НЕТОЛОГИЯ
групп

ЗАНЯТИЕ 1.5

Работа с PostgreSQL

Ирина Хомутова

Software developer in Eltex LTD

irinavkhomutova@icloud.com

Определение полнотекстового поиска

Документ - это единица обработки в системе полнотекстового поиска

Текстовый поиск - операция анализа набора документов с текстом на естественном языке

Типы данных PostgreSQL

tsvector

tsquery

tsvector

Представляет документ в виде оптимизированном для текстового поиска

Нормализованная строка по которой будет производиться поиск

to_tsvector([конфигурация regconfig,] документ text) returns tsvector

- разбирает текстовый документ на фрагменты,
- сводит фрагменты к лексемам и возвращает значение tsvector, в котором перечисляются лексемы и их позиции в документе

Нормализованная строка:

[ ]

SELECT to_tsvector('I find the system really useful');

'find':2 'realli':5 'system':4 'use':6

Для русского языка:

[ ]

SELECT to_tsvector('russian', 'жили - были дед да баба, и была у них курица Ряба');

'баб':5 'дед':3 'жил':1 'куриц':10 'ряб':11

tsquery

Для представления запросов поиска

Содержит искомые лексемы , объединяемые логическими операторами

Приоритеты операторов в порядке убывания:

! - не

<N>- предшествует

& - и

| - или

Для группировки операторов могут использоваться скобки ()

to_tsquery, plainto_tsquery и phraseto_tsquery - функции приведения запроса к типу данных tsquery

to_tsquery

to_tsquery([конфигурация regconfig,] текст_запроса text) returns tsquery

создаёт значение tsquery из текста_запроса, который может состоять из простых фрагментов, разделённых логическими операторами

[ ]

```
SELECT to_tsquery('russian', 'жили | были & !дед & баба');
'жил' | !'дед' & 'баб'

SELECT to_tsquery('''дед баба'' & !ряба');
--Если убрать апострофы, to_tsquery не примет фрагменты, не разделённые операторами AND и OR, и выдаст синтаксическую ошибку.
'дед' & 'баб' & !'ряб'
```

plainto_tsquery

plainto_tsquery([конфигурация regconfig,] текст_запроса text) returns tsquery

plainto_tsquery преобразует неформатированный текст_запроса в значение tsquery.

Текст разбирается и нормализуется подобно тому, как это делает to_tsvector, а затем между оставшимися словами вставляются логические операторы & (AND).

[ ]

SELECT plainto_tsquery('russian', 'жили - были дед да баба, и была у них курица ряба');

'жил' & 'дед' & 'баб' & 'куриц' & 'ряб'

SELECT plainto_tsquery('russian', 'жили | были дед | баба, и была у них курица ряба');

'жил' & 'дед' & 'баб' & 'куриц' & 'ряб'

phraseto_tsquery

работает подобно plainto_tsquery, за исключением того, что она использует оператор <->

полезно при поиске точных последовательностей лексем

[ ]

```
SELECT phraseto_tsquery('russian', 'жили - были дед да баба, и была у них курица ряба');
'жил' <2> 'дед' <2> 'баб' <5> 'куриц' <-> 'ряб'

SELECT phraseto_tsquery('russian', 'жили | были дед | баба, и была у них курица ряба');
'жил' <2> 'дед' <-> 'баб' <5> 'куриц' <-> 'ряб'
```

Поиск в таблице

Поиск должен найти документы tsvector соответствующие запросу tsquery.

Для сопоставления используется оператор @@.

Для таблицы news следующего формата:

id | title | content ... | ... | ...

поисковый запрос по колонкам title и content будет таким:

[ ]

```
SELECT * FROM news WHERE to_tsvector(title) || to_tsvector(content)
@@ plainto_tsquery('user search text');

--Оператор || используется для конкатенации tsvector.
```

Ранжирование результатов поиска

Оценка релевантности документа относительно запроса происходит с учетом весов элементов tsvector.

