



Министерство науки и высшего образования Российской Федерации  
Федеральное государственное бюджетное образовательное учреждение  
высшего образования  
«Московский государственный технический университет  
имени Н.Э. Баумана  
(национальный исследовательский университет)»  
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ Информатика, искусственный интеллект и системы управления

КАФЕДРА Системы обработки информации и управления

---

**Методические указания к лабораторным работам  
по курсу «Постреляционные базы данных»**

**Лабораторная работа №7  
«Полнотекстовый поиск в среде СУБД PostgreSQL»**

Виноградова М.В., Елисеева Е.А

Под редакцией к.т.н. доц. Виноградовой М.В.

Москва, 2023г.

## ОГЛАВЛЕНИЕ

1. ЗАДАНИЕ .....	3
Цель работы .....	3
Средства выполнения .....	3
Продолжительность работы .....	3
Пункты задания для выполнения .....	3
Содержание отчета .....	4
2. ТЕОРЕТИКО-ПРАКТИЧЕСКИЙ МАТЕРИАЛ .....	5
Контрольные вопросы .....	20
3. СПИСОК ИСТОЧНИКОВ .....	20

# 1. ЗАДАНИЕ

Лабораторная работа №7 «Полнотекстовый поиск в среде СУБД PostgreSQL» по курсу «Постреляционные базы данных».

## Цель работы

- Изучить методы работы с полнотекстовым поиском БД PostgreSQL.
- Освоить ранжирование и подсветку результата в полнотекстовом поиске.
- Получить навыки работы с объектно-реляционной БД PostgreSQL.

## Средства выполнения

- СУБД PostgreSQL;
- Утилита PostgreSQL.

## Продолжительность работы

Время выполнения лабораторной работы 4 часа.

## Пункты задания для выполнения

1. Определить систему таблиц в среде PostgreSQL (*базовая часть*).
2. Указать в таблицах атрибуты составных типов (диапазоны, геометрические типы, enum, JSON) и продемонстрировать работу с ними, используя встроенные функции (*базовая часть – хотя бы один тип, расширенная часть – все типы*).
3. Продemonстрировать полнотекстовый поиск используя ts vector и ts query. Поиск через ts vector и ts query. Использование операторов «и», «или», предшествование и соответствие (*базовая часть*).
4. Продemonстрировать полнотекстовый поиск используя свои словари и файлы со стоп-словами (*базовая часть*).
5. Продemonстрировать полнотекстовый поиск используя словари синонимов и тезаурусы (*расширенная часть*).
6. Продemonстрировать полнотекстовый поиск используя индексы для текстового поиска (*дополнительная часть*).

7. Продемонстрировать полнотекстовый поиск используя ранжирование и подсветку результата (*дополнительная часть*).

### **Содержание отчета**

- Титульный лист;
- Цель работы;
- Задание;
- Тексты запросов и команд в соответствии с пунктами задания.
- Результаты выполнения запросов (скриншоты);
- Вывод;
- Список используемой литературы.

## 2. ТЕОРЕТИКО-ПРАКТИЧЕСКИЙ МАТЕРИАЛ

### ■ Основы поиска (<https://postgrespro.ru/docs/postgresql/12/textsearch>)

Полнотекстовый поиск (или просто *поиск текста*) — это возможность находить **документы** на естественном языке, соответствующие *запросу*, и, возможно, дополнительно сортировать их по релевантности для этого запроса.

В контексте поиска в Postgres **документ** — это содержимое текстового поля в строке таблицы или объединение таких полей, которые могут храниться в разных таблицах или формироваться динамически.

Полнотекстовый поиск в Postgres реализован на базе **оператора соответствия @@**, который возвращает true, если ***tsvector*** (то есть документ) соответствует ***tsquery*** (поисковому запросу).

При формировании запросов применяют следующие операторы:

1) **&** (И) указывает, что оба операнда должны присутствовать в документе, чтобы он удовлетворял запросу.

2) **|** (ИЛИ) указывает, что в документе должен присутствовать минимум один из его операндов.

3) **<->** (ПРЕДШЕСТВУЕТ) находит соответствие, только если операнды расположены рядом и в заданном порядке.

