Text Search on PostgreSQL

Emre Hasegeli

Outline

* LIKE / REGEXP

* Extensions
    * citext
    * unaccent
    * fuzzystrmatch
    * pg_trgm

* Full Text Search

```
# SELECT 'abc' LIKE 'a%';

 ?column?
----------
 t

# SELECT 'abc' SIMILAR TO '%(b|c)%';

 ?column?
----------
 t

# SELECT 'abc' ~ '.*(b|c).*';

 ?column?
----------
 t
```

```
# SELECT 'abc' ILIKE 'A%';

 ?column?
----------
 t

# SELECT 'abc' ~~* 'A%';

 ?column?
----------
 t

# SELECT 'abc' ~* '.*(B|C).*';

 ?column?
----------
 t
```

```
# SELECT lower('ABC') LIKE 'a%';

 ?column?
----------
 t

# CREATE UNIQUE INDEX ON product (lower(name));

# EXPLAIN SELECT * FROM product
          WHERE lower(name) = 'Fisch';

             QUERY PLAN
-------------------------------------------------------
 Index Scan using product_lower_idx on product
   Index Cond: (lower(name) = 'fisch')
```

```
# select lower('ß' collate "C");

 lower
--------
 ß

# select lower('ß' collate "de_DE");  -- On my Mac

 lower
--------
 ß

# select lower('ß' collate "de_DE");  -- Debian

 lower
--------
 ß
```

```
# SELECT upper(lower(x)) = upper(x);
```

```
# SELECT upper('ß' COLLATE "de_DE");

 upper
-------
 ß
```

```
# SELECT lower('I');

 lower
-------
 i

# SELECT lower('I' COLLATE "tr_TR.utf8");

 lower
-------
 ı
```

Indexes

* btree
* SP-GiST
* GIN
* GiST
* BRIN

btree

```
# SELECT x LIKE 'c%';

# SELECT x >= 'c' AND x < 'd';
```

```
# SELECT 'ch' >= 'c' COLLATE "cs_CZ.utf8",
         'ch' <  'd' COLLATE "cs_CZ.utf8";

 ?column? | ?column?
----------+----------
 t        | f
```

```
# CREATE INDEX ON product (name COLLATE "C");

# EXPLAIN SELECT * FROM product WHERE name LIKE 'c%';

                QUERY PLAN
-------------------------------------------------
 Index Scan using product_name_idx1 on product
   Index Cond: ((name >= 'c'::text) AND
                (name <  'd'::text))
```

```
# CREATE INDEX ON product (name text_pattern_ops);

# EXPLAIN SELECT * FROM product WHERE name LIKE 'c%';

                QUERY PLAN
-------------------------------------------------------
 Index Scan using product_name_idx2 on product
   Index Cond: ((name ~>=~ 'c'::text) AND
                (name ~<~  'd'::text))

# EXPLAIN SELECT * FROM product WHERE name = Fisch';

                QUERY PLAN
-------------------------------------------------------
 Index Only Scan using product_name_idx2 on product
   Index Cond: (name = 'Fisch'::text)
```

