

МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ  
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Институт №8 «Компьютерные науки и прикладная математика»  
Кафедра 806 «Вычислительная математика и программирование»

**Лабораторная работа №3**  
**по курсу «Проектирование баз данных»**

Выполнила: Прудникова А. А.  
Группа: М8О-114СВ-24  
Преподаватель: Моргунов Е. П.

Москва, 2024

## Задание 2

Предположим, что возникла необходимость хранить в одном столбце таблицы данные, представленные с различной точностью. Это могут быть, например, результаты физических измерений разнородных показателей или различные медицинские показатели здоровья пациентов (результаты анализов). В таком случае можно использовать тип `numeric` без указания масштаба и точности. Теперь сделайте выборку из таблицы и посмотрите, что все эти разнообразные значения сохранены именно в том виде, как вы их вводили.

### Запрос

```
CREATE TABLE test_numeric
( measurement numeric,
  description text
);

INSERT INTO test_numeric
VALUES ( 1234567890.0987654321,
        'Точность 20 знаков, масштаб 10 знаков' );

INSERT INTO test_numeric
VALUES ( 1.5,
        'Точность 2 знака, масштаб 1 знак' );

INSERT INTO test_numeric
VALUES ( 0.12345678901234567890,
        'Точность 21 знак, масштаб 20 знаков' );

INSERT INTO test_numeric
VALUES ( 1234567890,
        'Точность 10 знаков, масштаб 0 знаков (целое число)' );

SELECT * FROM test_numeric;
```

### Результат

measurement	description
abc Filter...	abc Filter...
1234567890.0987654321	Точность 20 знаков, масштаб 10 знаков
1.5	Точность 2 знака, масштаб 1 знак
0.12345678901234567890	Точность 21 знак, масштаб 20 знаков
1234567890	Точность 10 знаков, масштаб 0 знаков (целое число)

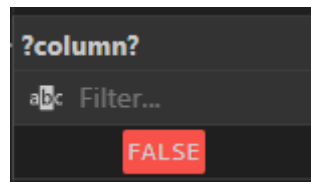
## Задание 4

При работе с числами типов `real` и `double precision` нужно помнить, что сравнение двух чисел с плавающей точкой на предмет равенства их значений может привести к неожиданным результатам.

### Запрос

```
SELECT '5e-324'::double precision > '4e-324'::double precision;
```

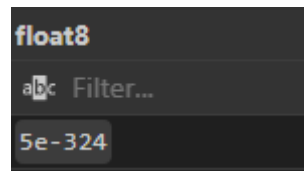
### Результат



### Запрос

```
SELECT '5e-324'::double precision;
```

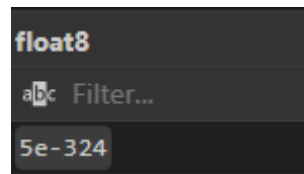
### Результат



### Запрос

```
SELECT '4e-324'::double precision;
```

### Результат



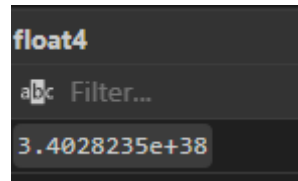
## Задание

Самостоятельно проведите аналогичные эксперименты с очень большими числами, находящимися на границе допустимого диапазона для чисел типов `real` и `double precision`.

## Запрос

```
SELECT '3.4028235e+38'::real;
```

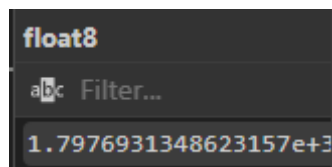
## Результат



## Запрос

```
SELECT '1.7976931348623157e+308'::double precision;
```

## Результат

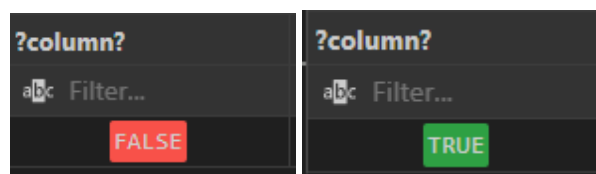


## Запрос

```
SELECT '3.4028235e+38'::real > '3.4028234e+38'::real;
```

```
SELECT '1.7976931348623157e+308'::double precision >  
'1.7976931348623156e+308'::double precision;
```

## Результат



## Задание 8

Пусть теперь столбец id будет первичным ключом таблицы.

```
CREATE TABLE test_serial  
    ( id serial PRIMARY KEY,  
      name text  
    );
```

