..76.44 rows=1088 width=0) (actual time=4.838..4.839 rows=1019 loops=1
)
         Index Cond: ((nom_commune)::text ~~* '%ourde%'::text)
 Planning Time: 0.615 ms
 Execution Time: 5.773 ms
(7 rows)
```

Full Text Search (FTS)

FTS : principles

- Search documents for text given in a query
  - **document** = any text
- Core PG feature
- Very flexible and customizable
- All languages
- Preprocess documents for features and efficiency
  - Parse **documents** to **tokens**
  - Convert **tokens** to **lexemes** thanks to **dictionaries**
    - normalize, strip stop words, synonyms…
  - Store specifically for searching, allowing ranking
- New types : **tsvector**, **tsquery**

FTS : generating tsvector

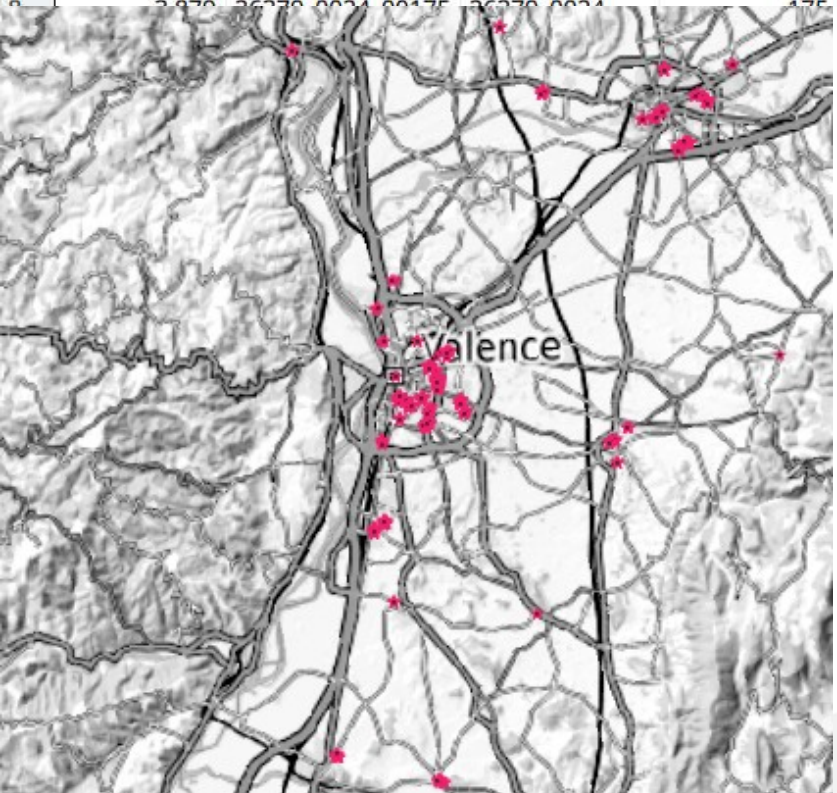
- Documents can be any generated text
- Use generated columns to store **tsvector** data

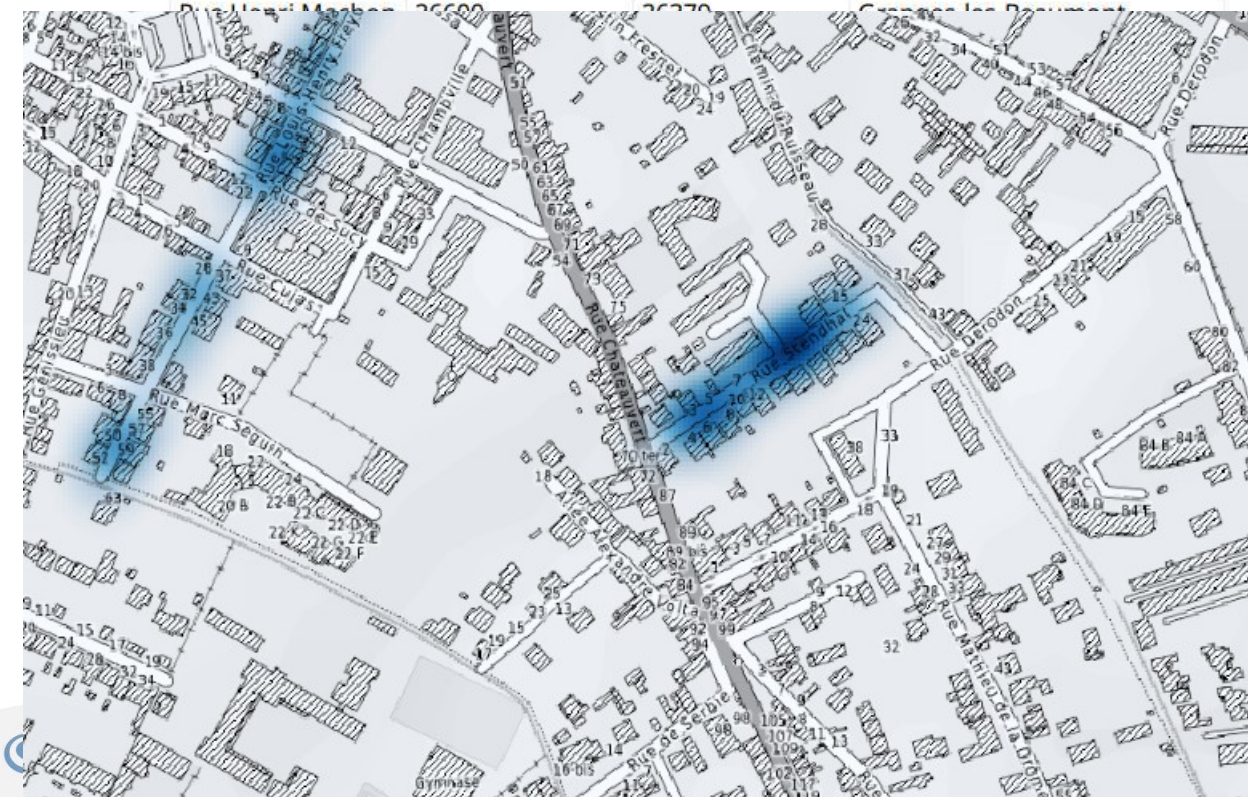
```
ALTER TABLE drome ADD COLUMN vectext tsvector GENERATED ALWAYS AS (
    setweight(to_tsvector('french', coalesce(nom_commune, '')), 'A') ||
    setweight(to_tsvector('french', coalesce(nom_ancienne_commune,'')), 'A') ||
    setweight(to_tsvector('french', coalesce(nom_ld,'')), 'A') ||
