МОСКОВСКИЙ ГОРОДСКОЙ УНИВЕРСИТЕТ
МГПУ

Аналитика с использованием сложных типов данных

Array Data Types

МОСКОВСКИЙ ГОРОДСКОЙ УНИВЕРСИТЕТ МГПУ

Массивы Postgres позволяют хранить несколько значений в поле таблицы.

Например, рассмотрим следующую первую запись в таблице клиентов:

```
customer_id      | 1
title            | NULL
first_name       | Arlena
last_name        | Riveles
suffix           | NULL
email            | ariveles0@stumbleupon.com
gender           | F
ip_address       | 98.36.172.246
phone            | NULL
street_address   | NULL
city             | NULL
state            | NULL
postal_code      | NULL
latitude         | NULL
longitude        | NULL
date_added       | 2017-04-23 00:00:00
```

Array Data Types

МОСКОВСКИЙ ГОРОДСКОЙ УНИВЕРСИТЕТ МГПУ

Каждое поле содержит ровно одно значение (значение NULL по-прежнему является значением); однако есть некоторые атрибуты, которые могут содержать несколько значений с неопределенной длиной.

Например, представьте, что хотим иметь поле **Purchased_products**. Это может содержать ноль или более значений в поле.

Например, представьте, что клиент купил скутеры **Lemon** и **Bat Limited Edition**; мы можем представить это следующим образом

```
customer_id      | 1
title            | NULL
first_name       | Arlena
last_name        | Riveles
suffix           | NULL
email            | ariveles0@stumbleupon.com
gender           | F
ip_address       | 98.36.172.246
phone            | NULL
street_address   | NULL
city             | NULL
state            | NULL
postal_code      | NULL
latitude         | NULL
longitude        | NULL
date_added       | 2017-04-23 00:00:00
```

```
purchased_products | {Lemon,"Bat Limited Edition"}
```

Array Data Types

МОСКОВСКИЙ ГОРОДСКОЙ УНИВЕРСИТЕТ МГПУ

Создать массив с помощью следующей команды

```
SELECT ARRAY['Lemon', 'Bat Limited Edition'] AS example_purchased_products;
```

Можем создавать массивы, используя агрегатную функцию **ARRAY_AGG**. Например, следующий запрос объединяет все автомобили для каждого типа продукта.

```
SELECT product_type, ARRAY_AGG(DISTINCT model) AS models FROM products
GROUP BY 1;
```

```
product_type|models                                                                                |
------------+--------------------------------------------------------------------------------------+
automobile  |{"Model Chi","Model Epsilon","Model Gamma","Model Sigma"}                             |
scooter     |{Bat,"Bat Limited Edition",Blade,Lemon,"Lemon Limited Edition","Lemon Zester"}|
```

Array Data Types

МОСКОВСКИЙ ГОРОДСКОЙ УНИВЕРСИТЕТ МГПУ

Можно отменить эту операцию, используя функцию **UNNEST**, которая создает одну строку для каждого значения в массиве.\

```
SELECT UNNEST(ARRAY[123, 456, 789]) AS example_ids;
```

Можно создать массив, разделив строковое значение с помощью функции **STRING_TO_ARRAY**, например:

```
SELECT STRING_TO_ARRAY('hello there how are you?', ' ');
```

Точно так же можно запустить обратную операцию и объединить массив строк в одну строку:

```
SELECT ARRAY_TO_STRING(ARRAY['Lemon', 'Bat Limited Edition'], ', ') AS
example_purchased_products;
```

Array Data Types

Есть и другие функции, позволяющие взаимодействовать с массивами. Вот несколько примеров дополнительных функций массива, которые предоставляет **Postgres**.

МОСКОВСКИЙ ГОРОДСКОЙ УНИВЕРСИТЕТ

Операция	Пример функции Postgres	Пример вывода
Объединить два массива	**SELECT array_cat(ARRAY**[1,2,3], **ARRAY**[4,5]); или ARRAY[1, 2] || ARRAY[3, 4]	{1,2,3,4}
Добавить значение в массив	**SELECT array_append(array**[1, 2], 3); или ARRAY[1, 2] || 3	{1,2,3}
Проверить, содержится ли значение в массиве	**SELECT** 3 = **ANY(ARRAY**[1, 2]);	false
Проверка, не перекрываются ли два массива	**SELECT ARRAY**[1, 2, 3] && **ARRAY**[3, 4];	true
Проверить, содержит ли массив другой массив	**SELECT ARRAY**[1, 2, 3] @> **ARRAY**[2, 1];	true

МОСКОВСКИЙ ГОРОДСКОЙ УНИВЕРСИТЕТ
МГПУ

Использование типов данных **JSON** в Postgres

Тип данных **JSON**

МОСКОВСКИЙ ГОРОДСКОЙ УНИВЕРСИТЕТ МГПУ

Массивы могут быть полезны для хранения списка значений в одном поле, иногда наши структуры данных могут быть сложными.
Однако, может понадобиться **хранить несколько значений разных типов в одном поле**, и необходимо, чтобы данные помечались метками, а не сохранялись последовательно.

Тип данных **JSON**

МОСКОВСКИЙ ГОРОДСКОЙ УНИВЕРСИТЕТ МГПУ

JavaScript Object Notation (JSON) — это открытый стандартный текстовый формат для хранения данных различной сложности.

Его можно использовать для обозначения чего угодно. Подобно тому, как таблица базы данных имеет имена столбцов, данные **JSON** имеют ключи.

Можно легко представить запись из базы данных клиентов, используя **JSON**, сохраняя имена столбцов в качестве ключей и значения строк в качестве значений.