Теперь выполните следующие команды для добавления строк в таблицу и удаления одной строки из нее. Для пошагового управления этим процессом выполняйте выборку данных из таблицы с помощью команды SELECT после каждой команды вставки или удаления.

```
INSERT INTO test_serial ( name ) VALUES ( 'Вишневая' );
```

Явно зададим значение столбца id:

```
INSERT INTO test_serial ( id, name ) VALUES ( 2, 'Прохладная' );
```

При выполнении этой команды СУБД выдаст сообщение об ошибке. Почему?

```
INSERT INTO test_serial ( name ) VALUES ( 'Грушевая' );
```

Повторим эту же команду. Теперь все в порядке. Почему?

```
INSERT INTO test_serial ( name ) VALUES ( 'Грушевая' );
```

Добавим еще одну строку.

```
INSERT INTO test_serial ( name ) VALUES ( 'Зеленая' );
```

А теперь удалим ее же.

```
DELETE FROM test_serial WHERE id = 4;
```

Добавим последнюю строку.

```
INSERT INTO test_serial ( name ) VALUES ( 'Луговая' );
```

Теперь сделаем выборку.

```
SELECT * FROM test_serial;
```

Вы увидите, что в нумерации образовалась «дыра». Это из-за того, что при формировании нового значения из последовательности поиск максимального значения, уже имеющегося в столбце, не выполняется.

## Запрос

```
CREATE TABLE test_serial
( id serial PRIMARY KEY,
  name text
);
```

```
INSERT INTO test_serial ( name ) VALUES ( 'Вишневая' );
```

```
INSERT INTO test_serial ( id, name ) VALUES ( 2, 'Прохладная' );
```

```
SELECT * FROM test_serial;
```

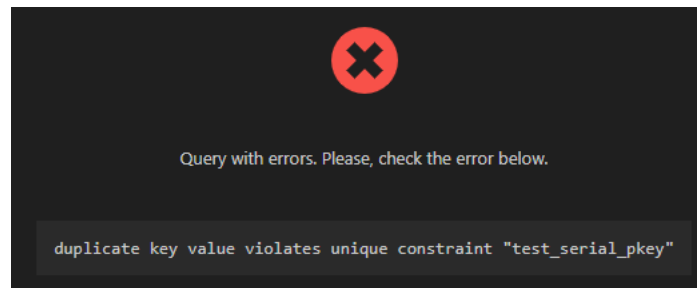
## Результат

id	name
<input type="text" value="abc Filter..."/>	<input type="text" value="abc Filter..."/>
1	Вишневая
2	Прохладная

## Запрос

```
INSERT INTO test_serial ( name ) VALUES ( 'Грушевая' );
```

## Результат



## Запрос

```
INSERT INTO test_serial ( name ) VALUES ( 'Грушевая' );
```

## Результат

```
INSERT successfully executed. 1 rows were affected. 3:22:35 PM
```

## Решение

В первом случае при вставке появилось сообщение об ошибке, так как не был указан ID для вставляемой строки, поэтому был использован ID = 2, не являющийся уникальным. Однако при обработке запроса произошел сдвиг, поэтому при повторной попытке вставки был сгенерирован уникальный ID (ID = 3).

## Запрос

```
INSERT INTO test_serial ( name ) VALUES ( 'Зеленая' );
```

```
SELECT * FROM test_serial;
```

## Результат

id	name
<input type="text" value="Filter..."/>	<input type="text" value="Filter..."/>
1	Вишневая
2	Прохладная
3	Грушевая
4	Зеленая

## Запрос

```
INSERT INTO test_serial ( name ) VALUES ( 'Зеленая' );
```

```
DELETE FROM test_serial WHERE id = 4;

INSERT INTO test_serial ( name ) VALUES ( 'Луговая' );

SELECT * FROM test_serial;
```

## Результат

id	name
abc Filter...	abc Filter...
1	Вишневая
2	Прохладная
3	Грушевая
7	Луговая

## Задание 12

Формат ввода и вывода даты можно изменить с помощью конфигурационного параметра `datestyle`. Значение этого параметра состоит из двух компонентов: первый управляет форматом вывода даты, а второй регулирует порядок следования составных частей даты (год, месяц, день) при вводе и выводе. Текущее значение этого параметра можно узнать с помощью команды `SHOW`:

```
SHOW datestyle;
```

Поскольку параметр `datestyle` состоит фактически из двух частей, которые можно задавать не только обе сразу, но и по отдельности, изменим только порядок следования составных частей даты, не изменяя формат вывода с ISO на какой-либо другой.