Важной особенностью поиска в Postgres является использование индексов. **Индексы** назначаются в соответствие каждому из слов, среди которых будет проводиться поиск. Наличие индексов позволяет системе производить поиск не по словам, а по назначенным им индексам, и поэтому скорость поиска возрастает.

Также при поиске часто встречается понятие лексемы. **Лексема** — это нормализованное слово, то есть слово, в соответствие которому поставлены некоторые его словоформы (например, «врач» может считаться лексемой для «врачу», «врачами», «врачом» и т. п.)

Для поиска принято использовать словари:

- 1) **Общие словари**, которые есть в Postgres по умолчанию и содержат все слова. Среди них словари *simple*, стеммер *Snowball* (стеммер – это особый словарь, в котором каждому слову ставится в соответствие его основа, то есть убирается окончание).
- 2) **Словари со стоп-словами**, в которые помещают часто встречающиеся слова, не несущие смысловой нагрузки для поиска (такие словари пишутся пользователями и зависят от целей поиска).
- 3) **Словари синонимов** (такие словари пишутся пользователями с учётом особенностей предметной области базы данных).

Функции, которые часто используются в полнотекстовом поиске:

- **ts\_lexize** (возвращает массив лексем, если входной фрагмент известен словарю, либо пустой массив, если этот фрагмент считается в словаре стоп-словом, либо NULL, если он не был распознан);
- **to\_tsvector** (разбирает текстовый документ на фрагменты, сводит фрагменты к лексемам и возвращает значение tsvector, в котором перечисляются лексемы и их позиции в документе заданного запроса);
- **to\_tsquery** (нормализует введённые слова и приводит их к типу tsquery для возможности их дальнейшего сравнения с tsvector при помощи оператора @@);
- **plainto\_tsquery** (принимает на вход строку и приводит её так же, как to\_tsquery, но без необходимости заранее разбивать фразу).

Приведём простой пример работы функции **to\_tsvector** (рис. 34).

**SELECT** to\_tsvector ('english', 'a fat cat sat on a mat - it ate a fat rats');

Data Output	Explain	Messages	Notifications
<div> <div>to_tsvector</div> <div>tsvector</div> </div>			
1	'ate':9 'cat':3 'fat':2,11 'mat':7 'rat':12 'sat':4		

Рис. 34. Функция to\_tsvector

В данном примере первый параметр, переданный в функцию, – ‘english’ – это конфигурация по умолчанию для английского языка. Вторым параметром, переданным в функцию – это строка слов. Результатом работы функции `to_tsvector` является набор лексем, которые были распознаны словарями, а также индексы, присвоенные найденным словам. В нашем случае артикли «а», предлог «on» и местоимение «it» были проиндексированы, однако не попали в вывод функции, так как по умолчанию они являются стоп-словами (подробнее со стоп-словами можно ознакомиться в следующем разделе методических указаний).

Аналогичным образом работает обычное преобразование строки к типу `tsvector`: мы получаем отсортированный список неповторяющихся лексем, т. е. слов, нормализованных так, что все словоформы сводятся к одной, однако без индексации и без исключения стоп-слов (рис. 35).

```
1 SELECT 'a fat cat sat on a mat and ate a fat rat'::tsvector;
```

	Data Output	Explain	Messages	Notifications
	tsvector tsvector			
1	'a' 'and' 'ate' 'cat' 'fat' 'mat' 'on' 'rat' 'sat'			

Рис. 35. Тип `tsvector`

Приведём простой пример работы функции `to_tsquery` (рис. 36).

`SELECT to_tsquery('english', 'The & Fat & Rats');`

	Data Output	Explain	Messages	Notifications
	to_tsquery tsquery			
1	'fat' & 'rat'			

Рис. 36. Функция `to_tsquery`

Функция `to_tsquery` используется для приведения запроса к типу данных `tsquery`. В приведённом примере входным значением для функции `to_tsquery`

является набор слов, разделённых оператором «&» (также можно использовать и другие операторы: «|», «!» или «<->»), а выходным значением – нормализованные фрагменты, приведённые к лексемам. То есть `to_tsquery` привела слова к нижнему регистру, распознала слова «the», «fat» и «rat», отбросила артикль «the», так как он является стоп-словом, и вернула полученный результат.

Аналогичным образом работает обычное преобразование строки к типу `tsquery`: мы получаем искомые лексемы, объединяемые логическими операторами «&» (И), «|» (ИЛИ) и «!» (НЕ), а также оператором поиска фраз «<->» (ПРЕДШЕСТВУЕТ) (рис. 37).