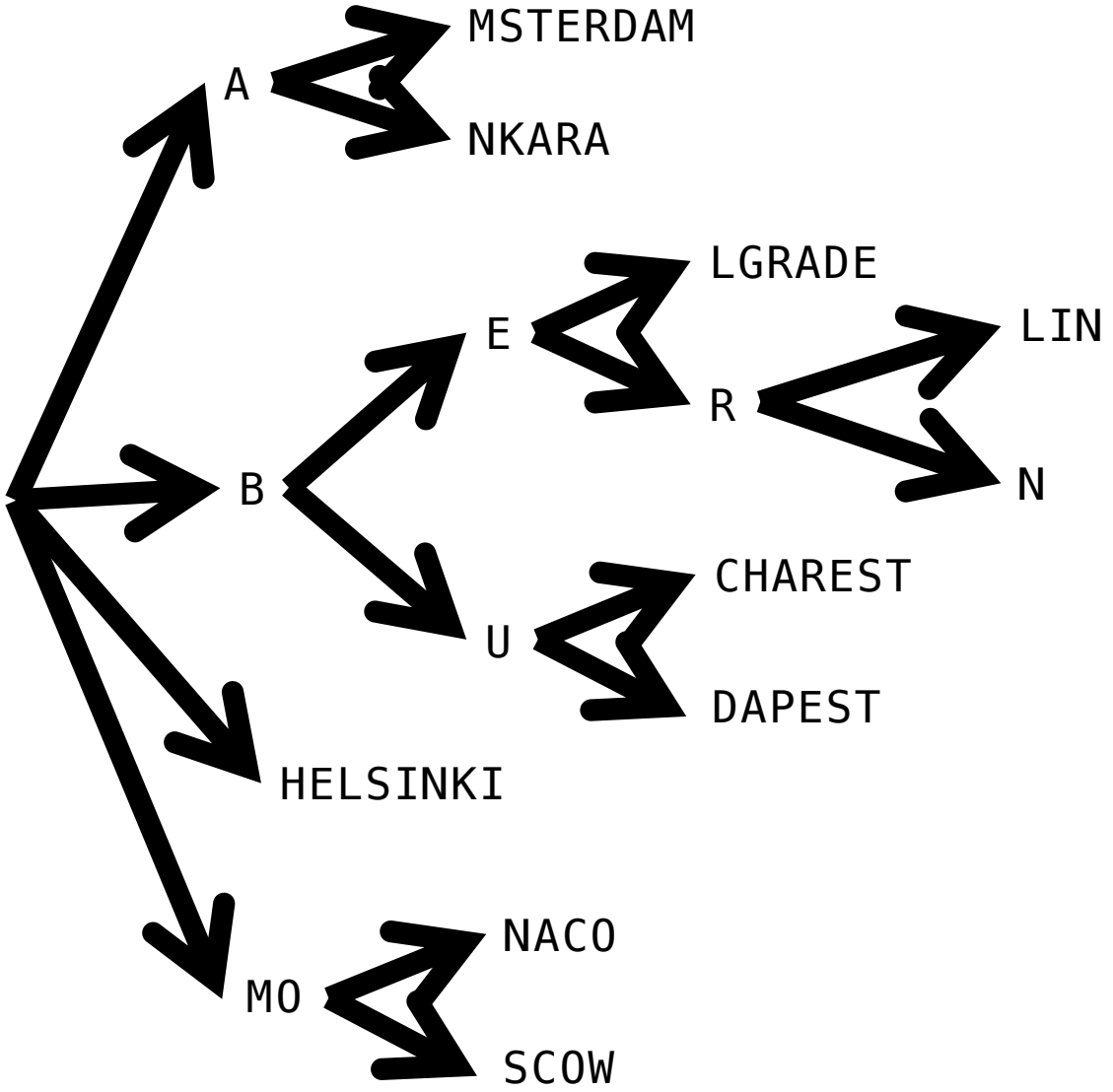
ICU Collations

```
# CREATE COLLATION "de-x-icu"
  (provider = icu, locale = 'de-x-icu');
```

* de-x-icu
* de-AT-x-icu
* und-x-icu
* de-u-co-phonebk-x-icu

```
# SELECT * FROM people ORDER BY name;

  name
---------
 Juergen
 Julia
 Jürgen

# SELECT * FROM people
  ORDER BY name COLLATE "de-u-co-phonebk-x-icu";

  name
---------
 Juergen
 Jürgen
 Julia
```

SP-GiST Prefix Tree



citext

“Essentially, it internally calls lower when comparing values. Otherwise, it behaves ***almost*** exactly like text.”

— Appendix F. Additional Supplied Modules

```
# CREATE EXTENSION citext;

# SELECT 'a'::citext = 'A'::citext;

 ?column?
----------
 t
```

unaccent

```
# SELECT unaccent('Crêpe'),
         unaccent('Äpfel'),
         unaccent('Gemüse');

 unaccent
----------
 Crepe
 Apfel
 Gemuse
```

```
# CREATE INDEX ON product (lower(unaccent(name)));
ERROR:  functions in index expression must be
        marked IMMUTABLE

# CREATE FUNCTION ulower(text) RETURNS text
  IMMUTABLE STRICT LANGUAGE SQL
  AS 'SELECT lower(unaccent($1))';

# CREATE INDEX ON product (ulower(name));
```