## Запрос

```
SHOW datestyle;
```

```
SET datestyle TO 'MDY';
```

```
SHOW datestyle;
```

## Результат

DateStyle
abc Filter...
ISO, DMY

DateStyle
abc Filter...
ISO, MDY

## Запрос

```
SET datestyle TO 'Postgres, DMY';  
  
SHOW datestyle;
```

## Результат

DateStyle ↑
abc Filter...
Postgres, DMY

## Запрос

```
SELECT '18-05-2016'::date;
```

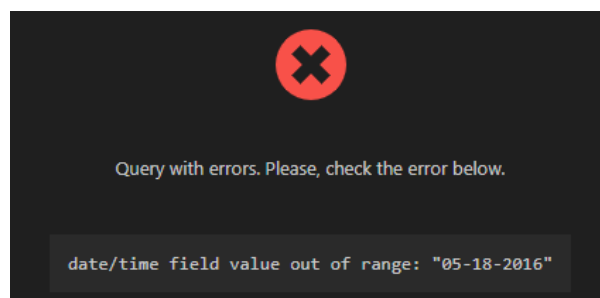
## Результат

date
abc Filter...
18-05-2016

## Запрос

```
SELECT '05-18-2016'::date;
```

## Результат



## Запрос

```
SELECT '2016-05-18'::date;
```

## Результат

date
abc Filter...
18-05-2016



## Запрос

```
SELECT '2016-05-18 12:34:56'::timestamp;
```

## Результат

timestamp
abc Filter...
Wed 18 May 12:34:56 2016

## Запрос

```
SET datestyle TO 'SQL, DMY';
```

```
SELECT current_date, current_timestamp;
```

```
SELECT '18-05-2016'::date;
```

```
SELECT '2016-05-18'::date;
```

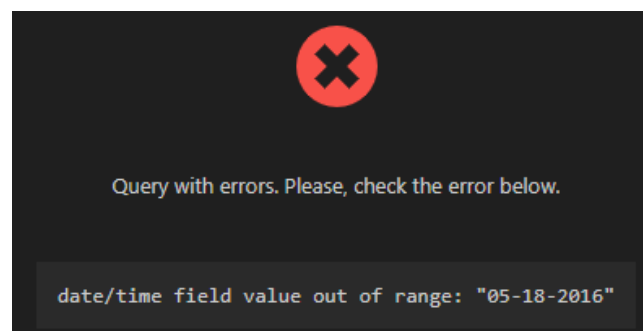
```
SELECT '05-18-2016'::date;
```

## Результат

current_date	current_timestamp
abc Filter...	abc Filter...
14/10/2024	14/10/2024 15:08:59.514057 UTC

date
abc Filter...
18/05/2016

date
abc Filter...
18/05/2016



## Запрос

```
SET datestyle TO 'German, DMY';
```

```
SELECT current_date, current_timestamp;
```

```
SELECT '18-05-2016'::date;
```

```
SELECT '2016-05-18'::date;
```

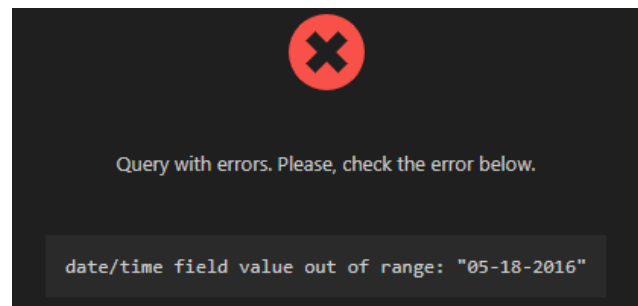
```
SELECT '05-18-2016'::date;
```

## Результат

current_date	current_timestamp
abc Filter...	abc Filter...
14.10.2024	14.10.2024 15:11:55.849499 UTC

date
abc Filter...
18.05.2016

date
abc Filter...
18.05.2016



## Задание 15

Поэкспериментируйте с функцией `to_char`, извлекая из значения типа `timestamp` различные поля и располагая их в нужном вам порядке.