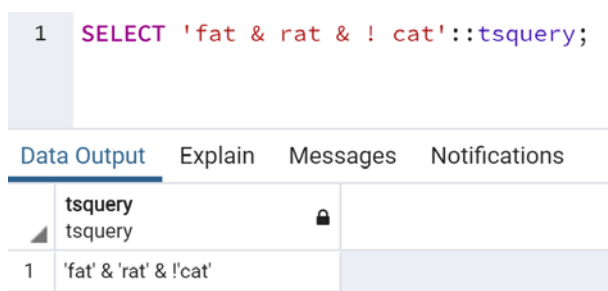


Рис. 37. Тип `tsquery`

#### ▪ Словари и файлы со стоп-словами

(<https://postgrespro.ru/docs/postgresql/12/textsearch-dictionaries>)

Словари полнотекстового поиска предназначены для исключения **стоп-слов** (слов, которые не должны учитываться при поиске) и **нормализации слов**, (чтобы разные словоформы считались совпадающими).

**Словарь** — это программа, которая принимает на вход фрагмент текста и возвращает:

- массив лексем, если входной фрагмент известен в словаре;
- пустой массив, если словарь воспринимает этот фрагмент, но считает его стоп-словом;
- NULL, если словарь не воспринимает полученный фрагмент.

Общее **правило настройки списка словарей** (выбора словарей, которые будут использоваться при поиске слов в текстовых полях) заключается в том, чтобы



поставить наиболее частные и специфические словари в начале, затем перечислить более общие и закончить самым общим словарём, который распознаёт все слова.

**Стоп-словами** называются слова, которые встречаются очень часто, практически в каждом документе, и поэтому не имеют различительной ценности. Таким образом, при полнотекстовом поиске их можно игнорировать. Например, в каждом английском тексте содержатся артикли *a* и *the*, которые не несут смысловой нагрузки при поиске, поэтому они внесены в стоп-лист.

Работа шаблона словарей *simple* сводится к преобразованию входного фрагмента в нижний регистр и проверки существования полученных слов в файле со списком стоп-слов. Если это слово находится в файле, словарь возвращает пустой массив и фрагмент исключается из дальнейшего рассмотрения. В противном случае словарь возвращает в качестве нормализованной лексемы слово в нижнем регистре.

Определим свой собственный словарь на основе шаблона *simple* (рис. 38).

```
1 CREATE TEXT SEARCH DICTIONARY public.simple_dict (  
2     TEMPLATE = pg_catalog.simple,  
3     STOPWORDS = my_own_stop_word_dict  
4 );
```

*Рис. 38. Добавление собственного словаря со стоп-словами*

Здесь *my\_own\_stop\_word\_dict* — имя файла со стоп-словами. Полным именем файла будет *\$SHAREDIR/tsearch\_data/my\_own\_stop\_word\_dict.stop*, где *\$SHAREDIR* указывает на каталог с общими данными PostgreSQL (например, полный путь: *C:\Program Files\PostgreSQL\12\share\tsearch\_data\my\_own\_stop\_word\_dict.stop*).

Этот текстовый файл должен содержать просто список слов, по одному слову в строке. Пустые строки и окружающие пробелы игнорируются, все символы переводятся в нижний регистр, и на этом обработка файла заканчивается.

Например, наш файл *my\_own\_stop\_word\_dict.stop* содержит строки:

*hi*

hello  
bye  
goodbye  
oh  
omg  
wow  
like  
anyway  
right  
really  
ok  
basically  
whatever

Используем функцию `ts_lexize` и убедимся, что слова из стоп-словаря не будут учтены при поиске: результатом запроса `ts_lexize` для 'wOw' будет '{}', так как данное слово присутствует в нашем пользовательском словаре стоп-слов (рис. 39).

```
7 SELECT ts_lexize('public.simple_dict','wOw');
```

Data Output		Expl.
ts_lexize text[]		
1	{}	

Рис. 39. Проверка работы словаря со стоп-словами

- **Словари синонимов** (<https://postgrespro.ru/docs/postgresql/12/textsearch-dictionaries#TEXTSEARCH-SYNONYM-DICTIONARY>)

Этот шаблон словарей используется для создания словарей, заменяющих слова синонимами. Такие словари не поддерживают словосочетания.