    setweight(to_tsvector('french', coalesce(alias, '')), 'B') ||
    setweight(to_tsvector('french', coalesce(nom_voie, '')), 'B') ||
    setweight(to_tsvector('french', coalesce(code_postal,'')), 'C') ||
    setweight(to_tsvector('french', coalesce(code_insee,'')), 'D')
) STORED;
```

FTS : @@ operator

- @@ : text matching
-
```
select * from drome where vectext @@ to_tsquery('french', 'henri');
```

	ogc_fid	id	id_fantoir	numero	repetition	nom_voie	code_postal	code_insee	nom_commune
1	3 579	26382_0019_00001	26382_0019	1		Place Henri Bosc	26300	26382	Saint-Vincent-la-Commanderie
2	3 873	26379_0024_00010	26379_0024	10		Rue Henri Machon	26600	26379	Granges-les-Beaumont
3	3 874	26379_0024_00025	26379_0024	25		Rue Henri Machon	26600	26379	Granges-les-Beaumont
4	3 875	26379_0024_00065	26379_0024	65		Rue Henri Machon	26600	26379	Granges-les-Beaumont
5	3 876	26379_0024_00070	26379_0024	70		Rue Henri Machon	26600	26379	Granges-les-Beaumont
6	3 877	26379_0024_00075	26379_0024	75		Rue Henri Machon	26600	26379	Granges-les-Beaumont
7	3 878	26379_0024_00115	26379_0024	115		Rue Henri Machon	26600	26379	Granges-les-Beaumont

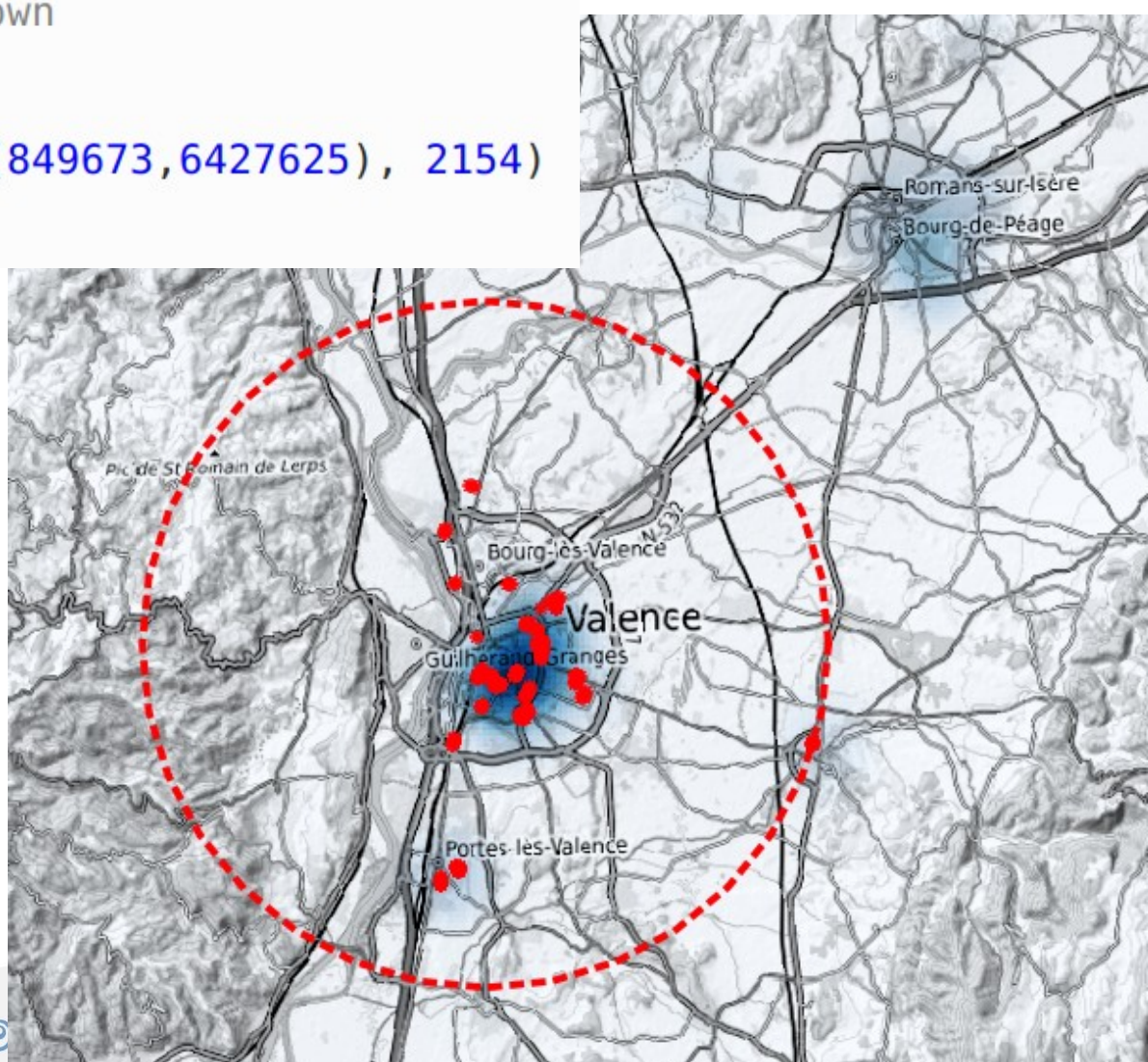




FTS : @@ operator + PostGIS

