Text Search and Similarity Search

PG 12.1–12.2, F.31

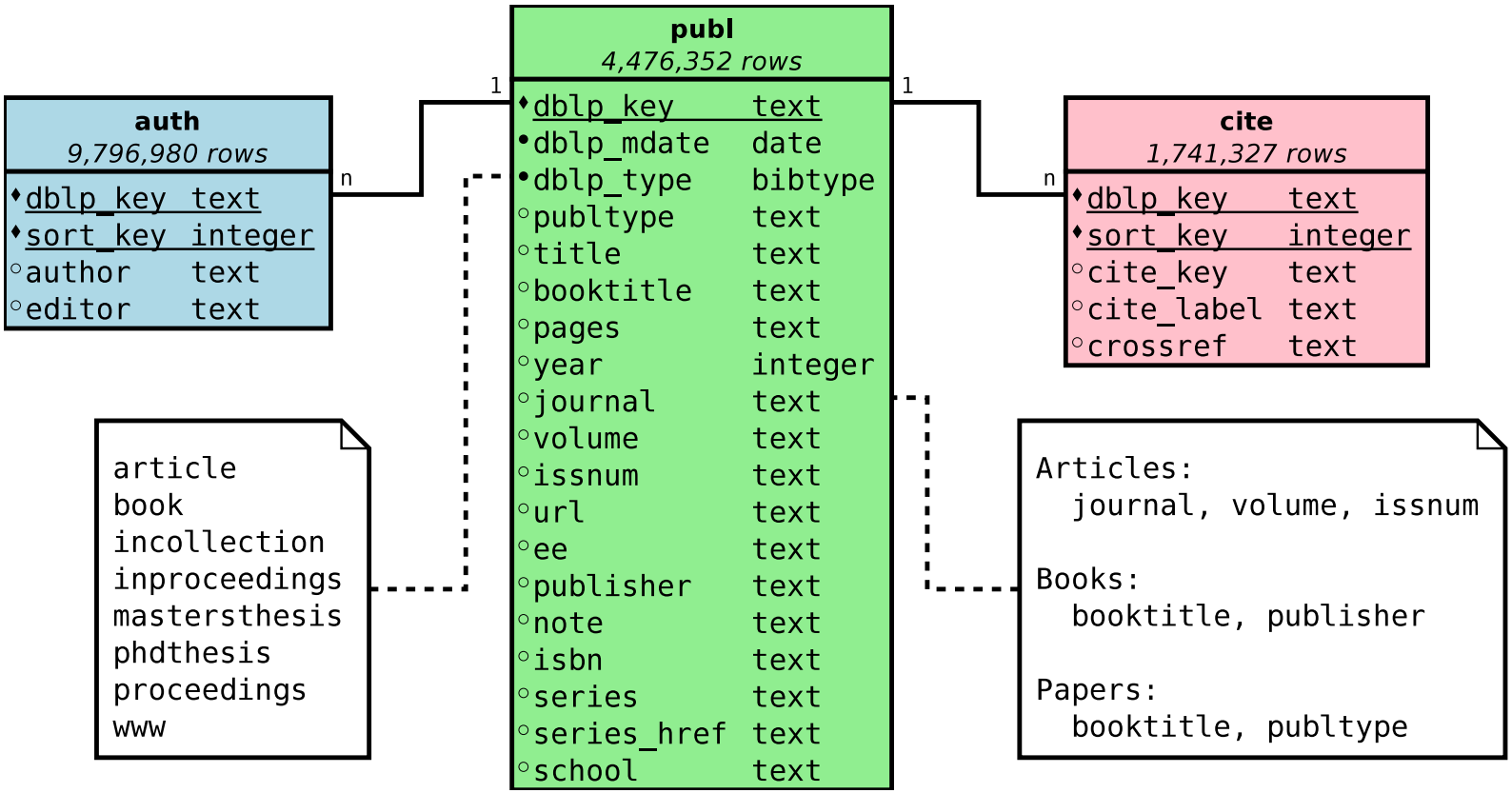
Dr. Chris Mayfield

Department of Computer Science
James Madison University

Mar 29, 2022

Hello DBLP

Database of CS journal articles and conference proceedings



Home page: https://dblp.org/

See also https://dl.acm.org/ and https://citeseerx.ist.psu.edu/

Example queries

```sql
-- find publications for an author
SELECT
  dblp_type, title, booktitle,
  journal, pages, ee, year
FROM publ
  NATURAL JOIN auth
WHERE author = 'Chris Mayfield'