Функция setweight

Значения весов: A B C D

Установим важность слов в заголовке больше чем в теле документа:

[ ]

```
SELECT * FROM news WHERE setweight(to_tsvector(title), 'A') ||
setweight(to_tsvector(content), 'B') @@ plainto_tsquery('user search text')
ORDER BY ts_rank(setweight(to_tsvector(title), 'A') ||
setweight(to_tsvector(content), 'B'), plainto_tsquery('user search text'))
DESC;
-- Функция ts_rank ранжирует результаты по частоте найденных лексем.
```

Выделение результата

Выделять часть документа, показывая фрагменты документа с отмеченными искомыми словами.

ts_headline([конфигурация regconfig,] документ text, запрос tsquery [, параметры text]) returns text

ts_headline принимает документ вместе с запросом и возвращает выдержку из документа, в которой выделяются слова из запроса

параметры - строка из списка разделённых запятыми пар параметр=значение

Параметры по умолчанию:

StartSel=, StopSel=,

MaxWords=35, MinWords=15,

ShortWord=3,

HighlightAll=FALSE,

MaxFragments=0,

FragmentDelimiter=" ... "

Работает с оригинальным документом, а не с его сжатым представлением tsvector.
Типичная ошибка — вызывать ts_headline для всех документов

```
SELECT ts_headline('russian',
  'Жили-были дед да баба.
   И была у них курочка Ряба. Снесла курочка яичко.
   Яичко не простое - золотое. Дед бил-бил, не разбил.
   Баба била-била, не разбила. А мышка бежала, хвостиком махнула, яичко упало и разбилось.
   Дед и баба плачут, а курочка кудахчет:
   "Не плачь дед, не плачь баба. Я снесу вам яичко другое, не золотое, а простое".',
  phraseto_tsquery('russian','дед да баба'));
```

Дед и баба плачут, а курочка кудахчет:
"Не плачь дед, не плачь баба. Я снесу
SELECT ts_headline('russian',
'Жили-были дед да баба.
И была у них курочка Ряба. Снесла курочка яичко.
Яичко не простое - золотое. Дед бил-бил, не разбил.
Баба била-била, не разбила. А мышка бежала, хвостиком махнула, яичко упало и разбилось.
Дед и баба плачут, а курочка кудахчет:
"Не плачь дед, не плачь баба. Я снесу вам яичко другое, не золотое, а простое".',
phraseto_tsquery('russian','дед да баба'),
'HighlightAll=TRUE');

Жили-были дед да баба.

И была у них курочка Ряба. Снесла курочка яичко.

Яичко не простое - золотое. Дед бил-бил, не разбил.

Баба била-била, не разбила. А мышка бежала, хвостиком махнула, яичко упало и разбилось.

Дед и баба плачут, а курочка кудахчет:

"Не плачь дед, не плачь баба. Я снесу вам яичко другое, не золотое, а простое".

Создание индексов

GIN (Generalized Inverted Index)

Содержит записи всех ключей (лексем) со списком мест их вхождений

Использует бинарное дерево, поэтому он слабо зависит от количества ключей и хорошо масштабируется

Не используйте индекс GIN для документов которые постоянно изменяются, так как изменения приводят к большому количеству обновлений индекса

[ ]

```
CREATE INDEX search_index news USING
GIN(setweight(to_tsvector('title'), 'A')
|| setweight(to_tsvector(content), 'B'));
```

Функции

Объекты базы данных, которые используются для автоматизации и упрощения расчетов
PL/pgSQL – язык программирования, который используется в СУБД PostgreSQL для написания функций, триггеров и других управляющих конструкций.

В функциях в СУБД PostgreSQL можно использовать все операторы SQL такие как: INSERT, DELETE, UPDATE и другие.

Создание функций на PL/pgSQL

Общий синтаксис написания функции в PL/pgSQL:

CREATE OR REPLACE FUNCTION название функции (типы передаваемых данных через запятую)

RETURNS тип возвращаемого значения AS

$BODY$

DECLARE

Объявление переменных

BEGIN

Тело программы

RETURN возвращаемый результат;

END;

$BODY$

LANGUAGE язык, на котором написана функция (например, SQL или plpgsql) VOLATILE

```
CREATE OR REPLACE FUNCTION public.cnt_jid(character varying)
    returns bigint AS
  $BODY$
    SELECT count(*) as cnt
    FROM public.supply
    where jid = $1;
  $BODY$
  LANGUAGE 'sql' VOLATILE;

 SELECT *, cnt_jid(id) AS numjid FROM public.job
```

Изменение таблиц

ALTER TABLE название_таблицы
{
ADD названи*столбца тип*данных*столбца [ограничения*столбца] |
DROP COLUMN название_столбца |
ALTER COLUMN названи*столбца параметры*столбца |
ADD [CONSTRAINT] определение_ограничения |
DROP [CONSTRAINT] имя_ограничения
}