```sql
select
    *
from
    drome
where
    -- addresses containing "henri"
    vectext @@ to_tsquery('french', 'henri')
    -- within 10km of Valence Downtown
    and st_dwithin(
            wkb_geometry,
            st_setsrid(st_makepoint(849673,6427625), 2154)
            , 10000);
```

- @@ (FTS)
- + && (Geometry)



FTS : functions

- to_tsvector(config, text) → tsvector
- to_tsquery(config, text) → tsquery
- plainto_tsquery(config, text) → tsquery
- phraseto_tsquery(config, text) → tsquery
- websearch_to_tsquery(config, text) → tsquery

- ts_rank
- ts_rank_cd
- setweight

FTS : build queries

- to_tsquery(config, text) → tsquery
- Query operators :
  - | : OR 'rue | chemin'
  - & : AND 'rue & henri'
  - ! : NOT 'rue ! henri'
  - <-> : FOLLOWED BY 'rue <-> henri'
  - ( ) : Grouping conditions '(rue | chemin) & henri'
  - :AB : restrict to specific weight(s) 'henri:B'
  - * : Prefix matching 'henri:*'

FTS : build more queries

- plainto_tsquery(config, text) → tsquery
  - Recombine all words with & (AND) operator
- phraseto_tsquery(config, text) → tsquery
  - Recombine all words with <-> (FOLLOWED BY) operator
- websearch_to_tsquery(config, text) → tsquery
  - Alternate syntax : "google-like"
  - or, - , quoted, unquoted

FTS : indexing

- GIN index