  AND dblp_type <> 'www'
ORDER BY year DESC;

-- list proceedings of SIGMOD 2010
SELECT
  dblp_key, title, pages, ee
FROM publ
  NATURAL JOIN cite
WHERE crossref = 'conf/sigmod/2010'
ORDER BY split_part(pages, '-', 1)::integer;
```

Full Text Search

Using `tsvector` and `tsquery`

Motivation

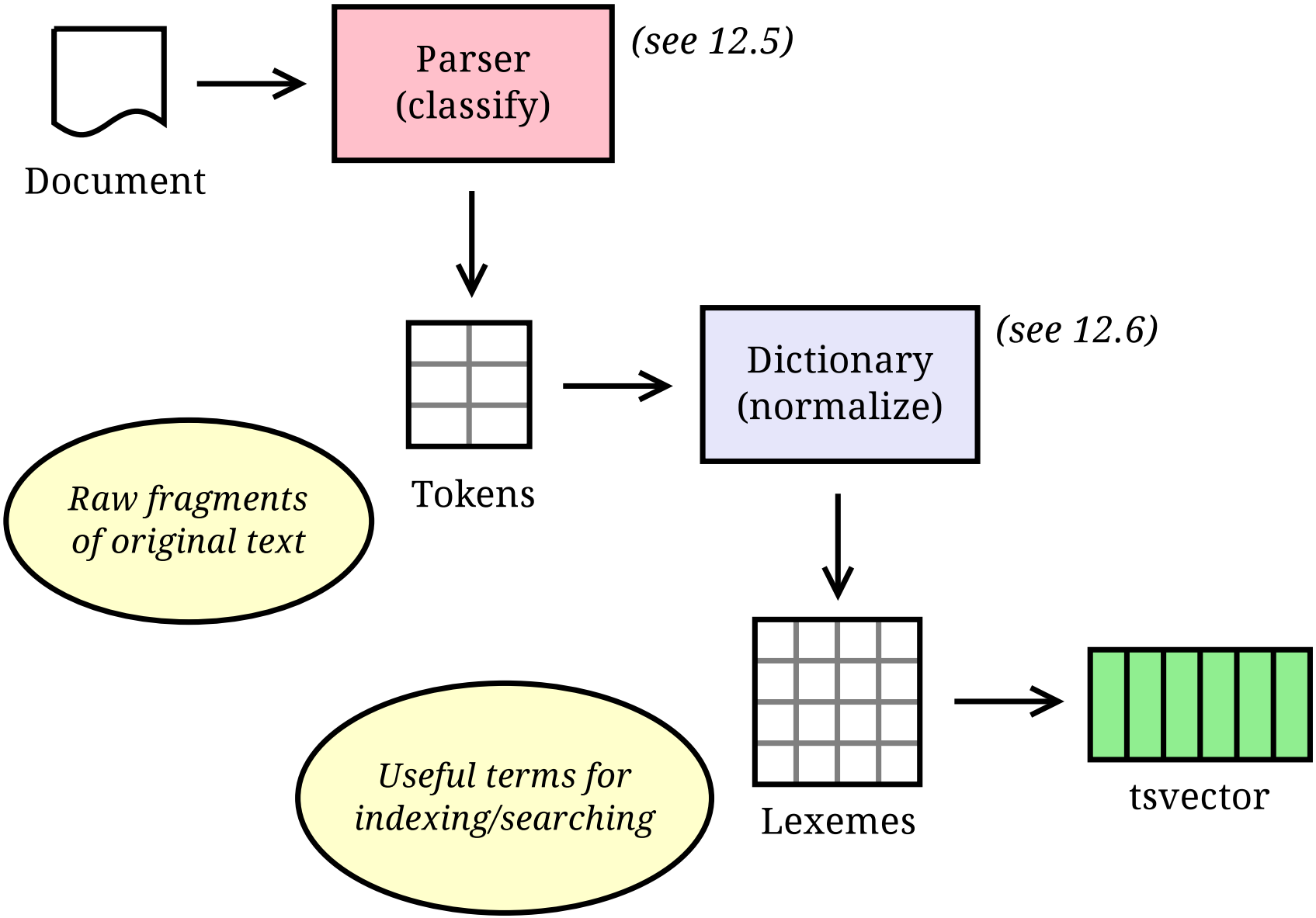
Write a query to select this paper: "ERACER: A Database Approach for Statistical Inference and Data Cleaning"

```
WHERE title = '...'

WHERE title LIKE '%Approach%'

WHERE title LIKE '%Approach%Cleaning%'
```

