



Full text search & JSON

<Practice with full text search & JSON>



The task has been
prepared by
Yevhen Yermolenko



Full Text Search

<Let's explore PostgreSQL Full Text Search>

Full-Text Search

```
SELECT col FROM table WHERE col LIKE '%some_value%';
```

Full-Text Search

```
SELECT col FROM table WHERE col LIKE '%some_value%';
```

Operators ~, ~*, LIKE, ILIKE don't work well for full-text search

Full-Text Search

```
SELECT col FROM table WHERE col LIKE '%some_value%';
```

Operators ~, ~*, LIKE, ILIKE don't work well for full-text search

- No linguistic support (satisfies != satisfy);

Full-Text Search

```
SELECT col FROM table WHERE col LIKE '%some_value%';
```

WASTED

Operators ~, ~*, LIKE, ILIKE don't work well for full-text search

- No linguistic support (satisfies != satisfy);
- No ordering provided for large search result;

Full-Text Search

```
SELECT col FROM table WHERE col LIKE '%some_value%';
```

WASTED

Operators ~, ~*, LIKE, ILIKE don't work well for full-text search

- No linguistic support (satisfies != satisfy);
- No ordering provided for large search result;
- No index support -> slow search

Full-Text Search

Full-Text Search



Core Concepts:

Document – the context you want to search through.

Full-Text Search



Core Concepts:

Document – the context you want to search through.

Tokens – raw fragments of parsed document.

Full-Text Search



Core Concepts:

Document – the context you want to search through.

Tokens – raw fragments of parsed document.

Lexemes – normalized token (Days -> day).

Full-Text Search



Core Concepts:

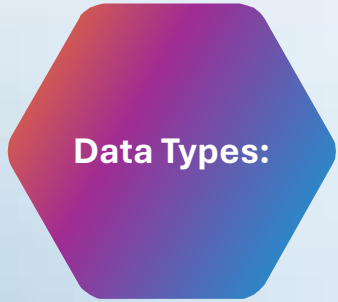
Document – the context you want to search through.

Tokens – raw fragments of parsed document.

Lexemes – normalized token (Days -> day).

Dictionaries – collection of lexemes.

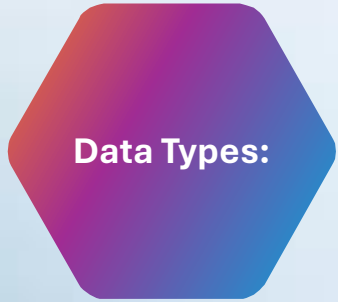
Full-Text Search



TSVECTOR

TSQUERY

Full-Text Search



TSVECTOR

representation of a document
optimized for searching

TSQUERY

Full-Text Search



Data Types:

TSVECTOR

representation of a document
optimized for searching

TSQUERY

representation of a search
query

Full-Text Search

Data Types:

TSVECTOR

representation of a document
optimized for searching

```
SELECT to_tsvector('The quick brown fox jumps over the lazy dog.');
```

```
to_tsvector
1 'brown':3 'dog':9 'fox':4 'jump':5 'lazi':8 'quick':2
```

TSQUERY

representation of a search
query

Full-Text Search

Data Types:

TSVECTOR

representation of a document
optimized for searching

```
SELECT to_tsvector('The quick brown fox jumps over the lazy dog.');
```

	to_tsvector
1	'brown':3 'dog':9 'fox':4 'jump':5 'lazi':8 'quick':2

TSQUERY

representation of a search
query

```
SELECT to_tsquery('jumping');
```

	to_tsquery
1	'jump'

Full-Text Search

Data Types:

TSVECTOR

representation of a document
optimized for searching

```
SELECT to_tsvector('The quick brown fox jumps over the lazy dog.');
```

	to_tsvector
1	'brown':3 'dog':9 'fox':4 'jump':5 'lazi':8 'quick':2

```
SELECT to_tsvector('jumps'),  
       to_tsvector('jumped'),  
       to_tsvector('jumping');
```

TSQUERY

representation of a search
query

```
SELECT to_tsquery('jumping');
```

	to_tsquery
1	'jump'

Full-Text Search

Data Types:

TSVECTOR

representation of a document
optimized for searching

```
SELECT to_tsvector('The quick brown fox jumps over the lazy dog.');
```

	to_tsvector
1	'brown':3 'dog':9 'fox':4 'jump':5 'lazi':8 'quick':2

```
SELECT to_tsvector('jumps'),  
       to_tsvector('jumped'),  
       to_tsvector('jumping');
```

	to_tsvector	to_tsvector	to_tsvector
1	'jump':1	'jump':1	'jump':1

TSQUERY

representation of a search
query

```
SELECT to_tsquery('jumping');
```

	to_tsquery
1	'jump'

Full-Text Search

Data Types:

TSVECTOR

representation of a document
optimized for searching

```
SELECT to_tsvector('The quick brown fox jumps over the lazy dog.');
```

	to_tsvector
1	'brown':3 'dog':9 'fox':4 'jump':5 'lazi':8 'quick':2

```
SELECT to_tsvector('jumps'),  
       to_tsvector('jumped'),  
       to_tsvector('jumping');
```

	to_tsvector	to_tsvector	to_tsvector
1	'jump':1	'jump':1	'jump':1

TSQUERY

representation of a search
query

```
SELECT to_tsquery('jumping');
```

	to_tsquery
1	'jump'

Include Boolean, negation,
group operators & | ! – ()

Full-Text Search

Data Types:

TSVECTOR

representation of a document
optimized for searching

```
SELECT to_tsvector('The quick brown fox jumps over the lazy dog.');
```

	to_tsvector
1	'brown':3 'dog':9 'fox':4 'jump':5 'lazi':8 'quick':2

```
SELECT to_tsvector('jumps'),  
       to_tsvector('jumped'),  
       to_tsvector('jumping');
```

	to_tsvector	to_tsvector	to_tsvector
1	'jump':1	'jump':1	'jump':1

TSQUERY

representation of a search
query

```
SELECT to_tsquery('jumping');
```

	to_tsquery
1	'jump'

Include Boolean, negation,
group operators & | ! – ()

plainto_tsquery transforms
the unformatted text to a
tsquery value. It doesn't
recognize operators & | ! – ()

Full-Text Search



PostgreSQL Full-Text Search Example

"Andi still learning
how to make a fried rice"

→ **to_tsvector()** →

"andi": 3, "learn": 1, "make": 1,
"fried rice": 2

→ **plainto_tsquery()** →

"andi" & "learn" & "make" &
"fried rice"

← **ts_rank()** ←

id	articles
4001	Budi with Ana learning how to make fried rice
7002	Andi doesn't like to make fried rice
1002	Another best fried rice in this city is from Jawa
...	...

Sorted by higher rank values correspond to more relevant search results

Full-Text Search

Operators

@@

Full-Text Search

Operators

@@ - evaluates the similarity between the text in a document and query

Full-Text Search

Operators

@@ - evaluates the similarity between the text in a document and query

tsvector @@ tsquery -> true/false

Full-Text Search

Operators

@@ - evaluates the similarity between the text in a document and query

tsvector @@ tsquery -> true/false

```
SELECT to_tsvector('The quick brown fox jumps over the lazy dog.')  
       @@ to_tsquery('jumping') result;
```

Full-Text Search

Operators

@@ - evaluates the similarity between the text in a document and query

tsvector @@ tsquery -> true/false

```
SELECT to_tsvector('The quick brown fox jumps over the lazy dog.')  
      @@ to_tsquery('jumping') result;
```

	result
1	true

Full-Text Search

Full-Text Search



Examples

```
CREATE TABLE posts(  
  id SERIAL PRIMARY KEY,  
  title TEXT NOT NULL,  
  body TEXT,  
  body_search TSVECTOR  
    GENERATED ALWAYS AS (to_tsvector('english',body)) STORED  
);
```

Full-Text Search



Examples

```
CREATE TABLE posts(  
  id SERIAL PRIMARY KEY,  
  title TEXT NOT NULL,  
  body TEXT,  
  body_search TSVECTOR  
  GENERATED ALWAYS AS (to_tsvector('english',body)) STORED  
);
```

```
INSERT INTO posts(title, body)  
VALUES  
  ('Introduction to PostgreSQL', 'This is an introductory post about PostgreSQL...'),  
  ('Advanced PostgreSQL Techniques', 'In this post, we delve into advanced...'),  
  ('PostgreSQL Optimization Strategies', 'This post explores various...');
```

