Full text search

Full text search

- Capability to identify natural-language *documents* that satisfy a *query*, and optionally to sort them by relevance to the query.
- Find all documents containing given *query terms* and return them in order of their *similarity* to the query.
- query and similarity are flexible terms and their definition depend on the specific application.

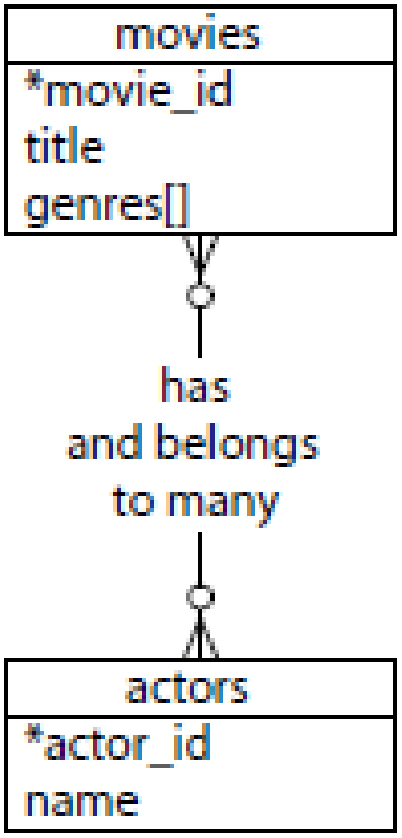
- Search frameworks exist:
  - Lucene (http://lucene.apache.org/)
  - Elasticsearch (https://www.elastic.co/products/elasticsearch)

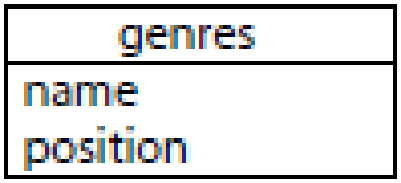
Text search in Postgres

Documents are *preprocessed* to support text search:

- *Parsing documents into tokens*, e.g., numbers, words, complex words, email addresses.
  PostgreSQL uses a *parser* to perform this step. You can write your own.
- *Converting tokens into lexemes*. A lexeme is a *normalized* string/token so that different forms of the same word map to the same lexeme. This step also typically eliminates *stop words*, common words that are useless for searching. PostgreSQL uses *dictionaries* to perform this step. Several available.
- *Storing preprocessed documents optimized for searching*. For example, each document can be represented as a sorted array of lexemes.
- Two indexes to speed up full text searches: GiST (Generalized Search Tree) and GIN (Generalized Inverted Index).
- Contributed packages
  - http://www.postgresql.org/docs/current/static/contrib.html
  - We'll see tablefunc, dict_xsyn, fuzzystmatch, pg_trgm, cube

A movie query system





- Search actor/movie names
- Movies suggestions based on similar genres of movies

- Standard string matches:
  - LIKE and regular expression
  - `SELECT title FROM movies WHERE title ILIKE 'stardust%’;`
  - `SELECT title FROM movies WHERE title ILIKE 'stardust_%';`

Fuzzy search

- Approximate string matching
- Regular expressions (regex)
  - POSIX style
  - All movies that do not start with ‘The’:
  - `SELECT COUNT(*) FROM movies WHERE title !~* '^the.*';`
  - ! -> not matching
  - ~ regular expression match
  - * case insensitive
- *Index columns for pattern matching by using a pattern_ops operator class index*
  - `CREATE INDEX movies_title_pattern ON movies (lower(title) text_pattern_ops);`
  - Also available: `varchar_patterns_ops, bpchar_pattern_ops, and name_pattern_ops`

Levenshtein string comparison

- Edit distance: steps required to change one string into another: count character updates, insertions, and deletions
  - `SELECT levenshtein('bat', 'fads');`
  - Distance = 3

