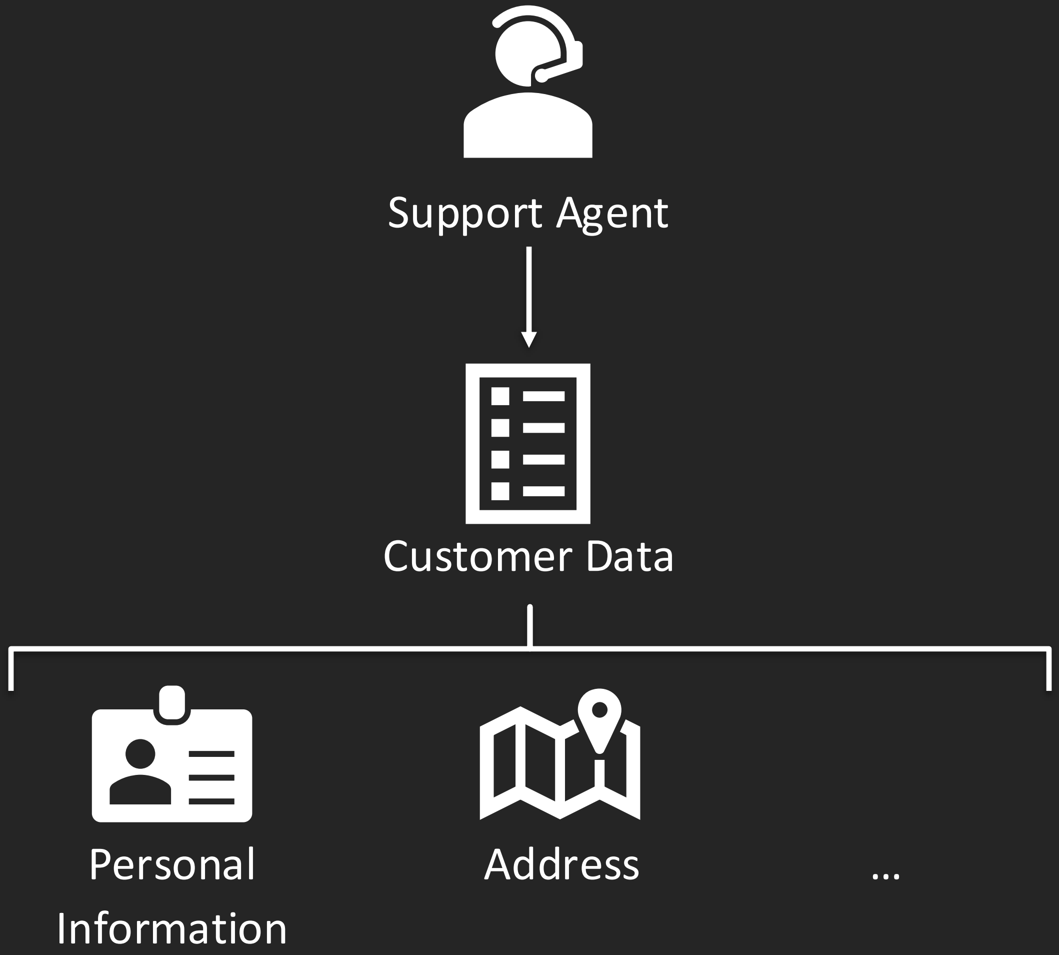
You don't need Elasticsearch!

Fuzzy Search with PostgreSQL and Spring Data

Thomas Gräfenstein - #springio25

About me – Thomas Gräfenstein

- Software Engineer & pragmatic perfectionist @ E.ON One
- Working with JVM (Java, Kotlin) & Spring ecosystem since 2017
- Leading & mentoring software development teams


Support Agent
Customer Data
Personal Information
Address
...