Возврат данных из хранимых процедур

Для возврата простых типов в заголовке хранимой процедуры достаточно указать тип данных, который будет возвращаться, а в теле самой процедуры использовать оператор RETURN

[ ]

```
CREATE OR REPLACE FUNCTION return_int() RETURNS int AS
$$
BEGIN
 RETURN 1;
END
$$ LANGUAGE plpgsql;

SELECT * FROM return_int();
return_int
------------
          1
```

Возврат набора данных

SETOF - в заголовке функции перед типом возвращаемых данных

RETURN NEXT - в теле функции для возврата каждой строки

[ ]

```
CREATE OR REPLACE FUNCTION return_setof_int() RETURNS SETOF int AS
$$
BEGIN
 RETURN NEXT 1;
 RETURN NEXT 2;
 RETURN NEXT 3;
 RETURN; -- Необязательный
END
$$ LANGUAGE plpgsql;
```

```
SELECT * FROM return_setof_int();
return_setof_int
------------------
                1
                2
                3
(3 rows)
```

Использование OUT-параметров

Режим аргумента:
IN (входной),
OUT (выходной),
INOUT (входной и выходной)
VARIADIC (переменный)
OUT и INOUT нельзя использовать с предложением RETURNS TABLE

```
CREATE OR REPLACE FUNCTION return_out_int(OUT result1 int, OUT result2 int) AS
$$
BEGIN
  result1 := 1;
  RETURN;
END
$$ LANGUAGE plpgsql;

SELECT * FROM return_out_int();
 result1 | result2
---------+---------
       1 |
(1 row)
```

Использование выражения TABLE

Использование похоже на OUT-параметры, которые просто вынесены в отдельно стоящую конструкцию языка

Отличает возможность возврата набора данных без использования ключевого слова SETOF в заголовке функции

```
CREATE OR REPLACE FUNCTION return_table() RETURNS table(id int, name varchar) AS
$$
BEGIN
 id := 1;
 name := 'name';
 RETURN NEXT;
 RETURN NEXT;
END
$$ LANGUAGE plpgsql;
```

```
SELECT * FROM return_table();
id | name
----+------
 1 | name
 1 | name
(2 rows)
```

Возврат табличных типов данных

табличные типы данных определяются структурой таблиц

[ ]

```
CREATE OR REPLACE FUNCTION return_products() RETURNS SETOF
products AS
$$
 SELECT * FROM products
$$ LANGUAGE sql;


SELECT * FROM  return_products();
```

Использование курсоров

Курсор в SQL –временная выборка записей в процессе выполнения функции, над которой могут выполняться необходимые действия

Данная выборка является указателем на область памяти.

Курсоры используют если:

- необходима итеративная обработка
- полная выборка занимает слишком много памяти
- нужна не вся выборка, но размер заранее неизвестен
- способ отдать управление выборкой клиенту

Курсор ресурсоемкое решение - поэтому использовать их нужно только в крайнем случае.

Курсоры в функциях на PL/pgSQL

Общий синтаксис курсора в функции

```
CREATE OR REPLACE FUNCTION название функции(типы переменных)
    RETURNS тип возвращаемого значения  AS
  $BODY$
   DECLARE
    объявление переменных
    объявление курсора
   BEGIN
     открытие курсора
    перебор данных и операции над ними
     закрытие курсора
     RETURN возвращение значения;
   END;
   $BODY$
  LANGUAGE 'plpgsql' VOLATILE
```

Возврат простого результата, вычисленного с помощью курсора

ALIAS

новое*имя ALIAS FOR старое*имя;