Тип данных JSON

МОСКОВСКИЙ ГОРОДСКОЙ УНИВЕРСИТЕТ МГПУ

Функция row_to_json преобразует строки в JSON:

```
SELECT row_to_json(c) FROM customers c limit 1;
```


```
{"customer_id":1,"title":null,"first_name":"Arlena","la
st_name":"Riveles","suffix":null,"email":"ariveles0@stu
mbleupon.com","gender":"F","ip_address":"98.36.172.246"
,"phone":null,"street_address":null,"city":null,"state"
:null,"postal_code":null,"latitude":null,"longitude":nu
ll,"date_added":"2017-04-23T00:00:00"}
```

Тип данных **JSON**

Это немного сложно читать, можем добавить флаг **pretty_bool** в функцию **row_to_json**, чтобы сгенерировать удобочитаемую версию:


```
{"customer_id":1,
"title":null,
"first_name":"Arlena",
"last_name":"Riveles",
"suffix":null,
"email":"ariveles0@stumbleupon.com",
"gender":"F",
"ip_address":"98.36.172.246",
"phone":null,
"street_address":null,
"city":null,
"state":null,
"postal_code":null,
"latitude":null,
"longitude":null,
"date_added":"2017-04-23T00:00:00"}
```


```
SELECT row_to_json(c, TRUE) FROM customers c limit 1;
```

Тип данных **JSON**

МОСКОВСКИЙ ГОРОДСКОЙ УНИВЕРСИТЕТ МГПУ

Структура **JSON** содержит ключи и значения.

В примере ключами являются просто имена столбцов, а значения берутся из значений строк.

Значения **JSON** могут быть:

- **числовыми значениями** (целыми или с плавающей запятой),
- **логическими значениями** (true или false),
- **текстовыми значениями** (заключенными в двойные кавычки)
- **null**.

JSON также может включать вложенные структуры данных.

JSONB: предварительно обработанный **JSON**

МОСКОВСКИЙ ГОРОДСКОЙ УНИВЕРСИТЕТ МГПУ

В то время как текстовое поле **JSON** необходимо анализировать каждый раз, когда на него ссылаются, значение **JSONB** предварительно анализируется, а данные хранятся в декомпозированном двоичном формате.

Это требует, чтобы первоначальный ввод был предварительно проанализирован, а преимущество заключается в значительном повышении производительности при запросе ключей или значений в этом поле.

Это связано с тем, что ключи и значения не нужно анализировать — **они уже извлечены и сохранены в доступном двоичном формате**.

JSONB: предварительно обработанный **JSON**

МОСКОВСКИЙ ГОРОДСКОЙ УНИВЕРСИТЕТ МГПУ

JSONB отличается от **JSON**:

- не может быть более одного ключа с одним и тем же именем,
- порядок ключей не сохраняется,
- не сохраняются семантически несущественные детали, например пробелы.

Доступ к данным из поля JSON или JSONB

МОСКОВСКИЙ ГОРОДСКОЙ УНИВЕРСИТЕТ МГПУ

Ключи JSON можно использовать для доступа к связанному значению с помощью оператора ->. Например:

```
SELECT
'{
      "a": 1,
      "b": 2,
      "c": 3
}'::JSON -> 'b' AS data;
```

Доступ к данным из поля **JSON** или **JSONB**

МОСКОВСКИЙ ГОРОДСКОЙ УНИВЕРСИТЕТ МГПУ

Postgres также позволяет более сложным операциям обращаться к вложенному **JSON** с помощью оператора **#>**. Например:

```
SELECT
'{
      "a": 1,
      "b": [
            {"d": 4},
            {"d": 6},
            {"d": 4}
            ],
      "c": 3
}'::JSON #> ARRAY['b', '1', 'd'] AS data;
```

Доступ к данным из поля JSON или JSONB

МОСКОВСКИЙ ГОРОДСКОЙ УНИВЕРСИТЕТ МГПУ

Справа от оператора #> текстовый массив определяет путь для доступа к нужному значению. В этом случае мы выбираем значение «b», которое представляет собой список вложенных объектов JSON. Затем мы выбираем элемент в списке, обозначенный «1», который является вторым элементом, потому что индексы массива начинаются с 0. Наконец, мы выбираем значение, связанное с ключом «d», и выход равен 6.

```
SELECT
'{
        "a": 1,
        "b": [
                {"d": 4},
                {"d": 6},
                {"d": 4}
                ],
        "c": 3
}'::JSON #> ARRAY['b', '1', 'd'] AS data;
```

Доступ к данным из поля JSON или JSONB

МОСКОВСКИЙ ГОРОДСКОЙ УНИВЕРСИТЕТ МГПУ

Эти функции работают с полями **JSON** или **JSONB** (имейте в виду, что они будут работать намного быстрее с полями **JSONB**).

Однако **JSONB** также обеспечивает дополнительную функциональность. Например, отфильтровать строки на основе пары **key-value**. Вы можете использовать оператор **@>**, который проверяет, содержит ли объект **JSONB** слева значение ключа справа. Например:

```
SELECT JSONB_PRETTY(customer_json) FROM customer_sales
WHERE customer_json @> '{"customer_id":20}'::JSONB;
```

Доступ к данным из поля JSON или JSONB

МОСКОВСКИЙ ГОРОДСКОЙ УНИВЕРСИТЕТ МГПУ

```
SELECT JSONB_PRETTY(customer_json) FROM customer_sales
WHERE customer_json @> '{"customer_id":20}'::JSONB;
```

```
{
    "email": "ihughillj@nationalgeographic.com",
    "phone": null,
    "sales": [
    ],
    "last_name": "Hughill",
    "date_added": "2012-08-08T00:00:00",
    "first_name": "Itch",
    "customer_id": 20
}
```