fuzzystrmatch

```
* soundex(text) returns text
* difference(text, text) returns int

* levenshtein(text source, text target) returns int
* levenshtein_less_equal(text source, text target,
                         int max_d) returns int

* metaphone(text source, int max_output_length)
  returns text
* dmetaphone(text source) returns text
* dmetaphone_alt(text source) returns text
```

```
# SELECT soundex('Mike'),
         soundex('Meik'),
         soundex('Maik');

 soundex | soundex | soundex
---------+---------+---------
 M200    | M200    | M200


# SELECT difference('Mike', 'Meik'),
         difference('Mike', 'Maik');

 difference | difference
------------+------------
          4 |          4
```

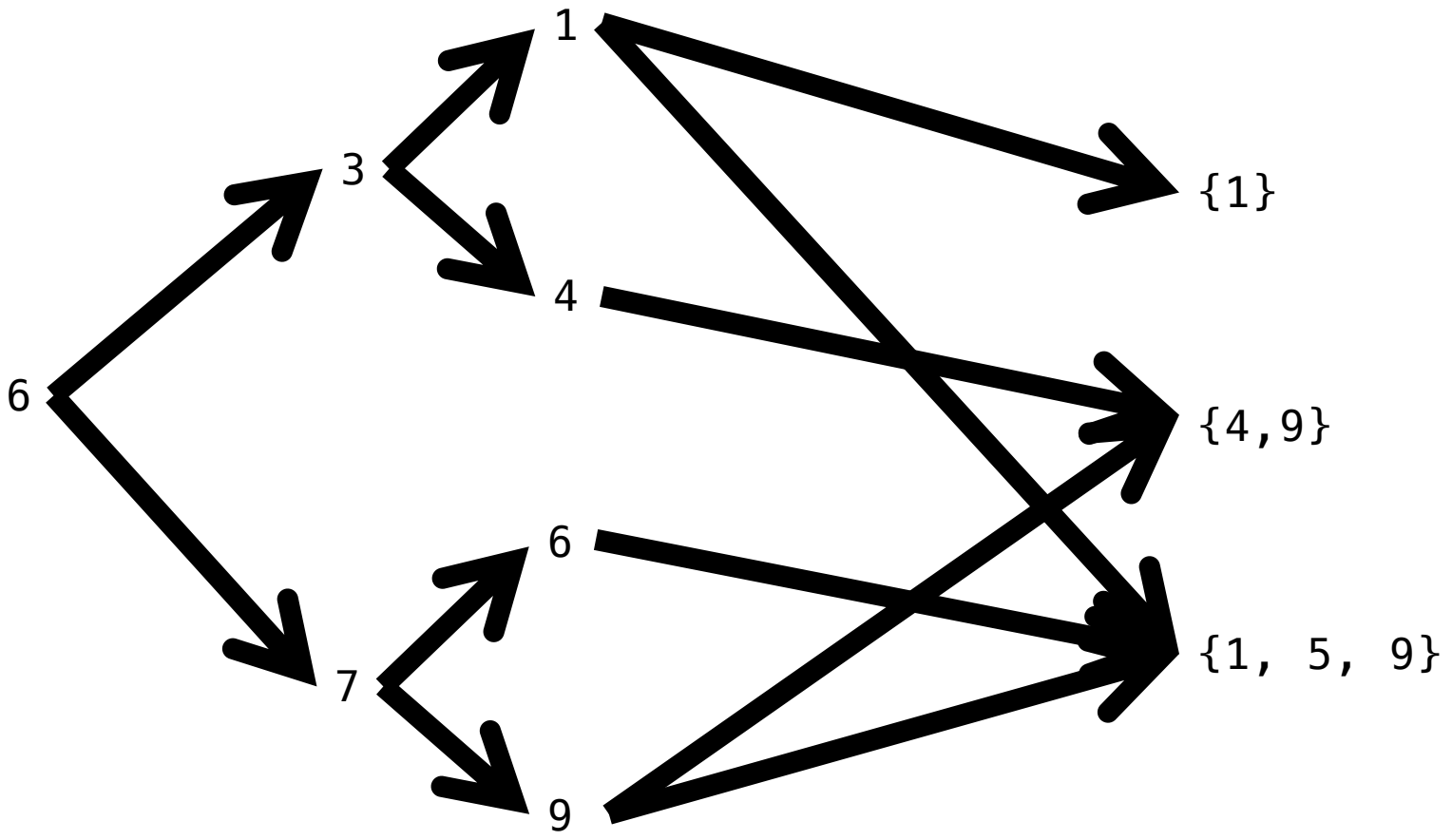
```
# SELECT levenshtein('Mike', 'Meik'),
         levenshtein('Mike', 'Maik');

 levenshtein | levenshtein

-------------+-------------
           2 |           2


# SELECT levenshtein('Christin', 'Christine'),
         levenshtein('Christin', 'Kristin');

 levenshtein | levenshtein
-------------+-------------
           1 |           2
```

```
# SELECT metaphone('Christin', 5),
          metaphone('Christine', 5),
          metaphone('Kristin', 5);

 metaphone | metaphone | metaphone
-----------+-----------+-----------
 KRSTN     | KRSTN     | KRSTN
```