Full-Text Search



Examples

```
CREATE TABLE posts(  
  id SERIAL PRIMARY KEY,  
  title TEXT NOT NULL,  
  body TEXT,  
  body_search TSVECTOR  
  GENERATED ALWAYS AS (to_tsvector('english',body)) STORED  
);
```

```
INSERT INTO posts(title, body)  
VALUES  
  ('Introduction to PostgreSQL', 'This is an introductory post about PostgreSQL...'),  
  ('Advanced PostgreSQL Techniques', 'In this post, we delve into advanced...'),  
  ('PostgreSQL Optimization Strategies', 'This post explores various...');
```

```
SELECT id, body_search FROM posts;
```


Full-Text Search

Examples

```
CREATE TABLE posts(  
  id SERIAL PRIMARY KEY,  
  title TEXT NOT NULL,  
  body TEXT,  
  body_search TSVECTOR  
  GENERATED ALWAYS AS (to_tsvector('english',body)) STORED  
);
```

```
INSERT INTO posts(title, body)  
VALUES  
  ('Introduction to PostgreSQL', 'This is an introductory post about PostgreSQL...'),  
  ('Advanced PostgreSQL Techniques', 'In this post, we delve into advanced...'),  
  ('PostgreSQL Optimization Strategies', 'This post explores various...');
```

```
SELECT id, body_search FROM posts;
```

```
id | body_search  
---+-----  
 1 | 'basic':10 'concept':11 'cover':9 'featur':13 'introductory':4 '  
 2 | 'advanc':7 'data':14 'delv':5 'effici':11 'manipul':15 'post':3 '  
 3 | 'databas':9 'effici':12 'explor':3 'optim':7 'perform':10 'post'  
(3 rows)
```

Full-Text Search



Examples

```
SELECT id, body  
FROM posts  
WHERE body_search @@ to_tsquery('PostgreSQL');
```

Full-Text Search

Examples

```
SELECT id, body
FROM posts
WHERE body_search @@ to_tsquery('PostgreSQL');
```

```
id | body
---+-----
1 | This is an introductory post about PostgreSQL. It covers basic concepts and features.
2 | In this post, we delve into advanced PostgreSQL techniques for efficient querying and
3 | This post explores various strategies for optimizing PostgreSQL database performance a
(3 rows)
```

Full-Text Search



Examples

```
CREATE INDEX pgweb_idx ON pgweb USING GIN (to_tsvector('english', body));
```

Full-Text Search



Examples

```
CREATE INDEX pgweb_idx ON pgweb USING GIN (to_tsvector('english', body));  
  
WHERE to_tsvector('english', body) @@ 'a & b'
```

Full-Text Search



Examples

```
CREATE INDEX pgweb_idx ON pgweb USING GIN (to_tsvector('english', body));
```

```
WHERE to_tsvector('english', body) @@ 'a & b'
```

```
WHERE to_tsvector(body) @@ 'a & b'
```

Full-Text Search



Examples

```
CREATE INDEX pgweb_idx ON pgweb USING GIN (to_tsvector('english', body));
```

```
WHERE to_tsvector('english', body) @@ 'a & b'
```

```
WHERE to_tsvector(body) @@ 'a & b'
```

Full-Text Search

Full-Text Search

Limitations

The length of each lexeme must be less than 2 kilobytes

Full-Text Search

Limitations

The length of each lexeme must be less than 2 kilobytes

The size of a tsvector (lexemes + positions) must be less than 1 megabyte

Full-Text Search

Limitations

The length of each lexeme must be less than 2 kilobytes

The size of a tsvector (lexemes + positions) must be less than 1 megabyte

The number of lexemes must be less than 2^{64}

Full-Text Search

Limitations

The length of each lexeme must be less than 2 kilobytes

The size of a tsvector (lexemes + positions) must be less than 1 megabyte

The number of lexemes must be less than 2^{64}

No more than 256 positions per lexeme

Full-Text Search

Full-Text Search



ADVANTAGES:

Cost Efficiency – no need for 3rd parties like ElasticSearch

Full-Text Search



ADVANTAGES:

Cost Efficiency – no need for 3rd parties like ElasticSearch

Data Consistency – no need to sync search engine with database engine

Full-Text Search



ADVANTAGES:

Cost Efficiency – no need for 3rd parties like ElasticSearch

Data Consistency – no need to sync search engine with database engine

Native support for multiple languages and custom dictionaries

JSON

JSON

JSON data types are for storing JSON (**JavaScript Object Notation**) data, as specified in **RFC 7159**. Such data can also be stored as text, but the JSON data types have the advantage of enforcing that each stored value is valid according to the JSON rules

JSON

JSON data types are for storing JSON (**JavaScript Object Notation**) data, as specified in **RFC 7159**. Such data can also be stored as text, but the JSON data types have the advantage of enforcing that each stored value is valid according to the JSON rules

PostgreSQL offers two types for storing JSON data: **json** and **jsonb**

JSON

JSON data types are for storing JSON (**JavaScript Object Notation**) data, as specified in **RFC 7159**. Such data can also be stored as text, but the JSON data types have the advantage of enforcing that each stored value is valid according to the JSON rules


PostgreSQL offers two types for storing JSON data: **json** and **jsonb** >>> **jsonpath**

JSON

JSON data types are for storing JSON (**JavaScript Object Notation**) data, as specified in **RFC 7159**. Such data can also be stored as text, but the JSON data types have the advantage of enforcing that each stored value is valid according to the JSON rules

PostgreSQL offers two types for storing JSON data: **json** and **jsonb** >>> **jsonpath**

stores an exact copy of the input text,
which processing functions
must reparse on each execution




JSON

JSON data types are for storing JSON (**JavaScript Object Notation**) data, as specified in **RFC 7159**. Such data can also be stored as text, but the JSON data types have the advantage of enforcing that each stored value is valid according to the JSON rules

PostgreSQL offers two types for storing JSON data: **json** and **jsonb** >>> **jsonpath**

stores an exact copy of the input text,
which processing functions
must reparse on each execution



stored in a decomposed binary format that makes it
slightly slower to input due to added conversion overhead,
but significantly faster to process.
Supports indexing

JSON

```
json -> integer → json
```

```
jsonb -> integer → jsonb
```

Extracts n 'th element of JSON array (array elements are indexed from zero, but negative integers count from the end).

```
'[{"a":"foo"}, {"b":"bar"}, {"c":"baz"}]' :: json -> 2 → {"c":"baz"}
```

```
'[{"a":"foo"}, {"b":"bar"}, {"c":"baz"}]' :: json -> -3 → {"a":"foo"}
```

JSON

```
json -> integer → json
```

```
jsonb -> integer → jsonb
```

Extracts n 'th element of JSON array (array elements are indexed from zero, but negative integers count from the end).

```
'[{"a":"foo"}, {"b":"bar"}, {"c":"baz"}]'::json -> 2 → {"c":"baz"}
```

```
'[{"a":"foo"}, {"b":"bar"}, {"c":"baz"}]'::json -> -3 → {"a":"foo"}
```

```
json -> text → json
```

```
jsonb -> text → jsonb
```

Extracts JSON object field with the given key.

```
'{"a": {"b":"foo"}}'::json -> 'a' → {"b":"foo"}
```


JSON

```
json -> integer → json
```

```
jsonb -> integer → jsonb
```

Extracts n 'th element of JSON array (array elements are indexed from zero, but negative integers count from the end).

```
'[{"a":"foo"}, {"b":"bar"}, {"c":"baz"}]':::json -> 2 → {"c":"baz"}
```

```
'[{"a":"foo"}, {"b":"bar"}, {"c":"baz"}]':::json -> -3 → {"a":"foo"}
```

```
json -> text → json
```

```
jsonb -> text → jsonb
```

Extracts JSON object field with the given key.

```
'{"a": {"b":"foo"}}':::json -> 'a' → {"b":"foo"}
```

```
json ->> text → text
```

```
jsonb ->> text → text
```

Extracts JSON object field with the given key, as text.

```
'{"a":1,"b":2}':::json ->> 'b' → 2
```

Full-Text Search & JSON

Resources:

[PostgreSQL Full-Text Search: A Powerful Alternative to Elasticsearch for Small to Medium Applications](#)

[PostgreSQL Full Text Search](#)

[Official documentation](#) for full-text search

[Official PostgreSQL documentation](#) for JSON

Thank you

- Author: Yevhen Yermolenko
- My LinkedIn: <https://www.linkedin.com/in/yerm/>
- May 2025
- [Join Codeus community in Discord](#)
- [Join Codeus community in LinkedIn](#)