Доступ к данным из поля JSON или JSONB

МОСКОВСКИЙ ГОРОДСКОЙ УНИВЕРСИТЕТ МГПУ

Можно выбрать только ключи из поля **JSONB** и разделить их на несколько строк с помощью функции **JSONB_OBJECT_KEYS**. Используя эту функцию, можем извлечь значение, связанное с каждым ключом, из исходного поля **JSONB**, используя оператор **->**.

```
SELECT
      JSONB_OBJECT_KEYS(customer_json) AS keys,
      customer_json -> JSONB_OBJECT_KEYS(customer_json) AS values
FROM customer_sales
WHERE customer_json @> '{"customer_id":20}'::JSONB;
```

Доступ к данным из поля JSON или JSONB

МОСКОВСКИЙ ГОРОДСКОЙ УНИВЕРСИТЕТ МГПУ

```
SELECT
      JSONB_OBJECT_KEYS(customer_json) AS keys,
      customer_json -> JSONB_OBJECT_KEYS(customer_json) AS values
FROM customer_sales
WHERE customer_json @> '{"customer_id":20}'::JSONB;
```

```
keys        |values                                  |
------------+----------------------------------------+
email       |"ihughillj@nationalgeographic.com"|
phone       |null                                    |
sales       |[]                                      |
last_name   |"Hughill"                               |
date_added  |"2012-08-08T00:00:00"                   |
first_name  |"Itch"                                  |
customer_id|20                                       |
```

Создание и изменение данных в поле JSONB

МОСКОВСКИЙ ГОРОДСКОЙ УНИВЕРСИТЕТ МГПУ

Добавление новой пары ключ-значение «c»: 2

```
SELECT jsonb_insert('{"a":1,"b":"foo"}', ARRAY['c'], '2');
```

```
jsonb_insert                |
----------------------------+
{"a": 1, "b": "foo", "c": 2}|
```

Вставка значения во вложенный объект JSON

```
SELECT jsonb_insert('{"a":1,"b":"foo", "c":[1, 2, 3, 4]}', ARRAY['c', '1'], '10');
```

```
jsonb_insert                            |
----------------------------------------+
{"a": 1, "b": "foo", "c": [1, 10, 2, 3, 4]}|
```

Practice. Поиск в **JSONB**

МОСКОВСКИЙ ГОРОДСКОЙ УНИВЕРСИТЕТ МГПУ

Идентифицировать всех клиентов, купивших самокат **Blade**; используем данные, хранящиеся как **JSNOB**.

Practice 06-2. Поиск в JSONB

МОСКОВСКИЙ ГОРОДСКОЙ УНИВЕРСИТЕТ МГПУ

Шаги для выполнения запроса PostgreSQL:

1. выделить каждую продажу в отдельную строку с помощью функции **JSONB_ARRAY_ELEMENTS**;

```
CREATE TEMP TABLE customer_sales_single_sale_json AS (
	SELECT
		customer_json,
		JSONB_ARRAY_ELEMENTS(customer_json -> 'sales') AS sale_json
	FROM customer_sales LIMIT 10
);
```

Practice. Поиск в JSONB

МОСКОВСКИЙ ГОРОДСКОЙ УНИВЕРСИТЕТ МГПУ

Шаги для выполнения запроса PostgreSQL:

2. отфильтровать этот вывод и получить записи, где **product_name** — **«Blade»**:

```
SELECT DISTINCT customer_json
FROM customer_sales_single_sale_json
WHERE sale_json ->> 'product_name' = 'Blade';
```


```
{"email": "nespinaye@51.la", "phone": "818-658-6748", "sales":
[{"product_id": 5, "product_name": "Blade", "sales_amount": 559.992,
"sales_transaction_date": "2014-07-19T06:33:44"}], "last_name": "Espinay",
"date_added": "2014-07-05T00:00:00", "first_name": "Nichols",
"customer_id": 15}
```

Practice. Поиск в JSONB

МОСКОВСКИЙ ГОРОДСКОЙ УНИВЕРСИТЕТ МГПУ

Шаги для выполнения запроса PostgreSQL:

2. отфильтровать этот вывод и получить записи, где **product_name** — **«Blade»**:

```
SELECT DISTINCT JSONB_PRETTY(customer_json)
FROM customer_sales_single_sale_json
WHERE sale_json ->> 'product_name' = 'Blade' ;
```

Practice. Поиск в JSONB

МОСКОВСКИЙ ГОРОДСКОЙ УНИВЕРСИТЕТ МГПУ

Шаги для выполнения запроса PostgreSQL:

2. отфильтровать этот вывод и получить записи, где **product_name** — **«Blade»**:


```
{
        "email": "nespinaye@51.la",
        "phone": "818-658-6748",
        "sales": [
                {
                        "product_id": 5,
                        "product_name": "Blade",
                        "sales_amount": 559.992,
                        "sales_transaction_date": "2014-07-19T06:33:44"
                }
        ],
        "last_name": "Espinay",
        "date_added": "2014-07-05T00:00:00",
        "first_name": "Nichols",
        "customer_id": 15
}
```

МОСКОВСКИЙ ГОРОДСКОЙ УНИВЕРСИТЕТ
МГПУ

Текстовая аналитика с использованием Postgres

Текстовая аналитика с использованием **Postgres**

МОСКОВСКИЙ ГОРОДСКОЙ УНИВЕРСИТЕТ МГПУ

Часто текст содержит ценную информацию —представим себе продавца, который ведет заметки о потенциальных клиентах:

«Очень многообещающее взаимодействие, клиент собирается совершить покупку завтра»

содержит ценные данные, как и это примечание:

«Клиент не заинтересован. больше не нуждаются в продукте».

Текстовая аналитика с использованием **Postgres**

МОСКОВСКИЙ ГОРОДСКОЙ УНИВЕРСИТЕТ МГПУ

Ключевые слова в этих заявлениях, такие как «обещаю», «покупка», «завтра», «не заинтересован» и «нет», могут быть извлечены с помощью правильных методов, чтобы попытаться определить наиболее перспективных клиентов в автоматическом режиме.