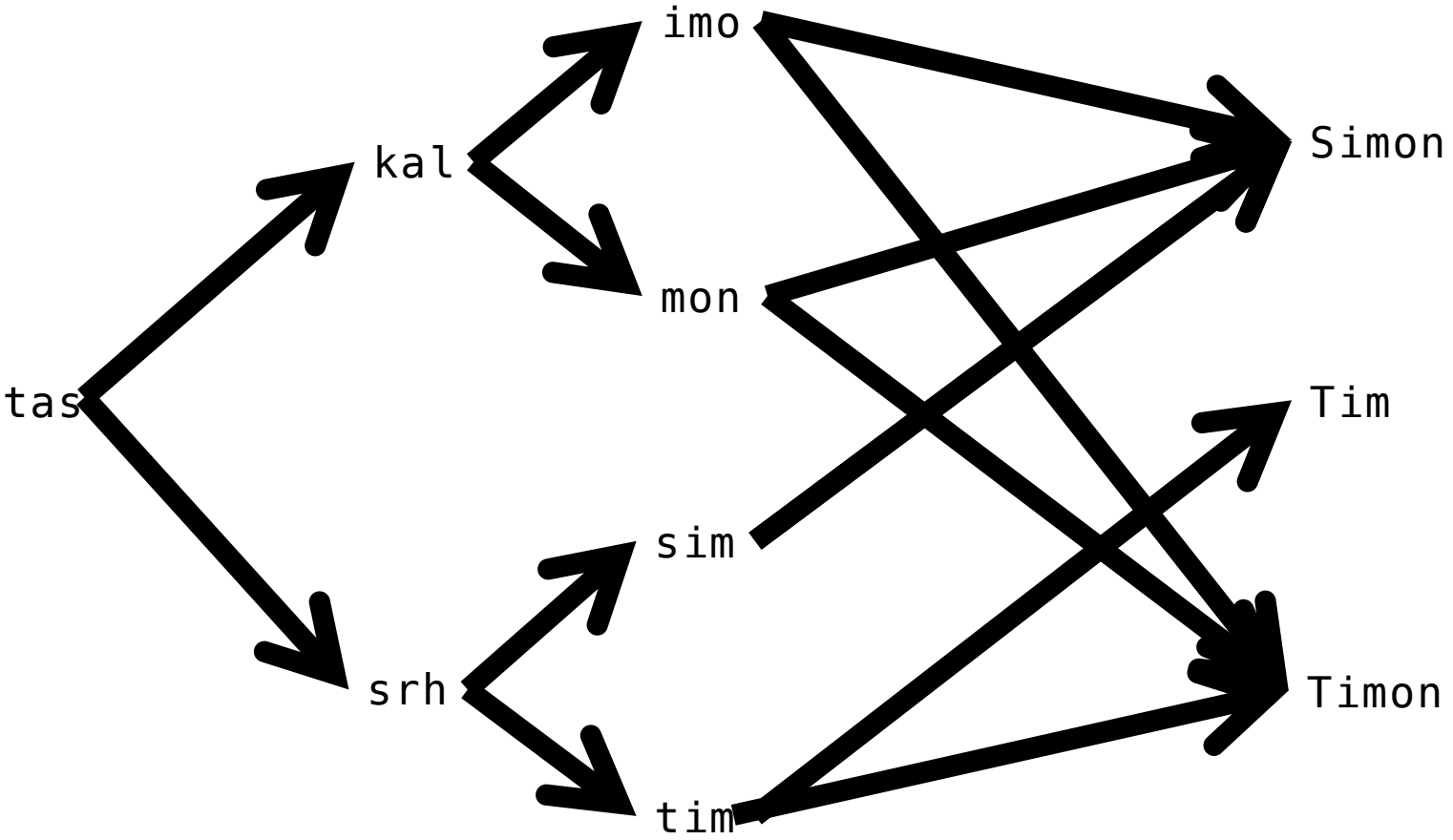
pg_trgm

```
# CREATE EXTENSION pg_trgm;

# CREATE INDEX ON product USING gin (name gin_trgm_ops);

# EXPLAIN SELECT * FROM people
                    WHERE name ~ '.*(a|b)cde.*';

              QUERY PLAN
-----------------------------------------------
 Bitmap Heap Scan on product
   Recheck Cond: (name ~ '.*(a|b)cde.*'::text)
   ->  Bitmap Index Scan on product_name_idx2
         Index Cond: (name ~ '.*(a|b)cde.*'::text)
```