Personal Information

Philipp Meyer
Philip Meier
Phillip Maier
Philipp Mayer

Address

Kölner Straße 17, 51105 Köln
Koelner Strasse 17, 51105 Koeln
Kölner Str. 17, 51105 Köln
kölner strase 17, 51104 köln

Requirements

- Search by a set of attributes
- Search is error-tolerant (typos, misspellings, user-generated content)
- Search is case-insensitive
- Search is fast
- Search provides relevant results
- Search is independent from natural language

Pattern matching with LIKE

Syntax: `string LIKE pattern`

`%` - matches any sequence of 0..n characters

`_` - matches any single character

Example:

```
'Barcelona' LIKE  'Barcelona'        ✓
'Barcelona' LIKE  'Bar%'             ✓
'Barcelona' LIKE  '_elon_'           ✗
'Barcelona' ILIKE '_A_______'        ✓
'Barcelona' ILIKE 'c'                ✗
```

Pattern matching with LIKE

```
'Barcelona' LIKE  'Barcelona' ✓
'Barcelona' LIKE  'Bar%'      ✓
'Barcelona' LIKE  '_elon_'    ✗
'Barcelona' ILIKE '_A_______' ✓
'Barcelona' ILIKE 'c'         ✗
```

🟢 Search by a set of attributes

→ Can easily applied to multiple columns

🔴 Search is error-tolerant (typos, misspellings, user-generated content)

→ human-errors are not predictable

🟢 Search is case-insensitive

→ when using the ILIKE operator

🟢 Search is fast

→ with help of GIST or GIN indexes

🟢 Search provides relevant results

→ for predictable patterns

🔴 Search is independent from natural language

Pattern matching with LIKE

Use LIKE for simple & predictable Patterns

Contract Numbers	`CN-____-__`
Phone Numbers	`+34%`
Email Accounts	`%@gmail.com`
File Paths	`uploads/2025/%`
…	

Trigrams

"A trigram is a group of three consecutive characters taken from a string. We can measure the similarity of two strings by counting the number of trigrams they share. This simple idea turns out to be very effective for measuring the similarity of words in many natural languages."

- *www.postgresql.org/docs/17/pgtrgm.html*

```
SELECT show_trgm('Hello World');
{  h,  w, he, wo,ell,hel,ld ,llo,lo ,orl,rld,wor}
```

Similarity Operator

Measures the similarity between two strings based on trigram matching.

```
SELECT similarity('Hello World', 'Hello World');
Output: 1.0
SELECT similarity('Hello', 'Hello World');
Output: 0.5
SELECT similarity('Bye World', 'Hello World');
Output: 0.375
SELECT similarity('orld', 'Hello World');
Output: 0.21428572
```

“Are these two words kind of alike overall?”

Word Similarity Operator

Measures the similarity between two strings at the word level, considering individual word matches.

```
SELECT word_similarity('Hello World', 'Hello World');
Output: 1.0
SELECT word_similarity('Hello', 'Hello World');
Output: 1.0
SELECT word_similarity('Bye World', 'Hello World');
Output: 0.6
SELECT word_similarity('orld', 'Hello World');
Output: 0.6
```

“Can I slide my little word around inside the other one and find enough matching bits?“

Strict Word Similarity Operator

Same rules of Word Similarity Operator apply + Word Boundaries
=> Greatest similarity between first string and any continuous extent of words

```
SELECT strict_word_similarity('Hello World', 'Hello World');
Output: 1.0
SELECT strict_word_similarity('Hello', 'Hello World');
Output: 1.0
SELECT strict_word_similarity('Bye World', 'Hello World');
Output: 0.6
SELECT strict_word_similarity('orld', 'Hello World');
Output: 0.375
```

“Can I match my little word to one of the full words over there, even if it’s a bit fuzzy?“

Applying Trigrams in Search

`%` – True, if **similarity** > 0.3
`<%` – True, if **word similarity** > 0.6
`<<%` – True, if **strict word similarity** > 0.5

```
SELECT *
FROM customer
WHERE 'Barcelona' <% city
```

equivalent to:

```
SELECT *
FROM customer
WHERE word_similarity('Barcelona', city) > 0.6
```

Similarity Thresholds

System-wide

Configure in `postgresql.conf` and run `pg_ctl reload`

or

```
ALTER SYSTEM SET pg_trgm.word_similarity_threshold = 0.2;
SELECT pg_reload_conf();
```