«Очень многообещающее взаимодействие, клиент собирается совершить покупку завтра»

содержит ценные данные, как и это примечание:

«Клиент не заинтересован. больше не нуждаются в продукте».

Токенизация текста

МОСКОВСКИЙ ГОРОДСКОЙ УНИВЕРСИТЕТ МГПУ

В то время как большие блоки текста (предложения, абзацы и т. д.) могут предоставить полезную информацию для передачи читателю, существует несколько аналитических решений, которые могут извлечь информацию из необработанного текста.

Почти во всех случаях полезно разобрать текст на отдельные слова. Часто текст разбивается на составные токены, где каждый токен представляет собой последовательность символов, сгруппированных вместе, чтобы сформировать семантическую единицу.

Токенизация текста

МОСКОВСКИЙ ГОРОДСКОЙ УНИВЕРСИТЕТ МГПУ

Даже передовые **методы обработки естественного языка (NLP)** обычно включают токенизацию перед обработкой текста.

NLP может быть полезно для проведения анализа, требующего более глубокого понимания текста.

Слова и **токены** полезны, поскольку их можно сопоставлять между документами в данных. Это позволяет делать общие выводы на **агрегированном уровне**.

Например, если у нас есть набор данных, содержащий примечания о продажах, и мы анализируем токен **«заинтересованы»**, можем предположить, что примечания о продажах, содержащие **«заинтересованы»**, связаны с клиентами, которые с большей вероятностью совершат покупку.

Токенизация текста

МОСКОВСКИЙ ГОРОДСКОЙ УНИВЕРСИТЕТ МГПУ

Postgres имеет функциональность, которая делает **токенизацию** довольно простой. Можно начать с использования функции **STRING_TO_ARRAY**, которая разбивает строку на массив с помощью разделителя, например, пробела:

```
SELECT STRING_TO_ARRAY('Danny and Matt are friends.', ' ');
```

```
string_to_array                |
-------------------------------+
{Danny,and,Matt,are,friends.}  |
```

Токенизация текста

МОСКОВСКИЙ ГОРОДСКОЙ УНИВЕРСИТЕТ МГПУ

В этом примере есть знаки препинания, которые лучше убрать, используя функцию **REGEXP_REPLACE**.

Эта функция принимает четыре аргумента:

- текст, который требуется изменить,
- шаблон текста, который требуется заменить,
- текст, который должен его заменить,
- любые дополнительные флаги (чаще всего флаг «g», указывающий, что замена должно происходить глобально или столько раз, сколько встречается шаблон).

Мы можем удалить точку, используя шаблон, который соответствует пунктуации, определенной в строке **\!@#$%^&*()-=_+,.<>/?|[],** и заменяет ее пробелом:

Токенизация текста

МОСКОВСКИЙ ГОРОДСКОЙ УНИВЕРСИТЕТ МГПУ

Мы можем удалить точку, используя шаблон, который соответствует пунктуации, определенной в строке **\!@#$%^&*()-=_+,.<>/?|[],** и заменяет ее пробелом:

```
SELECT REGEXP_REPLACE('Danny and Matt are friends.', '[!,.?-]', ' ', 'g');
```

```
regexp_replace
--------------------------
Danny and Matt are friends
```

Токенизация текста

МОСКОВСКИЙ ГОРОДСКОЙ УНИВЕРСИТЕТ МГПУ

Postgres также включает в себя функцию определения корня, которая полезна для определения **корневой основы токена**. Например, лексемы «быстро» и «быстрее» или «бег» и «бежать» не так уж различаются по своему значению и содержат одну и ту же основу.

Функция **ts_lexize** может помочь стандартизировать текст, возвращая основу слова, например:

```
SELECT TS_LEXIZE('english_stem', 'running');
```

```
ts_lexize|
---------+
{run}    |
```

Practice. Выполнение текстовой аналитики

МОСКОВСКИЙ ГОРОДСКОЙ УНИВЕРСИТЕТ МГПУ

Количественно определить ключевые слова, которые соответствуют рейтингу выше среднего или рейтингу ниже среднего, используя текстовую аналитику. В базе данных есть доступ к некоторым отзывам клиентов, а также к рейтингам вероятности того, что клиент порекомендует своим друзьям компанию.

Эти ключевые слова позволят нам определить сильные и слабые стороны, которые исполнительная команда будет учитывать в будущем.

Practice. Выполнение текстовой аналитики

МОСКОВСКИЙ ГОРОДСКОЙ УНИВЕРСИТЕТ МГПУ

Шаги для выполнения запроса PostgreSQL:

1. посмотрим, какие данные есть:

```
SELECT * FROM customer_survey limit 5;
```

```
rating|feedback                                                                                                                                |
------+----------------------------------------------------------------------------------------------------------------------------------------+
     9|I highly recommend the lemon scooter. It's so fast                                                                                      |
    10|I really enjoyed the sale - I was able to get the Bat for a 20% discount                                                                |
     4|Overall, the experience was ok. I don't think that the customer service rep was really understanding the issue.                         |
     9|The model epsilon has been a fantastic ride - one of the best cars I have ever driven.                                                  |
     9|I've been riding the scooter around town. It's been good in urban areas.                                                                |
```

Мы видим, что есть доступ к числовому рейтингу от 1 до 10 и обратной связи в текстовом формате.

Practice 06-3. Выполнение текстовой аналитики

МОСКОВСКИЙ ГОРОДСКОЙ УНИВЕРСИТЕТ МГПУ

Шаги для выполнения запроса PostgreSQL:

2. Чтобы проанализировать текст, нужно разобрать его на отдельные слова и связанные с ними рейтинги.

```
SELECT UNNEST(STRING_TO_ARRAY(feedback, ' ')) AS word, rating
FROM customer_survey limit 10;
```

word	rating
I	9
highly	9
recommend	9
the	9
lemon	9
scooter.	9
It's	9
so	9
fast	9
I	10

Practice 06-3. Выполнение текстовой аналитики

МОСКОВСКИЙ ГОРОДСКОЙ УНИВЕРСИТЕТ МГПУ

Шаги для выполнения запроса PostgreSQL:

3. Стандартизируем текст с помощью функции **ts_lexize** и **стеммира**(процесс нахождения основы слова для заданного исходного слова) английского языка **english_stem**.