GIN



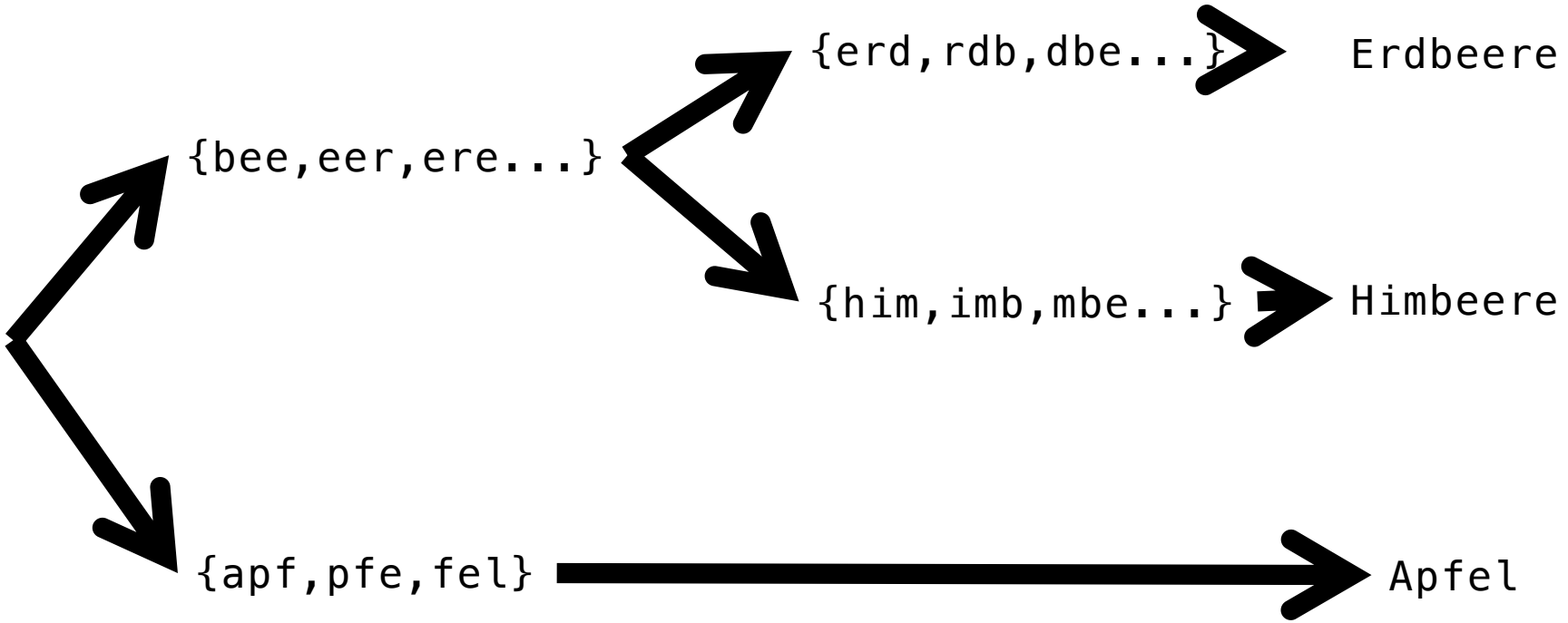
gin_trgm_ops



Similarity

```
# EXPLAIN SELECT *, name <-> 'Erdbere'
  FROM product
  ORDER BY name <-> 'Erdbere ';

   name    | ?column?
-----------+----------
 Erdbeere  |      0.3
 Himbeere  | 0.866667
 Apfel     |        1
```

gist_trgm_ops



Full Text Search

```
# SET default_text_search_config = 'german';


# SELECT name
  FROM product
  WHERE to_tsvector(description) @@ to_tsquery('rot');

 name
----------
 Himbeere
 Erdbeere
 Apfel
```

Full Text Search

```
# SET default_text_search_config = 'german';


# SELECT name, description
  FROM product
  WHERE to_tsvector(description) @@
        to_tsquery('rot & essbar');

   name    |                        description
-----------+-----------------------------------------------------
 Himbeere | ein Strauch mit roten essbaren Beeren
 Apfel    | eine essbare runde rote, grüne oder gelbe Frucht
```

```
# SELECT name, description, to_tsvector(description)
  FROM product;

-[ RECORD 1 ]-------------------------------------------
name        | Himbeere
description | ein Strauch mit roten essbaren Beeren
to_tsvector | 'beer':6 'essbar':5 'rot':4 'strauch':2
-[ RECORD 2 ]-------------------------------------------
name        | Erdbeere
description | eine Pflanze mit weißen Blüten und roten Früchten
to_tsvector | 'blut':5 'frucht':8 'pflanz':2 'rot':7 'weiss':4
-[ RECORD 3 ]-------------------------------------------
name        | Apfel
description | eine essbare runde rote oder grüne Frucht
to_tsvector | 'essbar':2 'frucht':7 'grun':6 'rot':4 'rund':3
```

