FULLTEXT SEARCH (IN SQL)

QUALITIES OF GOOD SEARCH

QUALITIES OF GOOD SEARCH

- Understands language
  - accents
  - word forms
  - compound words
  - ...
- Tolerates typos
- Fast
- Shows relevant matches first
- Handles "semantics" - "yamaha ydp144", or "airpods", or "100x40"
- ...

NAIVE SEARCH

NAIVE (BUT FREQUENT) SEARCH SOLUTION: LIKE

```
select * from contracts

where lower(name) like '%oracle%';
```

WHY IT'S PROBLEMATIC

- Slooow
- Only "dumb" exact match
- No sense of relevance
- What about multicolumn matches? And more complex queries? ("oracle socialna poistovna")

```
explain analyze select * from contracts
where lower(name) like '%oracle%';


Seq Scan on contracts  (cost=0.00..22391.99
rows=152 width=269) (actual
time=112.302..544.463 rows=28 loops=1)
   Filter: (lower(name) ~~ '%oracle%'::text)
   Rows Removed by Filter: 475038
 Planning time: 0.062 ms
 Execution time: 544.490 ms
```

TRIGRAM INDICES

```
create extension pg_trgm;

select show_trgm('oracle');

                  show_trgm
_____________________________________________
 {"  o"," or",acl,cle,"le ",ora,rac}


CREATE INDEX idx_name
ON contracts USING GIN (name gin_trgm_ops);
```

```
CREATE INDEX idx_name
ON contracts USING GIN (lower(name) gin_trgm_ops);

explain analyze select * from contracts
where lower(name) like '%oracle%';


Bitmap Heap Scan on contracts  (cost=53.18..617.96 rows=152
width=269) (actual time=0.326..0.448 rows=28 loops=1)
   Recheck Cond: (lower(name) ~~ '%oracle%'::text)
   Heap Blocks: exact=28
   ->  Bitmap Index Scan on idx_name  (cost=0.00..53.14
rows=152 width=0) (actual time=0.302..0.302 rows=28 loops=1)
         Index Cond: (lower(name) ~~ '%oracle%'::text)
 Planning time: 0.189 ms
 Execution time: 0.481 ms
```

- Our search is now fast(er), but still not good enough

FULLTEXT SEARCH

STAGES OF SEARCH

- Find matches
- Sort results (ranking)
  - Humans are only interested in top N matches, where N is usually small

STANDARD "PROCEDURE" FOR FINDING MATCHES

- Analyze text
  - Break it into tokens (tokenization)
  - Preprocess tokens into "canonical" forms = lexemes
- Build inverted index

- Query time:
  - Analyze input query
  - Compare the query tokens with inverted index to find a match

EXAMPLES

Background corpus:

"Nylonové gitarové struny"

Tokens:

Nylonové -> nylonové -> nylon

gitarové -> gitarové -> gitara

struny -> struny -> struna

EXAMPLES

Query: struny na gitaru

Tokens:

struny -> struna

na -> <null>

gitary -> gitara

EXAMPLES

- "Yamaha YDP-141"
  - Tokens: yamaha, ydp, 141
- Query: "yamaha ydp141"
  - Tokens: yamaha, ydp, 141

EXAMPLES

- "Dlažba 100x40"
  - Tokens: dlazba, 40x100
- Query: "40x100"
  - Tokens: 40x100

FULLTEXT ANALYSIS IN POSTGRESQL

- Default (*english*) analyzer

```
select to_tsvector(name) from contracts limit 4;
                                to_tsvector
-----------------------------------------------------------------------
 'kúpna':1 'zmluva':2
 'dotáci':4 'o':2 'poskytnutí':3 'zmluva':1
 'dotáci':4 'o':2 'poskytnutí':3 'zmluva':1
 'o':2 'ubytovaní':3 'zmluva':1
```

- Default (*english)* analyzer for query
  - Analyzes input into tokens and builds query
  - Supports syntax for and/or/not..

```
select to_tsquery('oracle');
 to_tsquery
____________
 'oracl'
```

```
explain analyze select * from contracts
where to_tsvector(name) @@ to_tsquery('oracle');

Seq Scan on contracts  (cost=0.00..23579.65
rows=2375 width=269) (actual
time=1161.980..6026.878 rows=28 loops=1)
   Filter: (to_tsvector(name) @@
to_tsquery('oracle'::text))
   Rows Removed by Filter: 475038
 Planning time: 0.099 ms
 Execution time: 6026.911 ms
```

```
select to_tsvector('oracle oracle why are you so
expensive?'), to_tsquery('oracle');

      to_tsvector       | to_tsquery
------------------------+------------
 'expens':7 'oracl':1,2 | 'oracl'
```

```
select ts_debug('oracle oracle why you so expensive?');
                                    ts_debug
-----------------------------------------------------------------------------
 (asciiword,"Word, all ASCII",oracle,{english_stem},english_stem,{oracl})
 (blank,"Space symbols"," ",{},,)
 (asciiword,"Word, all ASCII",oracle,{english_stem},english_stem,{oracl})
 (blank,"Space symbols"," ",{},,)
 (asciiword,"Word, all ASCII",why,{english_stem},english_stem,{})
 (blank,"Space symbols"," ",{},,)
 (asciiword,"Word, all ASCII",you,{english_stem},english_stem,{})
 (blank,"Space symbols"," ",{},,)
 (asciiword,"Word, all ASCII",so,{english_stem},english_stem,{})
 (blank,"Space symbols"," ",{},,)
 (asciiword,"Word, all ASCII",expensive,{english_stem},english_stem,
{expens})
 (blank,"Space symbols",?,{},,)
```