### Запрос

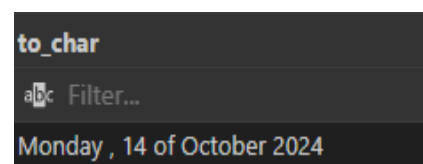
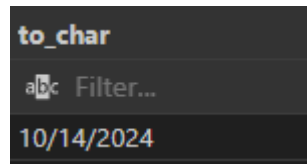
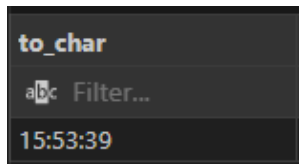
```
SET datestyle TO 'DMY';
```

```
SELECT to_char(current_timestamp, 'HH24:MI:SS');
```

```
SELECT to_char(current_timestamp, 'MM/DD/YYYY');
```

```
SELECT to_char(current_timestamp, 'Day, DD "of" Month YYYY');
```

## Результат



## Задание 21

Можно с высокой степенью уверенности предположить, что при прибавлении интервалов к датам и временным отметкам PostgreSQL учитывает тот факт, что различные месяцы имеют разное число дней. Но как это реализуется на практике? Например, что получится при прибавлении интервала в 1 месяц к последнему дню января и к последнему дню февраля? Сначала сделайте обоснованные предположения о результатах следующих двух команд, а затем проверьте предположения на практике и проанализируйте полученные результаты:

```
SELECT ( '2016-01-31'::date + '1 mon'::interval ) AS new_date;  
SELECT ( '2016-02-29'::date + '1 mon'::interval ) AS new_date;
```

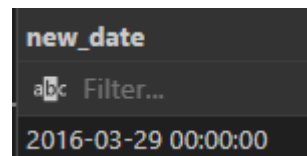
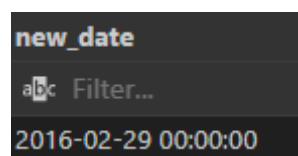
## Решение

**Предположение.** Поскольку январь имеет 31 день, а при прибавлении 1 месяца мы переходим в февраль, который имеет меньше дней (28 или 29), то результатом первого запроса будет последний день февраля. Также 2016 год был високосным, и 29 февраля существует, при прибавлении 1 месяца мы перейдем в март, который имеет 31 день. Поэтому результатом второго запроса будет 31 марта.

## Запрос

```
SELECT ( '2016-01-31'::date + '1 mon'::interval ) AS new_date;  
  
SELECT ( '2016-02-29'::date + '1 mon'::interval ) AS new_date;
```

## Результат



## Решение

Во втором случае предположение не совпало с результатом на практике. Вместо 31 марта мы получили 29 марта. Это связано с тем, что PostgreSQL при прибавлении интервала в 1 месяц, сохраняет день месяца, насколько это возможно.

## Задание 30

Обратимся к таблице, создаваемой с помощью команды

```
CREATE TABLE test_bool  
(  
    a boolean,  
    b text  
);
```

Как вы думаете, какие из приведенных ниже команд содержат ошибку? Проверьте свои предположения практически, выполнив эти команды.

## Запрос / Результат

```
INSERT INTO test_bool VALUES ( TRUE, 'yes' );
```

```
INSERT successfully executed. 1 rows were affected.
```

```
INSERT INTO test_bool VALUES ( yes, 'yes' );
```

```
column "yes" does not exist
```

```
INSERT INTO test_bool VALUES ( 'yes', true );
```

```
INSERT successfully executed. 1 rows were affected. 7:10:20 PM
```

```
INSERT INTO test_bool VALUES ( 'yes', TRUE );
```

```
INSERT successfully executed. 1 rows were affected. 7:11:01 PM
```

```
INSERT INTO test_bool VALUES ( '1', 'true' );
```

```
INSERT successfully executed. 1 rows were affected. 7:11:31 PM
```

```
INSERT INTO test_bool VALUES ( 1, 'true' );
```

```
column "a" is of type boolean but expression is of type integer
```

```
INSERT INTO test_bool VALUES ( 't', 'true' );
```

```
INSERT successfully executed. 1 rows were affected. 7:13:02 PM
```

```
INSERT INTO test_bool VALUES ( 't', truth );
```

```
column "truth" does not exist
```

```
INSERT INTO test_bool VALUES ( true, true );
```

```
INSERT successfully executed. 1 rows were affected. 7:14:00 PM
```

```
INSERT INTO test_bool VALUES ( 1::boolean, 'true' );
```

```
INSERT successfully executed. 1 rows were affected. 7:14:24 PM
```

```
INSERT INTO test_bool VALUES ( 111::boolean, 'true' );
```

```
INSERT successfully executed. 1 rows were affected. 7:15:10 PM
```

```
SELECT * FROM test_bool;
```

a	b
abc Filter...	abc Filter...
TRUE	yes
TRUE	true
TRUE	true
TRUE	true
TRUE	true
TRUE	true
TRUE	true
TRUE	true