Для добавления собственного словаря синонимов в FTS Dictionaries (см. дерево объектов) необходимо выполнить команду:

```
3 CREATE TEXT SEARCH DICTIONARY my_own_synonym (  
4     TEMPLATE = synonym,  
5     SYNONYMS = my_own_synonyms  
6 );
```

*Рис. 40. Добавление собственного словаря синонимов*

Шаблон `synonym` принимает единственный параметр: `SYNONYMS`, в котором задаётся имя словаря синонимов — `my_own_synonyms`. Полным именем файла будет `$SHAREDIR/tsearch_data/my_own_synonyms.syn` (где `$SHAREDIR` указывает на каталог общих данных PostgreSQL, например, полный путь: `C:\Program Files\PostgreSQL\12\share\tsearch_data\ my_own_synonyms.syn`).

Содержимое этого файла должны составлять строки с двумя словами в каждой (первое — заменяемое слово, а второе — его синоним), разделёнными пробелами. Пустые строки и окружающие пробелы при разборе этого файла игнорируются.

Например, пусть «`my_own_synonyms.syn`» содержит строки:

```
Paris paris  
something smth  
somebody smb  
versus vs  
senior sr  
reply re  
reference rf  
education edu  
inch in  
second sec  
refrigerator fridge  
comfortable comfy
```

doctor doc\*

В конце синонима в этом файле можно добавить звёздочку (\*), тогда этот синоним будет рассматриваться как префикс.

Поиск лексемы в словаре можно выполнить командой `ts_lexize` (см. рис. 41).

5	<code>SELECT ts_lexize('my_own_synonym','refrigerator');</code>	Data Output	Explain
		ts_lexize text[]	
1		{fridge}	

Рис. 41. Поиск лексемы в словаре

Чтобы применять добавленный словарь во время полнотекстового поиска, нужно создать конфигурацию, которая будет его задействовать (см. рис. 42). После этого, выполнив команду `to_tsvector` (см. рис. 43), получим набор лексем и их позиции в документе заданного запроса (первый принимаемый параметр функции – ‘`tst_config`’ – это имя созданной нами конфигурации поиска, второй параметр – строка для поиска).

```
1 CREATE TEXT SEARCH CONFIGURATION tst_config (copy=simple);
2 ALTER TEXT SEARCH CONFIGURATION tst_config ALTER MAPPING FOR asciiword
3 WITH my_own_synonym;
```

Рис. 42. Изменение конфигурации поиска

```
5 SELECT to_tsvector('tst_config','doctor comfortable education versus second');
```

	Data Output	Explain	Messages
	to_tsvector tsvector		
1	'comfy':2 'doc':1 'edu':3 'sec':5 'vs':4		

Рис. 43. Проверка конфигурации

Осуществив запрос поиска с использованием данной конфигурации при помощи `tsvector @@ to_tsquery`, увидим положительный результат, подтверждающий корректную работу пользовательского словаря (рис. 44).

```
4 SELECT 'Smb advised him to get the edu in Paris to become a doc.'::tsvector
5 @@ to_tsquery('tst_config','doctor');
```

Data Output		Explain
	?column? boolean	
1	true	

Рис. 44. Поиск с использованием словаря синонимов

Таким же образом, с указанием нужной конфигурации, проводится поиск в полях базы данных.

- **Тезаурусы** (<https://postgrespro.ru/docs/postgresql/12/textsearch-dictionaries#TEXTSEARCH-THESAURUS>)

Тезаурус содержит набор слов и информацию о связях слов и словосочетаний, то есть более широкие понятия, более узкие понятия, предпочитаемые названия, исключаемые названия, связанные понятия и т. д.

В основном тезаурус заменяет исключаемые слова и словосочетания предпочитаемыми и может также сохранить исходные слова для индексации. Текущая реализация тезауруса в PostgreSQL представляет собой расширение словаря синонимов с поддержкой фраз.