Затем удалим символы, которые не являются буквами в исходном тексте, используя **REGEXP_REPLACE**. Объединив эти две функции вместе с нашим исходным запросом, получим следующее:

Practice 06-3. Выполнение текстовой аналитики

МОСКОВСКИЙ ГОРОДСКОЙ УНИВЕРСИТЕТ МГПУ

Шаги для выполнения запроса PostgreSQL:

3.

```
SELECT
      (TS_LEXIZE('english_stem',
            UNNEST(STRING_TO_ARRAY(
                  REGEXP_REPLACE(feedback, '[^a-zA-Z]+', ' ', 'g'),
                  ' ')
            )))[1] AS token, rating
FROM customer_survey
LIMIT 10;
```

Practice 06-3. Выполнение текстовой аналитики

МОСКОВСКИЙ ГОРОДСКОЙ УНИВЕРСИТЕТ МГПУ

Шаги для выполнения запроса PostgreSQL:

3.

token	rating
	9
high	9
recommend	9
	9
lemon	9
scooter	9
	9
	9
	9
fast	9

Practice 06-3. Выполнение текстовой аналитики

МОСКОВСКИЙ ГОРОДСКОЙ УНИВЕРСИТЕТ МГПУ

Шаги для выполнения запроса PostgreSQL:

4. На следующем шаге найдем **средний рейтинг**, связанный с каждым токеном. Можем сделать это, просто используя предложение **GROUP BY**:

```
SELECT
       (TS_LEXIZE('english_stem',
              UNNEST(STRING_TO_ARRAY(
                     REGEXP_REPLACE(feedback, '[^a-zA-Z]+', ' ', 'g'),
                     ' ')
              )))[1] AS token,
       AVG(rating) AS avg_rating
FROM customer_survey
GROUP BY 1
HAVING COUNT(1) >= 3
ORDER BY 2;
```

Practice 06-3. Выполнение текстовой аналитики

МОСКОВСКИЙ ГОРОДСКОЙ УНИВЕРСИТЕТ МГПУ

Шаги для выполнения запроса PostgreSQL:

4. На следующем шаге найдем **средний рейтинг**, связанный с каждым токеном. Можем сделать это, просто используя предложение **GROUP BY**:

В этом запросе группируем по первому выражению в операторе **SELECT**, где выполняем **токенизацию**.

Теперь можем взять **средний рейтинг**, связанный с каждым токеном.

Убедимся, что берем только токены с более чем парой вхождений, чтобы отфильтровать шум — в этом случае из-за небольшого размера выборки ответов обратной связи нам требуется только, чтобы токен встречался три или более раз (**HAVING COUNT(1) >= 3**).

Наконец, упорядочим результаты по второму выражению — среднему баллу.

token	avg_rating
pop	2.0000000000000000
batteri	2.3333333333333333
servic	2.3333333333333333
custom	2.3333333333333333
issu	2.5000000000000000
long	2.6666666666666667
ship	2.6666666666666667
email	3.5000000000000000
help	4.0000000000000000
one	4.3333333333333333
littl	4.6666666666666667
hook	5.0000000000000000
get	5.0000000000000000
work	5.0000000000000000
	5.1872659176029963
realli	5.5000000000000000
scooter	5.9090909090909091
ride	6.7500000000000000
model	7.3333333333333333
lemon	7.6666666666666667
great	7.7500000000000000
fast	8.0000000000000000
dealership	9.0000000000000000
sale	9.5000000000000000
discount	9.6666666666666667

Practice 06-3. Выполнение текстовой аналитики

МОСКОВСКИЙ ГОРОДСКОЙ УНИВЕРСИТЕТ МГПУ

Шаги для выполнения запроса PostgreSQL:

4. На одном конце спектра видим, что у нас довольно много отрицательных результатов:

pop, вероятно, относится к лопающимся шинам,

batteri, вероятно, относится к проблемам со сроком службы батареи.

С положительной стороны, видим, что клиенты положительно реагируют на **discount**, **sale**,и **dealership**.

Practice 06-3. Выполнение текстовой аналитики

МОСКОВСКИЙ ГОРОДСКОЙ УНИВЕРСИТЕТ МГПУ

Шаги для выполнения запроса PostgreSQL:

5. Проверим предположения, отфильтровав ответы на опросы, содержащие эти токены, с помощью выражения **ILIKE** следующим образом:

Выражение **ILIKE** позволяет нам сопоставлять текст, содержащий шаблон. В этом примере найдем текст, содержащий всплывающий текст, операция нечувствительна к регистру.

Оборачивая это в символы **%**, указываем, что текст может содержать любое количество символов слева или справа.

```
SELECT * FROM customer_survey WHERE feedback ILIKE '%pop%';
```

Practice 06-3. Выполнение текстовой аналитики

МОСКОВСКИЙ ГОРОДСКОЙ УНИВЕРСИТЕТ МГПУ

Шаги для выполнения запроса PostgreSQL:

5. Проверим предположения, отфильтровав ответы на опросы, содержащие эти токены, с помощью выражения **ILIKE** следующим образом:

Выражение **ILIKE** позволяет нам сопоставлять текст, содержащий шаблон. В этом примере найдем текст, содержащий всплывающий текст, операция нечувствительна к регистру.