## Задание 33

Предположим, что пилоты авиакомпании имеют возможность высказывать свои пожелания насчет конкретных блюд, из которых должен состоять их обед во время полета. Для учета пожеланий пилотов необходимо модифицировать таблицу `pilots`, с которой мы работали в разделе 4.5.

```
CREATE TABLE pilots
(
    pilot_name text,
    schedule integer[],
    meal text[]
);
```

```
INSERT INTO pilots
VALUES ( 'Ivan', '{ 1, 3, 5, 6, 7 }'::integer[],
        '{ "сосиска", "макароны", "кофе" }'::text[]
),
( 'Petr', '{ 1, 2, 5, 7 }'::integer [],
    '{ "котлета", "каша", "кофе" }'::text[]
),
( 'Pavel', '{ 2, 5 }'::integer[],
    '{ "сосиска", "каша", "кофе" }'::text[]
),
( 'Boris', '{ 3, 5, 6 }'::integer[],
    '{ "котлета", "каша", "чай" }'::text[]
);
```

Обратите внимание, что каждое из текстовых значений, включаемых в литерал массива, заключается в двойные кавычки, а в качестве типа данных указывается `text[]`.




Создайте новую версию таблицы и соответственно измените команду INSERT, чтобы в ней содержались литералы двумерных массивов. Они будут выглядеть примерно так:




```
'{ { "сосиска", "макароны", "кофе" },  
{ "котлета", "каша", "кофе" },  
{ "сосиска", "каша", "кофе" },  
{ "котлета", "каша", "чай" } }'::text[][]
```

## Запрос

```
CREATE TABLE pilots  
(  
    pilot_name text,  
    schedule integer[],  
    meal text[]  
);  
  
INSERT INTO pilots  
VALUES ( 'Ivan', '{ 1, 3, 5, 6, 7 }'::integer[],  
'{ "сосиска", "макароны", "кофе" }'::text[]  
) ,  
( 'Petr', '{ 1, 2, 5, 7 }'::integer [],  
'{ "котлета", "каша", "кофе" }'::text[]  
) ,  
( 'Pavel', '{ 2, 5 }'::integer[],  
'{ "сосиска", "каша", "кофе" }'::text[]  
) ,  
( 'Boris', '{ 3, 5, 6 }'::integer[],  
'{ "котлета", "каша", "чай" }'::text[]  
);  
  
SELECT * FROM pilots;  
  
SELECT * FROM pilots WHERE meal[ 1 ] = 'сосиска';
```

## Результат

pilot_name	schedule	meal
 Filter...	 Filter...	 Filter...
Ivan	[ 1, 3, 5, 6, 7 ]	[ "сосиска", "макароны", "кофе" ]
Petr	[ 1, 2, 5, 7 ]	[ "котлета", "каша", "кофе" ]
Pavel	[ 2, 5 ]	[ "сосиска", "каша", "кофе" ]
Boris	[ 3, 5, 6 ]	[ "котлета", "каша", "чай" ]

pilot_name	schedule	meal
 Filter...	 Filter...	 Filter...
Ivan	[ 1, 3, 5, 6, 7 ]	[ "сосиска", "макароны", "кофе" ]
Pavel	[ 2, 5 ]	[ "сосиска", "каша", "кофе" ]

## Задание

Сделайте ряд выборок и обновлений строк в этой таблице. Для обращения к элементам двумерного массива нужно использовать два индекса. Не забывайте, что по умолчанию номера индексов начинаются с единицы.