```
create index idx_vectext on drome using gin(vectext);
```

```
foss4g=> explain analyze select * from drome where vectext @@ to_tsquery('french', 'henri');
                                                          QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------
 Bitmap Heap Scan on drome  (cost=21.60..3451.37 rows=1238 width=196) (actual time=0.486..0.960 rows=1168 loops=1)
   Recheck Cond: (vectext @@ '''henr'''::tsquery)
   Heap Blocks: exact=120
   ->  Bitmap Index Scan on idx_vectext  (cost=0.00..21.29 rows=1238 width=0) (actual time=0.459..0.459 rows=1168 loops=1)
         Index Cond: (vectext @@ '''henr'''::tsquery)
 Planning Time: 7.939 ms
 Execution Time: 1.227 ms
(7 rows)
```

FTS : ranking

- ts_rank(tsvector, tsquery) → float4
  - Ranks from lexeme frequency
- ts_rank_cd(tsvector, tsquery) → float4
  - Cover density : frequency + proximity
- Config & normalization options

```
select
    ts_rank_cd(vectext, q) as score, numero, nom_voie, code_postal, code_insee, nom_commune
from
    drome, to_tsquery('french', 'roche') as q
where
    vectext @@ q
order by score desc limit 10;
```

score	numero	nom_voie	code_postal	code_insee	nom_commune
2	131	Rue du village	26400	26277	La Roche-sur-Grane
2	105	Rue du village	26400	26277	La Roche-sur-Grane
2	121	Rue du village	26400	26277	La Roche-sur-Grane
2	141	Rue du village	26400	26277	La Roche-sur-Grane
2	113	Rue du village	26400	26277	La Roche-sur-Grane
1,39999998	3	Le Hameau de la Roche	26600	26271	La Roche-de-Glun
1,39999998	2	Le Hameau de la Roche	26600	26271	La Roche-de-Glun
1,39999998	4	Le Hameau de la Roche	26600	26271	La Roche-de-Glun

FTS : going further

- FTS also provides :
  - Highlighting results
  - Custom dictionaries
  - Unaccent for special character handling
  - pg_trgm use combination
  - Custom parsers (in C)

Collations

Collation : problem

- Text ordering is complex !
- E.g. Alphabetical ordering != natural ordering
  - Is 'a21' before 'a123' or after ?
- E.g. German string ordering
  - Phone book versus dictionary
- E.g. Sorting emojis ?
  - Smile before cry or opposite ?

Beispiel für deutschsprachige Sortierungen [Bearb

DIN 5007 Var.1 (Lexikon)	DIN 5007 Var.2 (Telefonbuch)	Österreichische Sortierung
…	…	…
Göbel	Göbel	Goethe
Goethe	Goethe	Goldmann
Goldmann	Göthe	Göbel
Göthe	Götz	Göthe
Götz	Goldmann	Götz
…	…	…

(kɒleɪʃən , kə-)

NOUN

1. the act or process of collating
2. a description of the technical features of a book
3. *Roman Catholic Church*
   a light meal permitted on fast days
4. any light informal meal
5. the appointment of a member of the clergy to a benefice

Collation : definition

- Collatable data : text, varchar, char
- Collation
  - Sort order
  - Character classification
- Granularity
  - Per column
  - Per operation
- Available collations : libc, ICU
- Examples
  - CREATE COLLATION "und-u-co-emoji-x-icu"
    (provider = icu, locale = 'und-u-co-emoji');
  - SELECT name FROM names
    ORDER BY name COLLATE "de-u-co-standard-x-icu";

Collation : some use cases

- Natural sort
- Diacritic characters
- Specific characters for some languages (e.g. “ß”)
- Case sensitivity
  - Need icu_ext for full support

```
foss4g=> create collation naturalsort (provider = icu, locale = 'en@colNumeric=yes');
CREATE COLLATION
foss4g=> select * from (values('EM001'), ('EM999'), ('EM1000')) as f(emp) order by emp collate naturalsort;
  emp
--------
 EM001
 EM999
 EM1000
(3 rows)

foss4g=> select * from (values('EM001'), ('EM999'), ('EM1000')) as f(emp) order by emp;
  emp
--------
 EM001
 EM1000
 EM999
(3 rows)
```

Collation : emojis

```
foss4g=# CREATE COLLATION "und-u-co-emoji-x-icu" (provider = icu, locale = 'und-u-co-emoji');
CREATE COLLATION
foss4g=# SELECT chr(x) FROM generate_series(x'1F634'::int, x'1F644'::int) AS _(x)
       ORDER BY chr(x)
     COLLATE "und-x-icu";
 chr
-----
 😴
 😵
 😶
 😷
 😸
 😹
 😺
 😻
 😼
 😽
 😾
 😿
 🙀
 🙁
 🙂
 🙃
 🙄
(17 rows)
```

```
     COLLATE "und-u-co-emoji-x-icu";
 chr
-----
 🙂
 🙃
 😶
 🙄
 😴
 😷
 😵
 🙁
 😺
 😸
 😹
 😻
 😽
 😼
 😾
 🙀
 😿
(17 rows)
```

OSLANDIA

Thank you ! Any question

illustrations : Sylvain Beorchia, unDraw.co, unsplash