USING SIMPLE ANALYZER (TEXT SEARCH CONFIGURATION)

```
select to_tsvector('simple', 'oracle oracle why
are you so expensive?'), to_tsquery('simple',
'oracle');
```

```
 to_tsvector                  | to_tsquery
------------------------------+------------
'are':4 'expensive':7 | 'oracle'
'oracle':1,2 'so':6   |
'why':3 'you':5.      |
```

TOKENS FROM DOC ANALYSIS AND QUERY ANALYSIS MUST MATCH

```
select * from contracts

where to_tsvector('simple', name) @@
to_tsquery('oracle');

(0 rows)
```

MULTI-COLUMN SEARCH

```
select * from contracts
where
  to_tsvector('simple',
    name || ' ' || department || ' ' ||
  customer || ' ' || supplier)
@@ to_tsquery('simple', 'oracle');
```

```
Seq Scan on contracts  (cost=0.00..29517.98
rows=2375 width=269) (actual
time=691.457..9536.115 rows=57 loops=1)

   Filter: (to_tsvector('simple'::regconfig,
((((((name || ' '::text) || (department)::text)
|| ' '::text) || (customer)::text) || ' '::text)
|| (supplier)::text)) @@ '''oracle'''::tsquery)

   Rows Removed by Filter: 475009

 Planning time: 0.132 ms

 Execution time: 9536.159 ms
```

FAST INVERTED INDEX

```
create index index_contracts_ft on contracts
using gin(to_tsvector('simple', name || ' ' ||
department || ' ' || customer || ' ' ||
supplier))
```

Bitmap Heap Scan on contracts (cost=34.43..6409.35 rows=2375 width=269) (actual time=0.049..0.111 rows=57 loops=1)

  Recheck Cond: (to_tsvector('simple'::regconfig, ((((((name || ' '::text) || (department)::text) || ' '::text) || (customer)::text) || ' '::text) || (supplier)::text)) @@ '''oracle'''::tsquery)

  Heap Blocks: exact=57

  -> **Bitmap Index Scan** on index_contracts_ft (cost=0.00..33.83 rows=2375 width=0) (actual time=0.034..0.034 rows=57 loops=1)

        Index Cond: (to_tsvector('simple'::regconfig, ((((((name || ' '::text) || (department)::text) || ' '::text) || (customer)::text) || ' '::text) || (supplier)::text)) @@ '''oracle'''::tsquery)

Planning time: 0.127 ms

Execution time: **0.152 ms**

QUERY SYNTAX

```
select * from contracts

where to_tsvector(name) @@ to_tsquery('oracle
socialna');


ERROR:  syntax error in tsquery: "oracle
socialna"
```

QUERY SYNTAX

```
to_tsquery('oracle & socialna');
to_tsquery('oracle | socialna');
to_tsquery('oracle & !socialna');
```

CUSTOM TEXT SEARCH CONFIGURATIONS

```
create extension unaccent;

select unaccent('unaccent', 'ľaľa tu beží teľa');

create text search configuration sk(copy = simple);

alter text search configuration sk alter mapping for
word with unaccent, simple;

select to_tsvector('sk', 'ľaľa tu beží teľa')
```

STAGES OF SEARCH

- Find matches
- Sort results (ranking)
  - Humans are only interested in top N matches, where N is usually small

RANKING

- Fulltext score
  - TF.IDF
  - BM25 (more than just score)
- Domain-specific score
  - Pagerank

TF.IDF PRINCIPLES

- TF = term frequency
  - more occurences of the term (token) in document = more relevant document
- IDF = inverse document frequency
  - the more documents in the corpus contain the term = the less important this term (token) is
  - only useful for queries with >2 tokens

VECTOR SPACE MODEL

- query = "best electric cars"
- tokens = [best, electric, cars]
- tf.idf for query = [0.1, 0.2, 0.5]
- tf.idfs for doc 1 = [0.2, 0.8, 0.1]
- tf.idfs for doc 2 = [0.1, 0.3, 0.42]

- Vector similarity (e.g., cosine similarity)

```
select

ts_rank(to_tsvector('sk', name || ' ' ||
department || ' ' || customer || ' ' ||
supplier), to_tsquery('sk', 'oracle')) rank, *
from contracts
where to_tsvector('sk', name || ' ' ||
department || ' ' || customer || ' ' ||
supplier) @@ to_tsquery('sk', 'oracle')

order by rank desc;
```

EVEN MORE FEATURES

- Custom field weights
- Fuzzy matching
- Highlighting matches
- Custom lexemes dictionaries, synonym dictionaries

WHAT YOU SHOULD KNOW

- Understand principles of tokenization and analysis
- Understand the TF.IDF principles and formula