Оборачивая это в символы **%**, указываем, что текст может содержать любое количество символов слева или справа.

```
SELECT * FROM customer_survey WHERE feedback ILIKE '%pop%';
```

МОСКОВСКИЙ ГОРОДСКОЙ УНИВЕРСИТЕТ
МГПУ

Практическое задание 10. SQL для подготовки данных

Практическое задание 10.
Аналитика с использованием сложных типов данных. Поиск и анализ продаж

МОСКОВСКИЙ ГОРОДСКОЙ УНИВЕРСИТЕТ МГПУ

Руководитель отдела продаж выявил проблему: у отдела продаж нет простого способа найти клиента. К счастью, вы вызвались создать проверенную внутреннюю поисковую систему, которая сделает всех клиентов доступными для поиска по их контактной информации и продуктам, которые они приобрели в прошлом:

1. Используя таблицу **customer_sales**, создайте доступное для поиска представление с одной записью для каждого клиента. Это представление должно быть отключено от столбца **customer_id** и доступно для поиска по всей базе данных, что связано с этим клиентом:

- **имя,**
- **адрес электронной почты,**
- **телефон,**
- **приобретенные продукты.**

Можно также включить и другие поля.

Практическое задание 10.
Аналитика с использованием сложных типов данных. Поиск и анализ продаж

МОСКОВСКИЙ ГОРОДСКОЙ УНИВЕРСИТЕТ МГПУ

2. Создайте доступный для поиска индекс, созданного вами ранее представления.

3. У кулера с водой продавец спрашивает, можете ли вы использовать свой новый поисковый прототип, чтобы найти покупателя по имени Дэнни, купившего скутер Bat. Запросите новое представление с возможностью поиска, используя ключевые слова «Danny Bat». Какое количество строк вы получили?

4. Отдел продаж хочет знать, насколько часто люди покупают скутер и автомобиль. Выполните перекрестное соединение таблицы продуктов с самой собой, чтобы получить все отдельные пары продуктов и удалить одинаковые пары (например, если название продукта совпадает). Для каждой пары выполните поиск в представлении, чтобы узнать, сколько клиентов соответствует обоим продуктам в паре. Можно предположить, что выпуски ограниченной серии можно сгруппировать вместе с их аналогом стандартной модели (например, Bat и Bat Limited Edition можно считать одним и тем же скутером).

МОСКОВСКИЙ ГОРОДСКОЙ УНИВЕРСИТЕТ
МГПУ

Решение

Стратегии материализованного представления с использованием **PostgreSQL**

МОСКОВСКИЙ ГОРОДСКОЙ УНИВЕРСИТЕТ МГПУ

Запросы, возвращающие совокупные, сводные и вычисленные данные, часто используются при разработке приложений.

Иногда эти запросы **выполняются недостаточно быстро**. Кэширование результатов запроса с помощью **Memcached** или **Redis** — распространенный подход к решению этих проблем с производительностью.

Однако они приносят свои собственные проблемы. Прежде чем обратиться к внешнему инструменту, стоит изучить, какие методы предлагает **PostgreSQL** для кэширования результатов запросов.

Пример БД

МОСКОВСКИЙ ГОРОДСКОЙ УНИВЕРСИТЕТ МГПУ

Рассмотрим подходы, используя пример домена(бд) упрощенной системы учета.

Счета могут иметь много транзакций.

Транзакции могут быть записаны заранее и вступят в силу только после их завершения.

Дебет, вступивший в силу 9 марта, можно ввести в марте.

Образцы данных и запросы

Для этого примера мы создадим 30 000 учетных записей, в среднем по 50 транзакций в каждой.

createdb pg_cache_demo

```
CREATE DATABASE pg_cache_demo
WITH
OWNER = admin
ENCODING = 'UTF8'
LC_COLLATE = 'en_US.UTF-8'
LC_CTYPE = 'en_US.UTF-8'
TABLESPACE = pg_default
CONNECTION LIMIT = -1
IS_TEMPLATE = False;
```

Образцы данных и запросы

МОСКОВСКИЙ ГОРОДСКОЙ УНИВЕРСИТЕТ МГПУ

psql -f accounts.sql pg_cache_demo
psql -f **transactions.sql** pg_cache_demo

Updated Rows	1500000
Query	with r as (
	select (random() * 29999)::bigint as account_offset
	from generate_series(1, 1500000)
	)
	insert into transactions(name, amount, post_time)
	select
	(select name from accounts offset account_offset limit 1),
	((random()-0.5)*1000)::numeric(8,2),
	current_timestamp + '90 days'::interval - (random()*1000 || ' days')::interval
	from r
Start time	Mon May 22 13:33:24 MSK 2023
Finish time	Mon May 22 13:41:53 MSK 2023

```
with r as (
select (random() * 29999)::bigint as account_offset
from generate_series(1, 1500000)
)
insert into transactions(name, amount, post_time)
select
(select name from accounts offset account_offset limit 1),
((random()-0.5)*1000)::numeric(8,2),
current_timestamp + '90 days'::interval - (random()*1000 || '
days')::interval
from r
;
```

Образцы данных и запросы

МОСКОВСКИЙ ГОРОДСКОЙ УНИВЕРСИТЕТ МГПУ

Наш запрос, для которого будем оптимизировать, — это **поиск баланса счетов**.

Для начала создадим **представление**, которое находит балансы для всех счетов.

Представление PostgreSQL — это сохраненный запрос.

После создания выбор из представления точно такой же, как и выбор из исходного запроса, т. е. каждый раз выполняется повторный запрос.

Образцы данных и запросы

МОСКОВСКИЙ ГОРОДСКОЙ УНИВЕРСИТЕТ МГПУ

```
create view account_balances as
select
       name,
       coalesce(
              sum(amount) filter (where post_time <= current_timestamp),
              0
       ) as balance
from accounts
       left join transactions using(name)
group by name;
```

Теперь просто выбираем все строки с отрицательным балансом.

```
select * from account_balances where balance < 0;
```

Вывод

МОСКОВСКИЙ ГОРОДСКОЙ УНИВЕРСИТЕТ МГПУ

После нескольких запусков формирования кэша, этот запрос занимает примерно **30-40** мс для 30000 записей.