  - Case changes cost a point too, convert all strings to the same case before querying:
  - ```
    SELECT movie_id, title
    FROM movies
    WHERE levenshtein(lower(title),
          lower('a hard day nght')) <= 3;
    ```
    ```

Trigrams

- A trigram is a group of three consecutive characters taken from a string
  - `SELECT show_trgm('Avatar');`
- To query: count the number of matching trigrams, return the most similar
- Great for movie titles because they have similar lengths
- Generalized Index Search Tree (GiST), generic API
  - `CREATE INDEX movies_title_trigram ON movies USING gist (title gist_trgm_ops);`
- Now query:
  - `SELECT title FROM movies WHERE title % 'Avatre';`

Full-text

- *@@ Text-search matching operator:*
  - `SELECT title FROM movies WHERE title @@ 'night & day';`

- Special datatypes to split string into an array of tokens:
  - TSVECTOR to represent text data
  - TSQUERY to represent search predicates
  - Equivalent to above query:
  - `SELECT title FROM movies WHERE to_tsvector(title) @@ to_tsquery('english', 'night & day');`

TSVector and TSQuery

- *Tokens on a tsvector are called lexemes*
- `SELECT to_tsvector('A Hard Day''s Night'), to_tsquery('english', 'night & day');`

- *Number is the position of the lexeme in the text*
- *Stop words are removed*
- `SELECT * FROM movies WHERE title @@ to_tsquery('english', 'a');`

- *Simple dictionary: breaks up strings by nonword characters and makes them lowercase*
- `SELECT to_tsvector('simple', 'A Hard Day''s Night');`
- *Support for many languages*
  - `SELECT to_tsvector('spanish', 'que haces?');`

Indexing lexemes

- *GIN – Generalized Inverted index, like GiST, it's an index API*
- `CREATE INDEX movies_title_searchable ON movies USING gin(to_tsvector('english', title));`

- `EXPLAIN SELECT * FROM movies WHERE title @@ 'night & day';`

- `EXPLAIN SELECT * FROM movies WHERE to_tsvector('english',title) @@ 'night & day';`

Example "Collection"

Four sentences from the Wikipedia entry for *tropical fish*

S_1 Tropical fish include fish found in tropical environments around the world, including both freshwater and salt water species.

S_2 Fishkeepers often use the term tropical fish to refer only those requiring fresh water, with saltwater tropical fish referred to as marine fish.

S_3 Tropical fish are popular aquarium fish, due to their often bright coloration.

S_4 In freshwater fish, this coloration typically derives from iridescence, while salt water fish are generally pigmented.

Inverted Index

- Each index term (posting) is associated with an *inverted list*
  - Contains lists of documents, or lists of word occurrences in documents, and other information
  - Lists are usually *document-ordered* (sorted by document number)

Simple Inverted Index

and	1			
aquarium	3			
are	3	4		
around	1			
as	2			
both	1			
bright	3			
coloration	3	4		
derives	4			
due	3			
environments	1			
fish	1	2	3	4
fishkeepers	2			
found	1			
fresh	2			
freshwater	1	4		
from	4			
generally	4			
in	1	4		
include	1			
including	1			
iridescence	4			
marine	2			
often	2	3		
only	2			
pigmented	4			
popular	3			
refer	2			
referred	2			
requiring	2			
salt	1	4		
saltwater	2			
species	1			
term	2			
the	1	2		
their	3			
this	4			
those	2			
to	2	3		
tropical	1	2	3	
typically	4			
use	2			
water	1	2	4	
while	4			
with	2			
world	1			

Metaphones

- *Algorithms to create string representation of word sounds*
- `Select metaphone('Modern databases',9);`

- `SELECT * FROM actors WHERE name = 'Broos Wils'; /*No matches*/`
- `SELECT * FROM actors WHERE name % 'Broos Wils'; /*Using trigrams*/`

- `SELECT * FROM actors WHERE metaphone(name,6) = metaphone('Broos Wils',6); /*Using metaphone*/`

More fuzzystmatch functions

- *dmetaphone:* double metaphone
- *dmetaphone_alt:* alternative name pronunciations
- *soundex:* old algorithm (1880) to compare American surnames

- `SELECT name, dmetaphone(name), dmetaphone_alt(name), metaphone(name, 8), soundex(name) FROM actors;`

- One of the most flexible aspects of metaphones is that their outputs are just strings

For today…

Write a query that combines AT LEAST TWO of the string match functions we covered today.

For example, use the trigram operator against metaphone() outputs and then order the results by the lowest Levenshtein distance.
E.g. “Get me names that sound the most like Robin Williams”

Submit your sql statement in ICON