#### Задание 1

Выполним запрос, используя функцию to\_tsvector.

```
1 SELECT to_tsvector('Съешь ещё этих мягких французских булок, да выпей чаю');
```

Результатом выполнения запроса будет являться следующая строка лексем



Изучив документацию, выясняется, что «да» не отображается в векторе, так как является «стоп-словом» - встречается слишком, чтобы поиск по ним был осмысленным.

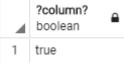
## Задание 2

# Выполним седующие запросы.

```
1 SELECT to_tsvector('The quick brown fox jumped over the lazy dog')
2 @@ to_tsquery('fox');

Результат:
```

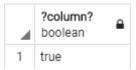
# Результат:



# Следующим запросом является:

```
1 SELECT to_tsvector('The quick brown fox jumped over the lazy dog')
2 @@ to_tsquery('foxes');
```

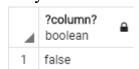
# Результат:



#### Последний запрос:

```
1 SELECT to_tsvector('The quick brown fox jumped over the lazy dog')
2 @@ to_tsquery('foxhound');
```

#### Результат:

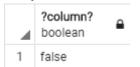


#### Отвечая на вопросы:

- 1) Символ @ @ является оператором соответствия для полнотекстового поиска.
- 2) Во втором запросе, в отличие от третьего используется множественное число слова fox, поэтому, при выполнении функции to\_tsvector, первый словарь, который *распознал* фрагмент, выдаёт одну или несколько представляющих его *лексем*. Поэтому foxes превращается в fox, так как один из словарей понимает, что слово foxes это слово fox во множественном числе. Слово foxhound не является множественным числом, либо однокоренным со словом fox. Поэтому результатом третьего запроса было false.
- 3) Выполним запрос

```
1 SELECT to_tsvector('Russian', 'Съешь ещё этих мягких французских булок, да выпей чаю.')
2 @@ to_tsquery('Russian','булка');
```

# Результатом выполнения является следующее:



Для ответа на вопрос выполним следующий запрос:

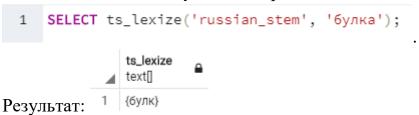
```
1 SELECT ts_lexize('russian_stem', 'булок');
```

# Результатом которого будет:



Функция ts\_lexize возвращает массив лексем, если входной фрагмент известен словарю, либо пустой массив, если этот фрагмент считается в словаре стоп-словом.

# Так же выполним следующий запрос:

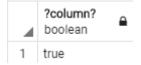


Соответственно, при выполнении исходного запроса to\_tsvector распознает слово булок как лексему 'булок', а слово булка имеет лексему 'булк', именно из-за разницы лексем слово булка и не было найдено в запросе.

4) Выполним следующие запросы:

```
1 SELECT to_tsvector('Russian', 'Съешь ещё этих мягких французских пирожков, да выпей чаю.')
       @@ to_tsquery('Russian','пирожки');
```

#### Результатом является следующее:



#### Выполним следующий запрос:

```
1 SELECT to_tsvector('Russian', 'Съешь ещё этих мягких французских пирожков, да выпей чаю.')
       @@ to_tsquery('Russian','пирожок');
```

# Результатом будет следующее:



При выполнении аналогичного запроса представленного в пункте 3, выясняется, что лексемой слова пирожки является "пирожк", лексемой пирожок является "пирожок", лексемой же слова пирожков является так же "пирожк". Именно поэтому результатом первого запроса является true, а второго false.