Для создания нового словаря-тезауруса используется шаблон `thesaurus` (см. рис. 45). Здесь «`thesaurus_it`» - имя нового словаря, «`my_own_thesaurus`» - имя файла тезауруса (Полным путём к файлу будет `$SHAREDIR/tsearch_data/my_own_thesaurus.ths`, где `$SHAREDIR` указывает на каталог общих данных PostgreSQL. Например, полный путь: `C:\Program Files\PostgreSQL\12\share\tsearch_data\my_own_thesaurus.ths.`), «`english_stem`» - внутренний словарь (в данном случае это стеммер Snowball для английского) для нормализации тезауруса.

```

1 CREATE TEXT SEARCH DICTIONARY thesaurus_it (
2     TEMPLATE = thesaurus,
3     DictFile = my_own_thesaurus,
4     Dictionary = english_stem
5 );

```

Рис. 45. Добавление собственного тезауруса

Например, пусть содержимое нового тезауруса «thesaurus\_it.ths»:

content management systems : cms  
corporate information system : cis  
customer relationship management : crm  
virtual private network : vpn  
application service provider : asp

Для использования тезауруса «thesaurus\_it» его нужно связать с нужной конфигурацией (см. рис. 46).

```
7 ALTER TEXT SEARCH CONFIGURATION english
8     ALTER MAPPING FOR asciiword, asciihword, hword_asciipart
9     WITH thesaurus_it, english_stem;
```

Рис. 46. Изменение конфигурации поиска

После этого можно проверить работу тезауруса. Функция ts\_lexize не очень полезна для проверки тезауруса, так как она обрабатывает входную строку как один фрагмент. Вместо неё лучше использовать функции plainto\_tsquery и to\_tsvector, которые разбивают входную строку на несколько фрагментов (см. рис. 47-48). При этом фразы из тезауруса, написанные во множественном числе, тоже будут восприниматься конфигурацией (например, «virtual private networks» как «vpn»), так как в определении тезауруса был подключен стеммер english\_stem.

```
7 SELECT plainto_tsquery('Content management systems are popular site management tools.');
```

Data Output	Explain	Messages	Not
plainto_tsquery tsquery			🔒
1	'cms' & 'popular' & 'site' & 'manag' & 'tool'		

Рис. 47. Тестирование тезауруса: plainto\_tsquery

```
9 SELECT to_tsvector('Virtual private network is now a common method to stay protected on the network.');
```

Data Output	Explain	Messages	Notifications
to_tsvector tsvector			🔒
1	'common':5 'method':6 'network':12 'protect':9 'stay':8 'vpn':1		

Рис. 48. Тестирование тезауруса: `to_tsvector`

Чтобы проиндексировать исходную фразу вместе с заменой (рис. 49), её нужно добавить в правую часть соответствующего определения в тезаурусе:

corporate information system : cis corporate information system

```
12 SELECT plainto_tsquery('The success of an organization also depends on the corporate information system.');
```

Data Output	Explain	Messages	Notifi
plainto_tsquery tsquery			
1	'success' & 'organ' & 'also' & 'depend' & 'cis'		

Data Output	Explain	Messages	Notifications
plainto_tsquery tsquery			
1	'success' & 'organ' & 'also' & 'depend' & 'corpor' & 'inform' & 'system' & 'cis'		

Рис. 49. Тестирование тезауруса: до и после индексации исходной фразы

Аналогичным образом при помощи пользовательского тезауруса можно производить поиск и в других документах базы данных.

#### ■ Индексы для текстового поиска

(<https://postgrespro.ru/docs/postgrespro/9.5/textsearch-tables#textsearch-tables-index>)

Для ускорения полнотекстового поиска можно использовать индексы двух видов:

- **GIN** (Generalized Inverted Index – Обобщённый Инвертированный Индекс). Столбец должен иметь тип `tsvector`.
- **GIST** (Generalized Search Tree - Обобщённое дерево поиска). Здесь столбец может иметь тип `tsvector` или `tsquery`.

Эти индексы не требуются для поиска, но если по какому-то столбцу таблицы поиск выполняется регулярно, обычно желательно её индексировать.

Более предпочтительными для текстового поиска являются индексы `GIN`, поэтому создадим такой индекс для столбца `body` таблицы `text_`.