Объявление псевдонимов для переменных

[ ]

```
CREATE OR REPLACE FUNCTION public.sum_with_cursor(varchar, varchar)
    RETURNS numeric AS
  $BODY$
  DECLARE
 shipper_id ALIAS FOR $1;
 product_id ALIAS FOR $2;
 --объявляем курсор
```

```
crs_sum CURSOR FOR
    SELECT pid, num
    FROM public.supply
    where supply.sid = shipper_id;
 --объявляем нужные нам переменные
   pid_cur varchar;
      shipper_num integer;
      shipper_sum integer;
      BEGIN
       shipper_sum:=0;
       OPEN crs_sum;--открываем курсор
       LOOP --начинаем цикл по курсору
       --извлекаем данные из строки и записываем их в переменные
```

```
 FETCH crs_sum INTO pid_cur, shipper_num;
   --если такого периода и не возникнет, то мы выходим
   IF NOT FOUND THEN EXIT;
   END IF;
   IF pid_cur = product_id then
      shipper_sum = shipper_sum  + shipper_num;
   END IF;
    END LOOP;--заканчиваем цикл по курсору
    CLOSE crs_sum; --закрываем курсор
   RETURN shipper_sum;--возвращаем результат
  END;
  $BODY$
    LANGUAGE 'plpgsql' volatile;
```

```
SELECT *, sum(num) over (partition by sid, pid), public.sum_with_cursor(sid, pid)
from supply;
```

Возврат табличного результата с использованием курсора

record

Переменные похожи на переменные строкового типа, но они не имеют предопределённой структуры.

Структура может меняться каждый раз при присвоении значения

Не подлинный тип данных, а только лишь заполнитель.

```
CREATE OR REPLACE FUNCTION public.count_percent()
    RETURNS TABLE(sid varchar, pid varchar, jid varchar, percents varchar)
AS
  $BODY$
   declare
    rec RECORD;
    cur_percents CURSOR for
       SELECT public.supply.sid,
            public.supply.pid,
            public.supply.jid,
                                                    round(100                    *
public.supply.num/public.sum_with_cursor(public.supply.sid,
public.supply.pid), 3) as percents
       FROM public.supply ;
```

```
BEGIN
   FOR rec IN cur_percents
      LOOP
      sid = rec.sid;
      pid =rec.pid;
      jid = rec.jid;
      percents = cast(rec.percents as varchar)|| '%';
      RETURN next;
   END LOOP;
   END;
  $BODY$
    LANGUAGE 'plpgsql' VOLATILE

SELECT * from public.count_percent();
```

Возврат курсора

Курсоры существуют только в рамках одной транзакции

Для получения данных необходимо явным образом создавать транзакцию

Указание CONSTANT предотвращает изменение значения переменной после инициализации

```
CREATE OR REPLACE FUNCTION return_cursor_job() RETURNS refcursor
AS
$$
DECLARE
  _result CONSTANT refcursor := '_result';
BEGIN
  OPEN _result FOR SELECT * FROM public.job;
  RETURN _result;
END
$$ LANGUAGE plpgsql;
```

```
BEGIN;
SELECT * FROM return_cursor_job();
FETCH ALL FROM _result;
COMMIT;
SELECT * FROM return_cursor_job();
FETCH ALL FROM _result;
```

ИТОГИ ЗАНЯТИЯ

Мы сегодня научились:

1. Составлять запросы для полнотекстового поиска
2. Писать хранимые функции на языке PL/pgSQ
3. Использовать курсоры внутри функций

ВОПРОСЫ

Полезные материалы

Более сложные типы данных, переменных и результата, а также создание триггеров можно изучить здесь:

- массивы: https://postgrespro.ru/docs/postgrespro/9.6/arrays

- работа с json: https://postgrespro.ru/docs/postgrespro/9.5/functions-json

- работа с составными типами: https://postgrespro.ru/docs/postgresql/11/rowtypes

- триггеры: https://postgrespro.ru/docs/postgresql/11/plpgsql-trigger

Домашнее задание

Домашнее задание

1. создайте связанную с film таблицу film_annotation(film_id - первичный ключ ссылается на film, annotation - текст аннотации) .

2. заполните таблицу 6 аннотациями к фильмам (произвольный текст с фразами

"- Кто свидетель?

- Я! А что случилось?",

"- Какое горе! Принцесса Диана была так молода и красива!

- Будь она старой и страшной — было бы не так плохо?",

"Есть такая история про Париж и людей, умиравших от голода во время войны. Они все сидели вокруг стола, и в тишине кто-то сказал: «Ангел пролетает», а кто-то другой сказал: «Давайте его съедим»")

3. Напишите функцию, возвращающую таблицу документов, содержащих различные фразы (полнотекстовый поиск).

4. Протестируйте функцию на исходных фразах

НЕТОЛОГИЯ
групп

Спасибо за внимание!

Ирина Хомутова

irinavkhomutova@icloud.com