## Запрос

```
CREATE TABLE pilots
(
    pilot_name text,
    schedule integer[],
    meal text[][]
);

INSERT INTO pilots
VALUES ('Ivan', '{1, 3, 5, 6, 7}'::integer[],
    '{{
        "сосиска", "макароны", "кофе"
    },
    {
        "сосиска", "макароны", "кофе"
    },
    {
        "сосиска", "макароны", "кофе"
    },
    {
        "сосиска", "макароны", "кофе"
    }}'::text[][]),
('Petr', '{1, 2, 5, 7}'::integer[],
    '{{
        "котлета", "каша", "кофе"
    },
    {
        "котлета", "каша", "кофе"
    },
    {
        "котлета", "каша", "кофе"
    },
    {
        "котлета", "каша", "кофе"
    }}'::text[][]),
('Pavel', '{2, 5}'::integer[],
    '{{
        "сосиска", "каша", "кофе"
    },
    {
        "сосиска", "каша", "кофе"
    },
    {
        "сосиска", "каша", "кофе"
    },
    {
        "сосиска", "каша", "кофе"
    }}'::text[][]);
```

```

        {
            "сосиска", "каша", "кофе"
        },
        {
            "сосиска", "каша", "кофе"
        }}':::text[][]),
('Boris', '{3, 5, 6}':::integer[],
'{{
    "котлета", "каша", "чай"
}},
{
    "котлета", "каша", "чай"
}},
{
    "котлета", "каша", "чай"
}},
{
    "котлета", "каша", "чай"
}}':::text[][]);

```

```
SELECT * FROM pilots;
```

## Результат

pilot_name	schedule	meal
<input type="text" value="Filter..."/>	<input type="text" value="Filter..."/>	<input type="text" value="Filter..."/>
Ivan	[ 1, 3, 5, 6, 7 ]	[ [ "сосиска", "макароны", "кофе" ], [ "сосиска", "макароны", "кофе" ], [ "сосиска", "макароны", "кофе" ] ]
Petr	[ 1, 2, 5, 7 ]	[ [ "котлета", "каша", "кофе" ], [ "котлета", "каша", "кофе" ], [ "котлета", "каша", "кофе" ] ]
Pavel	[ 2, 5 ]	[ [ "сосиска", "каша", "кофе" ], [ "сосиска", "каша", "кофе" ], [ "сосиска", "каша", "кофе" ] ]
Boris	[ 3, 5, 6 ]	[ [ "котлета", "каша", "чай" ], [ "котлета", "каша", "чай" ], [ "котлета", "каша", "чай" ] ]

## Запрос

```
SELECT meal FROM pilots WHERE pilot_name = 'Ivan';
```

```
SELECT pilot_name, meal[1][3] FROM pilots;
```

## Результат

```
[ [ "сосиска", "макароны", "кофе" ], [ "сосиска", "макароны", "кофе" ], [ "сосиска", "макароны", "кофе" ] ]
```

pilot_name	meal
<input type="text" value="Filter..."/>	<input type="text" value="Filter..."/>
Ivan	кофе
Petr	кофе
Pavel	кофе
Boris	чай



## Запрос

```
UPDATE pilots
SET meal[2][2] = 'омлет';

SELECT meal[2][2] from pilots;

UPDATE pilots
SET meal[2][2] = 'бутерброд'
WHERE pilot_name = 'Boris';

SELECT * FROM pilots;
```

## Результат

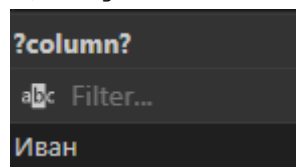
pilot_name	s...	meal
abc Filter...	abc Filt	abc Filter...
Ivan	[ 1, ...	[ [ "сосиска", "макароны", "кофе" ], [ "сосиска", "омлет"
Petr	[ 1, ...	[ [ "котлета", "каша", "кофе" ], [ "котлета", "омлет", "
Pavel	[ 2, ...	[ [ "сосиска", "каша", "кофе" ], [ "сосиска", "омлет", "
Boris	[ 3, ...	[ [ "котлета", "каша", "чай" ], [ "котлета", "бутерброд"

## Задание 35

Для работы с типами JSON предусмотрено много различных функций и операторов, представленных в разделе документации 9.15 «Функции и операторы JSON». Самостоятельно ознакомьтесь с ними, используя описанную технологию работы с командой SELECT.

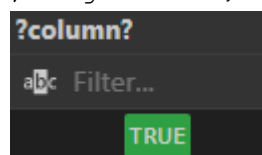
- Извлечение значения по ключу

```
SELECT '{ "name": "Иван", "age": 30 }'::jsonb->>'name';
```



- Проверка наличия ключа в JSON-объекте

```
SELECT '{ "name": "Иван", "age": 30 }'::jsonb ? 'age';
```



- Преобразование JSON-объекта в текст

```
SELECT CAST('{ "name": "Иван", "age": 30 }'::jsonb AS text);
```