Добавим в таблицу `text_` столбец `tsvector`, в котором сохраним результат `to_tsvector` (рис. 50) Функцию **coalesce** применяют при конкатенации нескольких столбцов для их корректного соединения, в случае если один из них

может оказаться NULL. Это поможет избежать появления слова “NULL” в результирующем столбце. Таким образом, мы получим в таблице столбец «textsearchable\_index\_col», в котором будут находиться проиндексированные слова из столбца «body». Полученный столбец мы будем использовать в дальнейшем для ускорения процесса поиска.

```
2 ALTER TABLE text_
3     ADD COLUMN textsearchable_index_col tsvector
4     GENERATED ALWAYS AS (to_tsvector('english', coalesce(body, ''))) STORED;
```

	textid [PK] integer	orderid integer	information jsonb	body text	textsearchable_index_col tsvector
1		1	{ "type": "novel", "title": "Fahrenheit 451", "author": { "last...	Ray Bradbury Fahrenheit 451 This on...	'11':9865 '1555':11028 '16':11027 '1790':9469 '451':4...
2		2	{ "type": "poems", "title": "Sigh No More", "author": { "last...	Sigh no more, ladies, sigh no more,	'blith':36 'bonni':38 'constant':23 'convert':39 'deceiv':1...
3		3	{ "type": "article", "title": "Famed Roman statue not anci...	A statue symbolising the mythical or...	'1200s':26 '13th':143 '15th':80 '18th':164 '1960':308 '1...
4		4	{ "type": "article", "title": "Magic Mushrooms May Perma...	Just one strong dose of hallucinoge...	'14':56,613 '1960s':245 '25':105 '29':629 '30':107,732 '...
5		5	{ "type": "article", "title": "The Worlds Largest Indoor Far...	A former Sony Corporation semicon...	'000':63,95 '10':62,260 '100':268 '15':119 '17':109 '18':...

Рис. 50. Добавление столбца типа tsvector для хранения индексов

Далее создадим индекс GIN для ускорения поиска (рис. 51).

```
9 CREATE INDEX textsearch_idx ON text_ USING GIN (textsearchable_index_col);
```

Рис. 51. Создание индекса GIN

После этого можно быстро выполнять полнотекстовый поиск в столбце «textsearchable\_index\_col», используя созданный индекс «textsearch\_idx» (рис. 52).

```
12 SELECT orderid, information->'type' AS type, information->'author'->'last_name' AS author
13 FROM text_
14 WHERE textsearchable_index_col @@ to_tsquery('temperature')
15 ORDER BY orderid DESC
16 LIMIT 5;
```

Data Output	Explain	Messages	N
orderid integer	type jsonb	author jsonb	
1	9 "article"	""	
2	8 "novel"	"Bradbury"	

Рис. 52. Полнотекстовый поиск при помощи индекса

## ■ Ранжирование и подсветка результата

(<https://postgrespro.ru/docs/postgresql/12/textsearch-controls>)



Ранжирование документов можно представить как попытку оценить, насколько они релевантны заданному запросу, и отсортировать их так, чтобы наиболее релевантные выводились первыми. В Postgres встроены две функции ранжирования:

- ***ts\_rank*** ранжирует векторы по частоте найденных лексем;
- ***ts\_rank\_cd*** вычисляет *плотность покрытия* (то есть в расчёт берётся близость соответствующих лексем друг к другу) для данного вектора документа и запроса.

Функции принимают во внимание лексическую, позиционную и структурную информацию, то есть они учитывают, насколько часто и насколько близко встречаются в документе ключевые слова и какова важность содержащей их части документа.

Для примера вычислим ранг результатов запроса поиска последовательности слов «sigh no more» в столбце «body» таблицы «text\_» и выберем три найденных документа с максимальным рангом (рис. 53).

```
2 SELECT orderid, information->'type' AS type, information->'author'->'last_name' AS author,
3      ts_rank_cd(to_tsvector('english', body), query) AS rank
4 FROM text_, to_tsquery('english', 'sigh<->no<->more') query
5 WHERE query @@ to_tsvector('english', body)
6 ORDER BY rank DESC
7 LIMIT 3;
```