Snowball Stemmers

* Danish
* Dutch
* English
* Finnish
* French
* German
* Hungarian
* Italian
* Norwegian
* Portuguese
* Romanian
* Russian
* Spanish
* Swedish
* Turkish

Ispell / MySpell / Hunspell

* LibreOffice
* LibreOffice extensions
* Mozilla Add-Ons

Hunspell Example

```
$ cd share/tsearch_data

$ wget https://cgit.freedesktop.org/libreoffice/
dictionaries/tree/de/de_DE_frami.aff

$ wget https://cgit.freedesktop.org/libreoffice/
dictionaries/tree/de/de_DE_frami.dic

$ iconv -f ISO_8859-1 -t UTF-8 de_DE_frami.aff >
  de_DE_frami.affix

$ iconv -f ISO_8859-1 -t UTF-8 de_DE_frami.dic >
  de_DE_frami.dict
```

TEXT SEARCH CONFIGURATION

```
# CREATE TEXT SEARCH DICTIONARY german_hunspell
    (template = ispell,
     dictfile = de_DE_frami,
     afffile = de_DE_frami,
     stopwords = german);


# CREATE TEXT SEARCH CONFIGURATION german_custom
    (copy = german);


# ALTER TEXT SEARCH CONFIGURATION german_custom
  ALTER MAPPING FOR asciiword, asciihword, hword_asciipart,
                    word, hword, hword_part
  WITH german_hunspell, german_stem;
```

```
# SELECT to_tsvector('german', 'Kiwis'),
         to_tsvector('german_custom', 'Kiwis');

 to_tsvector | to_tsvector
-------------+-------------
 'kiwis':1   | 'kiwi':1
```

```
# CREATE INDEX name ON table
  USING GIN (to_tsvector('german_custom', description));
```

Comparison

LIKE
----

* Simple
* Case sensitive

pg_trgm
----

* Lightweight
* Well suited for similarity

Regexp
------

* Unsafe
* Standard

Full Text Search
----

* Language dependent
* Complicated preprocessing
* Extensive configuration