#### 3 Задание

# 1) Выполним следующие запросы:

```
1 SELECT to_tsvector('The quick brown fox jumped over the lazy dog')
2
       @@ to_tsquery('fox & dog');
```

# Результат:



#### Следующий запрос:

```
1 SELECT to_tsvector('The quick brown fox jumped over the lazy dog')
2
       @@ to_tsquery('fox | rat');
```

## Результат:



#### Следующий запрос:

```
SELECT to_tsvector('The quick brown fox jumped over the lazy dog')
       @@ to_tsquery('!clown');
2
```

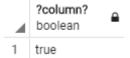
## Результат:

```
?column?
boolean □
```

# Последний запрос:

```
1 SELECT to_tsvector('The quick brown fox jumped over the lazy dog')
2 @@ to_tsquery('fox & (dog | rat) & !mice');
```

# Результат:



#### Выполним аналогичные запросы для предложения на русском языке

```
1 SELECT to_tsvector('Russian', 'В центре большого города стоит деревянный дом')
2 @@ to_tsquery('Russian','город & дом');
```

#### Результат:



# Следующий запрос:

```
SELECT to_tsvector('Russian', 'В центре большого города стоит деревянный дом')

@@ to_tsquery('Russian','дом | деревня');
```

# Результат:



#### Следующий результат:

```
1 SELECT to_tsvector('Russian', 'В центре большого города стоит деревянный дом')
2 @@ to_tsquery('Russian','!квартира');
```

#### Результат:



#### Последний запрос:

```
SELECT to_tsvector('Russian', 'В центре большого города стоит деревянный дом')

@@ to_tsquery('Russian','город & (дом | квартира) & !деревня');
```

#### Результат:



2) Английский язык является языком по умолчанию, поэтому его не нужно указывать в первом аргументе. Если не указывать анализатор, то возьмётся язык по умолчанию.

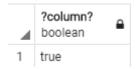
#### 4 Задание

## Выполним следующий запрос:

```
SELECT to_tsvector('Russian', 'Съешь ещё этих мягких французских булок, да выпей чаю.')

@@ to_tsquery('Russian','мягких<2>булок');
```

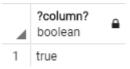
#### Результатом выполнения запросов является:



- 1) Оператор <-> является оператором поиска фраз <> (ПРЕДШЕСТВУЕТ). В нашем случае представлена вариация оператора ПРЕДШЕСТВУЕТ вида <N>, где N целочисленная константа, задающая расстояние между двумя искомыми лексемами. Соответственно <2> обозначает расстояние между лексемами 'мягких' и 'булок'.
- 2) Выполним запрос для поиска фразы:

```
1 SELECT to_tsvector('Russian', 'Съешь ещё этих мягких французских булок, да выпей чаю.')
2 @@ to_tsquery('Russian','Съешь<->ещё');
```

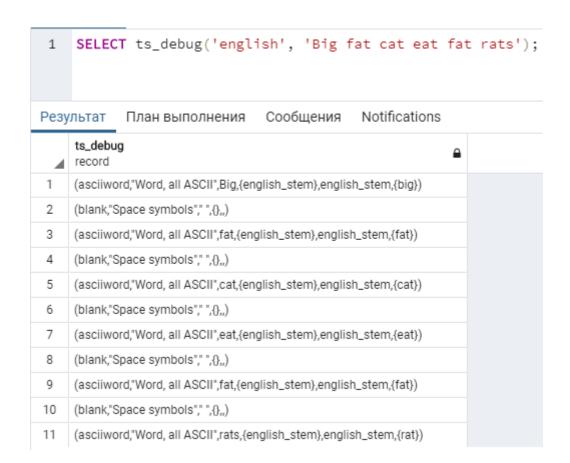
# Результат:



3) Функция phraseto\_tsquery преобразует неформатированный текст запроса в значение tsquery. Она вставляет между оставшимися словами оператор <->. Стоп-слова не отбрасываются, а подсчитываются, и вместо операторов <-> используются операторы <*N*> с подсчитанным числом.

#### Задание 5

1) Функция ts\_debug - выводит информацию обо всех фрагментах данного документа, которые были выданы анализатором и обработаны настроенными словарями.



2) Функция ts\_headline принимает документ вместе с запросом и возвращает выдержку из документа, в которой выделяются слова из запроса.