Per DB

```
ALTER DATABASE mydb SET pg_trgm.word_similarity_threshold = 0.2;
```

=> New setting will be effective for all new connections

Index Support

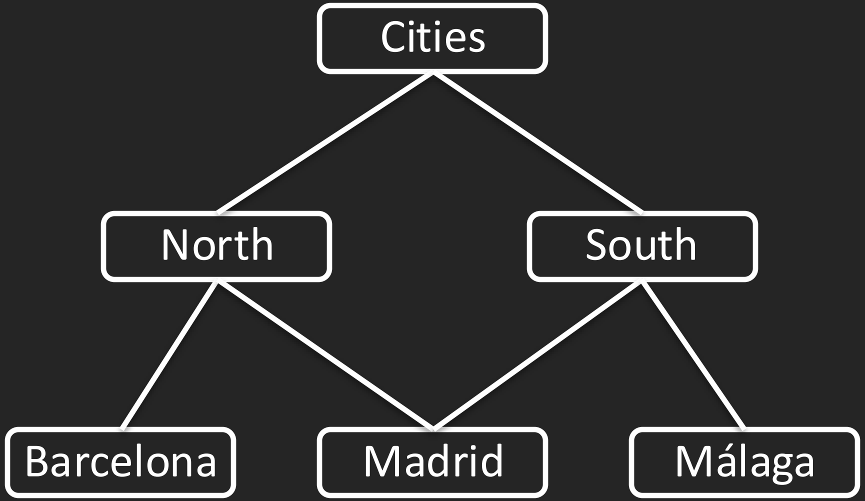
GIST: Generalized Search Tree-based index

```
    CREATE INDEX name ON table USING gist(column)
```

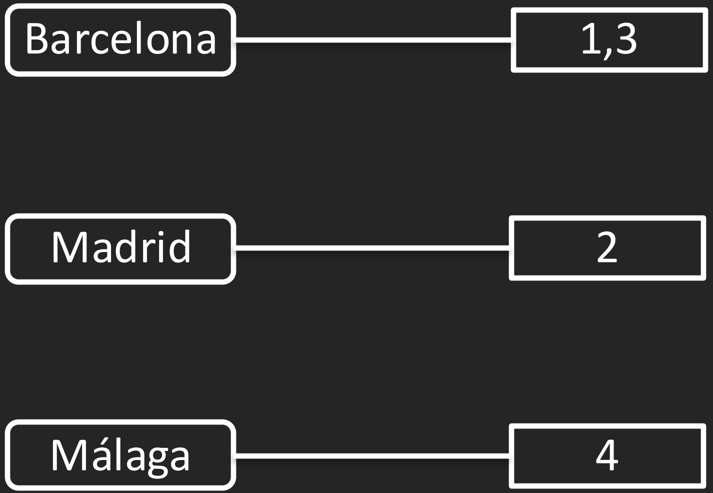
GIN: Generalized Inverted Index-based index

```
    CREATE INDEX name ON table USING gin(column)
```

GIST Index



GIN Index



Which index to choose?

- GIN index lookups are about three times faster than GiST
- GIN indexes are two-to-three times larger than GiST indexes
- GIN indexes take about three times longer to build than GiST
- GIN indexes are moderately slower to update than GiST indexes

Which index to choose?

GIN indexes are best for static data because lookups are faster

vs.

GIST indexes are best for dynamic data because they are faster to update

Search with Trigrams

- ● Search by a set of attributes
  - → Can easily applied to multiple columns
- ● Search is error-tolerant (typos, misspellings, user-generated content)
  - → good fit for natural language
  - → error-tolerance configurable by similarity threshold
- ● Search is case-insensitive
- ● Search is fast
  - → GIST/GIN index support
- ● Search provides relevant results
  - → especially in in layer 8 scenarios
- ● Search is independent from natural language*

* with support of the unaccent extension

Demo Time 😎

Alternatives

- Searching with regular expressions
- Full-Text search (tsvector, tsquery)
- Using the levenshtein distance
- Elasticsearch / OpenSearch
- …

Thank You!