Изучим несколько решений по оптимизации транзакционных действий.

Чтобы сохранить их пространство имен, создадим отдельные схемы для каждого подхода.

Получается для **1500000** записей(стандартная БД) запрос будет длиться порядка **1500-2000** мс.

Оптимизация запроса

МОСКОВСКИЙ ГОРОДСКОЙ УНИВЕРСИТЕТ МГПУ

Чтобы сохранить их пространство имен, создадим отдельные схемы для каждого подхода.

```
create schema matview;
create schema eager;
create schema lazy;
```

Материализованные представления PostgreSQL

МОСКОВСКИЙ ГОРОДСКОЙ УНИВЕРСИТЕТ МГПУ

Простейший способ повысить производительность — использовать **материализованное представление**.

Материализованное представление — это снимок запроса, сохраненный в таблице.

```
create materialized view matview.account_balances as
select
        name,
        coalesce(
        sum(amount) filter (where post_time <= current_timestamp),
        0
        ) as balance
from accounts
        left join transactions using(name)
group by name;
```

Материализованные представления PostgreSQL

МОСКОВСКИЙ ГОРОДСКОЙ УНИВЕРСИТЕТ МГПУ

Поскольку **материализованное представление** на самом деле представляет собой таблицу, мы можем создавать индексы.

```
create index on matview.account_balances (name);
create index on matview.account_balances (balance);
```

Чтобы получить баланс из каждой строки, выбираем из материализованного представления необходимые данные.

Материализованные представления PostgreSQL

Чтобы получить баланс из каждой строки, выбираем из материализованного представления необходимые данные.

```
select * from matview.account_balances where balance < 0;
```

Время выполнения запроса для 30000 записей около **3-7** мс, что в 10 раз быстрее обычного запроса.

К сожалению, у этих **материализованных представлений** есть два существенных недостатка.

Во-первых, они обновляются только по требованию.

Во-вторых, необходимо обновить все материализованное представление; нет способа обновить только одну устаревшую строку.

Материализованные представления PostgreSQL

МОСКОВСКИЙ ГОРОДСКОЙ УНИВЕРСИТЕТ МГПУ

Чтобы получить баланс из каждой строки, выбираем из материализованного представления необходимые данные.

```
refresh materialized view matview.account_balances;
```

В случае, когда допустимы устаревшие данные, этот метрод является отличным решением. Но если данные всегда должны быть актуальными, требуется другой метод оптимизации.

Материализованные представления PostgreSQL
Жадный метод.

МОСКОВСКИЙ ГОРОДСКОЙ УНИВЕРСИТЕТ МГПУ

Следующий подход состоит в том, чтобы материализовать запрос в виде таблицы, которая с обновляется всякий раз, когда происходит изменение, которое делает строку неактуальной. Метод использует **триггер**.

Триггер — это фрагмент кода, который запускается, когда происходит какое-либо событие, такое как вставка или обновление.

Материализованные представления PostgreSQL
Жадный метод.

МОСКОВСКИЙ ГОРОДСКОЙ УНИВЕРСИТЕТ МГПУ

1. Создать таблицу для хранения материализованных строк.

```
create table eager.account_balances(
      name varchar primary key references accounts
            on update cascade
            on delete cascade,
      balance numeric(9,2) not null default 0
);

create index on eager.account_balances (balance);
```

Материализованные представления PostgreSQL
Жадный метод.

МОСКОВСКИЙ ГОРОДСКОЙ УНИВЕРСИТЕТ МГПУ

2. Рассмотреть методы, на основании которых данные **account_balances** станут не актуальны или изменятся.

1. **Создана новая учетная запись**.
2. **Учетная запись обновлена или удалена**.
3. **Транзакция вставлена, обновлена или удалена**.

Материализованные представления PostgreSQL

Жадный метод.

МОСКОВСКИЙ ГОРОДСКОЙ УНИВЕРСИТЕТ МГПУ

1. Создана новая учетная запись.

При вставке учетной записи нам необходимо создать запись **account_balances** с нулевым балансом для новой учетной записи.

```
create function eager.account_insert() returns trigger
        security definer
        language plpgsql
as $$
        begin
                insert into eager.account_balances(name) values(new.name);
                return new;
        end;
$$;

create trigger account_insert after insert on accounts
        for each row execute procedure eager.account_insert();
```

Материализованные представления PostgreSQL
Жадный метод.

МОСКОВСКИЙ ГОРОДСКОЙ УНИВЕРСИТЕТ МГПУ

2. Учетная запись обновлена или удалена.

Обновление и удаление учетной записи будут обрабатываться автоматически, поскольку внешний ключ учетной записи объявляется как при каскадном обновлении при каскадном удалении.

Материализованные представления PostgreSQL
Жадный метод.

МОСКОВСКИЙ ГОРОДСКОЙ УНИВЕРСИТЕТ МГПУ

3. Транзакция вставлена, обновлена или удалена.

Вставка, обновление и удаление транзакций имеют одну общую черту: они делают недействительным баланс счета. Поэтому первым шагом является определение функции обновления баланса учетной записи.

Материализованные представления PostgreSQL

Жадный метод.

МОСКОВСКИЙ ГОРОДСКОЙ УНИВЕРСИТЕТ

```
create function eager.refresh_account_balance(_name varchar)
      returns void
      security definer
      language sql
as $$
      update eager.account_balances
      set balance=
            (
                  select sum(amount)
                  from transactions
                  where account_balances.name=transactions.name
                  and post_time <= current_timestamp
            )
      where name=_name;
$$;
```

Материализованные представления PostgreSQL
Жадный метод.