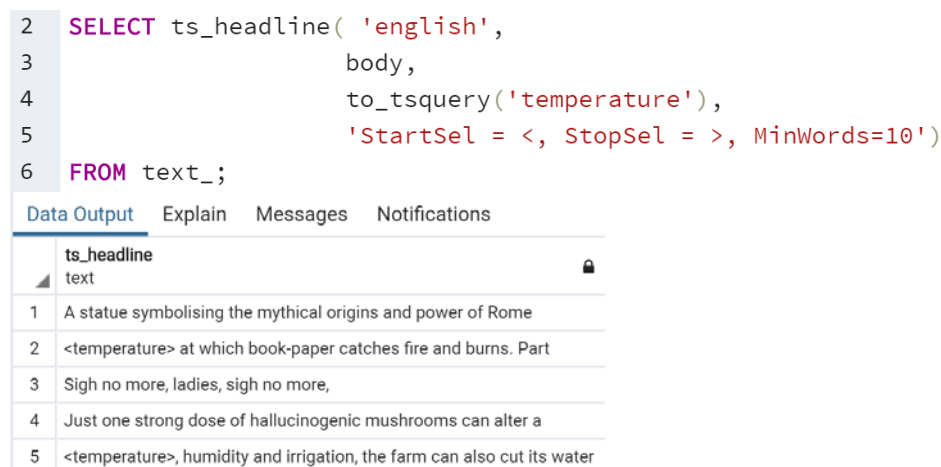
Data Output					Explain	Messages	Notifications
	orderid		type	author	rank		
	integer		jsonb	jsonb	real		
1	3	"poems"	"Shakespeare"	0.3			
2	8	"novel"	"Bradbury"	0.1			

Рис. 53. Ранжирование результатов поиска

Представляя результаты поиска, в идеале нужно выделять часть документа и показывать, как он связан с запросом. Обычно поисковые системы показывают фрагменты документа с отмеченными искомыми словами. В Postgres для реализации этой возможности представлена функция ***ts\_headline***.

Например, можно выделить результаты поиска слова «temperature» в колонке body таблицы text\_, указав при этом, что разграничивать слова следует знаками

« < > », а минимальное количество слов, выводимых по запросу должно быть равно 10 (см. рис. 54).



The screenshot shows a SQL query editor with the following code:

```
2 SELECT ts_headline( 'english',
3                     body,
4                     to_tsquery('temperature'),
5                     'StartSel = <, StopSel = >, MinWords=10')
6 FROM text_;
```

Below the query, there are tabs for 'Data Output', 'Explain', 'Messages', and 'Notifications'. The 'Data Output' tab is active, showing a table with the following data:

	ts_headline text
1	A statue symbolising the mythical origins and power of Rome
2	<temperature> at which book-paper catches fire and burns. Part
3	Sigh no more, ladies, sigh no more,
4	Just one strong dose of hallucinogenic mushrooms can alter a
5	<temperature>, humidity and irrigation, the farm can also cut its water

Рис. 54. Выделение результатов поиска

Если в параметрах передаётся строка опций, она должна состоять из списка разделённых запятыми пар параметр=значение. Параметры могут быть следующими:

- **StartSel**, **StopSel**: строки, которые будут разграничивать слова запроса в документе;
- **MaxWords**, **MinWords**: эти числа определяет нижний и верхний предел размера выдержки;
- **ShortWord**: слова такой длины или короче в начале и конце выдержки будут отбрасываться;
- **HighlightAll**: логический флаг; если он равен true, выдержкой будет весь документ;
- **MaxFragments**: максимальное число выводимых текстовых выдержек или фрагментов;
- **FragmentDelimiter**: Когда выводятся несколько фрагментов, они будут разделяться этой строкой.

Значения этих параметров по умолчанию:

- **StartSel**=<b>, **StopSel**=</b>;
- **MaxWords**=35, **MinWords**=15;

- *ShortWord*=3;
- *HighlightAll*=FALSE;
- *MaxFragments*=0;
- *FragmentDelimiter*=” ... ”.

### **3. Контрольные вопросы**

1. В чем особенности объектно-реляционных СУБД?
2. Какие применяют операторы при формировании запросов?
3. Что такое словарь? Виды словарей, применяемые для поиска.
4. Как реализовать поиск через ts vector и ts query?
5. Как применяются файлы со стоп-словами?
6. Что такое тезаурусы и в чем их особенности?
7. Как работают индексы для текстового поиска?
8. Как организована подсветка результата и ранжирование?

### **4. СПИСОК ИСТОЧНИКОВ**

1. Документация PostgreSQL <https://postgrespro.ru/docs/postgresql/>
2. Теория - <https://habr.com/ru/articles/40218/>