What's wrong with = and `LIKE`?

1. No linguistic support (cleaning vs cleansing)
2. No ranking of results (order by relevance)
3. Limited index support (must be left anchored)
4. Difficult to search for multiple words / phrases

How FTS works



Text search vectors

The `to_tsvector` function runs the whole process:

```
SELECT to_tsvector('Thanks for giving your BEST TRY!');
```

```
'best':5 'give':3 'thank':1 'tri':6
```

"A `tsvector` value is a sorted list of distinct lexemes, which are words that have been normalized to merge different variants of the same word."

(https://www.postgresql.org/docs/11/datatype-textsearch.html)

Text search queries

The `plainto_tsquery` function is similar:

```
SELECT plainto_tsquery('Try your best');
```

```
'tri' & 'best'
```

"A `tsquery` value stores lexemes that are to be searched for, and combines them honoring the Boolean operators `&`, `|`, and `!`."

(https://www.postgresql.org/docs/11/datatype-textsearch.html)

FTS example in SQL

Use the @@ operator to run full text search

```
tsvector @@ tsquery
tsquery  @@ tsvector
text @@ tsquery
text @@ text      -- to_tsvector(x) @@ plainto_tsquery(y)
```

For example:

```
SELECT * FROM publ
WHERE title @@ 'clean data';
```

- Sequential scan takes forever (i.e., 80 seconds)

Indexing tsvectors

1. Create new table/column of text search vectors

```
CREATE TABLE publ_fts AS
  SELECT dblp_key, to_tsvector(title) AS title_tsv
  FROM publ;
```

2. Create a *Generalized Inverted Index*

```
CREATE INDEX ON publ_fts USING gin (title_tsv);
```

3. Search the index and join the results

```
SELECT * FROM publ NATURAL JOIN publ_fts
WHERE title_tsv @@ plainto_tsquery('clean data');
```

- Bitmap index scan + Index join (about 100 ms)

https://www.postgresql.org/docs/11/textsearch-indexes.html

Debugging and statistics

See what parser/dictionary are doing:

```
SELECT * FROM ts_debug('Now this is clean data!');
```

Calculate document statistics:

```
SELECT *
FROM ts_stat('SELECT title_tsv FROM publ_fts')
ORDER BY nentry DESC, ndoc DESC, word
LIMIT 100;
```

DBLP `title` statistics

- Unique words: 496,644 / Total words: 26,426,910
- Most frequent word: *base* 337,533 docs / 343,480 entries

https://www.postgresql.org/docs/11/functions-textsearch.html

How dictionaries work

A dictionary is a function that accepts a token and returns:

- An array of lexemes, if token is known
- An empty array, if token is a stop word
- New version of the lexeme (if filtering)
- `NULL` if the token is not recognized

Example dictionaries:

- `simple` - remove the stop words (the, and, a, . . .)
- `snowball` - canonical form using *Snowball* stemmer
- `ispell` - use spell checker for word variations
- `synonym` - map related words to a single word
- `thesaurus` - map phrases to a single word

Ranking query results

Many ways to define relevance

- Built-in `ts_rank` function takes into account: lexical, proximity, and structural information

```
SELECT
  dblp_key,
  ts_headline(title, tsq) AS body,
  ts_rank(title_tsv, tsq) AS rank
FROM publ NATURAL JOIN publ_fts,
  plainto_tsquery('approach for cleaning data') AS tsq
WHERE title_tsv @@ tsq
ORDER BY rank DESC;
```

- Tip: call functions in `FROM` clause to reuse them
- Comma means `CROSS JOIN`, but it's only one row

https://www.postgresql.org/docs/11/textsearch-controls.html

Similarity Search

Using the `pg_trgm` module

Problem with real data

Based on the original CSV files from VDOE's website

```
SELECT div_num, sch_num, count(DISTINCT sch_name)
FROM fall_membership
GROUP BY div_num, sch_num
HAVING count(DISTINCT sch_name) > 2
ORDER BY count DESC;
```

div=029 sch=1371

- Thomas Jefferson High
- Thomas Jefferson High School
- Thomas Jefferson High School for Science and Technology
- Thomas Jefferson High for Science and Technology

div=043 sch=0600

- John RandolphTucker High
- John Randolph Tucker High
- Tucker High
- John Randolphtucker High

Similarity queries

How do you join with other data sets?

- Names of cities and counties in VA
- Names of schools (past and present)

One solution: approximate string matching

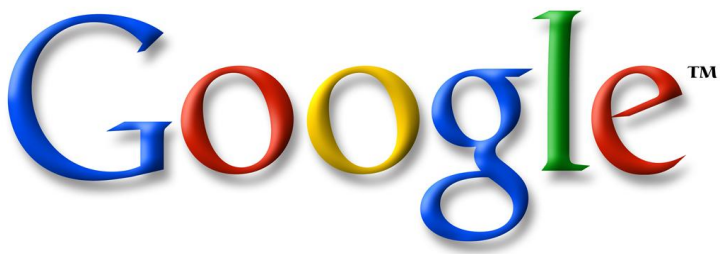
- Fuzzy string search
- Levenshtein distance
- Regular expressions
- Soundex / metaphone
- ...

Example: n-grams

"An n-gram is a contiguous sequence of n items from a given sequence of text or speech."

https://en.wikipedia.org/wiki/N-gram

https://books.google.com/ngrams/ (see About page)



Trigram (or trigraph)

https://www.postgresql.org/docs/11/pgtrgm.html

- 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.
- Each word is considered to have two spaces prefixed and one space suffixed.
- For example, the set of trigrams in the string `'cat'` is:
  `'  c'`, `' ca'`, `'cat'`, `'at '`

Trigram module

To install in your database:

```
CREATE EXTENSION pg_trgm;  -- must be superuser
```

Example queries:

```
-- how similar are the two strings?
SELECT similarity('Rockingham County',
                  'County of Rockingham');


-- true if similarity > threshold (0.3 by default)
SELECT set_limit(0.75);
SELECT 'Rockingham County' % 'County of Rockingham';
```

- GiST and GIN index support for %, `LIKE`, and regex

Text search integration

Use `simple` TS config to analyze originial (unstemmed) words

```
CREATE TABLE stats AS
  SELECT * FROM ts_stat('
    SELECT to_tsvector(''simple'', title) FROM publ');
```

GIN is faster to search than GiST, but slower to build/update

```
  CREATE INDEX ON stats USING gin (word gin_trgm_ops);
--CREATE INDEX ON stats USING gist (word gist_trgm_ops);
```

Suggest corrections for misspelled words

```
SELECT * FROM stats
WHERE word % 'algorithum'
ORDER BY ndoc DESC
LIMIT 5;
```