МОСКОВСКИЙ ГОРОДСКОЙ УНИВЕРСИТЕТ МГПУ

3. Транзакция вставлена, обновлена или удалена.

Далее создать **триггер-функцию**, которая вызывает **refresh_account_balance** всякий раз, когда вставляется транзакция.

```
create function eager.transaction_insert()
        returns trigger
        security definer
        language plpgsql
as $$
        begin
                perform eager.refresh_account_balance(new.name);
                return new;
        end;
$$;
create trigger eager_transaction_insert after insert on transactions
        for each row execute procedure eager.transaction_insert();
```

МОСКОВСКИЙ ГОРОДСКОЙ УНИВЕРСИТЕТ
МГПУ

Практическое задание 10. SQL для подготовки данных

Практическое задание 10.
Аналитика с использованием сложных типов данных. Поиск и анализ продаж

МОСКОВСКИЙ ГОРОДСКОЙ УНИВЕРСИТЕТ МГПУ

Руководитель отдела продаж выявил проблему: у отдела продаж нет простого способа найти клиента. К счастью, вы вызвались создать проверенную внутреннюю поисковую систему, которая сделает всех клиентов доступными для поиска по их контактной информации и продуктам, которые они приобрели в прошлом:

1. Используя таблицу **customer_sales**, создайте доступное для поиска представление с одной записью для каждого клиента. Это представление должно быть отключено от столбца **customer_id** и доступно для поиска по всей базе данных, что связано с этим клиентом:

- **имя,**
- **адрес электронной почты,**
- **телефон,**
- **приобретенные продукты.**

Можно также включить и другие поля.

Практическое задание 10.
Аналитика с использованием сложных типов данных. Поиск и анализ продаж

МОСКОВСКИЙ ГОРОДСКОЙ УНИВЕРСИТЕТ МГПУ

2. Создайте доступный для поиска индекс, созданного вами ранее представления.

3. У кулера с водой продавец спрашивает, можете ли вы использовать свой новый поисковый прототип, чтобы найти покупателя по имени Дэнни, купившего скутер Bat. Запросите новое представление с возможностью поиска, используя ключевые слова «Danny Bat». Какое количество строк вы получили?

4. Отдел продаж хочет знать, насколько часто люди покупают скутер и автомобиль. Выполните перекрестное соединение таблицы продуктов с самой собой, чтобы получить все отдельные пары продуктов и удалить одинаковые пары (например, если название продукта совпадает). Для каждой пары выполните поиск в представлении, чтобы узнать, сколько клиентов соответствует обоим продуктам в паре. Можно предположить, что выпуски ограниченной серии можно сгруппировать вместе с их аналогом стандартной модели (например, Bat и Bat Limited Edition можно считать одним и тем же скутером).

Практическое задание 10. **Решение**
Аналитика с использованием сложных типов данных. Поиск и анализ продаж

МОСКОВСКИЙ ГОРОДСКОЙ УНИВЕРСИТЕТ МГПУ

1. Создать материализованное представление для таблицы **customer_sales:**

```
CREATE MATERIALIZED VIEW customer_search AS (
        SELECT
                customer_json -> 'customer_id' AS customer_id, customer_json,
                to_tsvector('english', customer_json) AS search_vector
        FROM customer_sales
);
```

2. Создать индекс **GIN** в представлении:

```
CREATE INDEX customer_search_gin_idx ON customer_search USING GIN(search_vector);
```

МОСКОВСКИЙ ГОРОДСКОЙ УНИВЕРСИТЕТ МГПУ

3. Выполнить запрос, используя новую базу данных с возможностью поиска:

```
SELECT
      customer_id,
      customer_json
FROM customer_search
WHERE search_vector @@ plainto_tsquery('english', 'Danny Bat');
```

МОСКОВСКИЙ ГОРОДСКОЙ УНИВЕРСИТЕТ МГПУ

4. Вывести уникальный список скутеров и автомобилей (и удаление ограниченных выпусков) с помощью **DISTINCT**:

```
SELECT DISTINCT
      p1.model,
      p2.model
FROM products p1
      LEFT JOIN products p2 ON TRUE
WHERE p1.product_type = 'scooter'
      AND p2.product_type = 'automobile'
      AND p1.model NOT ILIKE '%Limited
Edition%';
```

МОСКОВСКИЙ ГОРОДСКОЙ УНИВЕРСИТЕТ МГПУ

5. Преобразование вывода в запрос:

```
SELECT DISTINCT
plainto_tsquery('english', p1.model) &&
plainto_tsquery('english', p2.model)
FROM products p1
LEFT JOIN products p2 ON TRUE
WHERE p1.product_type = 'scooter'
AND p2.product_type = 'automobile'
AND p1.model NOT ILIKE '%Limited Edition%';
```

МОСКОВСКИЙ ГОРОДСКОЙ УНИВЕРСИТЕТ МГПУ

6. Запрос базы данных, используя каждый из объектов **tsquery**, и подсчитать вхождения для каждого объекта:

```
SELECT
        sub.query,
        (
                SELECT COUNT(1)
                FROM customer_search
                WHERE customer_search.search_vector @@ sub.query)
FROM    (
        SELECT DISTINCT
                plainto_tsquery('english', p1.model) &&
                plainto_tsquery('english', p2.model) AS query
        FROM products p1
        LEFT JOIN products p2 ON TRUE
        WHERE p1.product_type = 'scooter’
        AND p2.product_type = 'automobile’
        AND p1.model NOT ILIKE '%Limited Edition%’
        ) sub
ORDER BY 2 DESC;
```

МОСКОВСКИЙ
ГОРОДСКОЙ
УНИВЕРСИТЕТ
МГПУ

СПАСИБО ЗА ВНИМАНИЕ