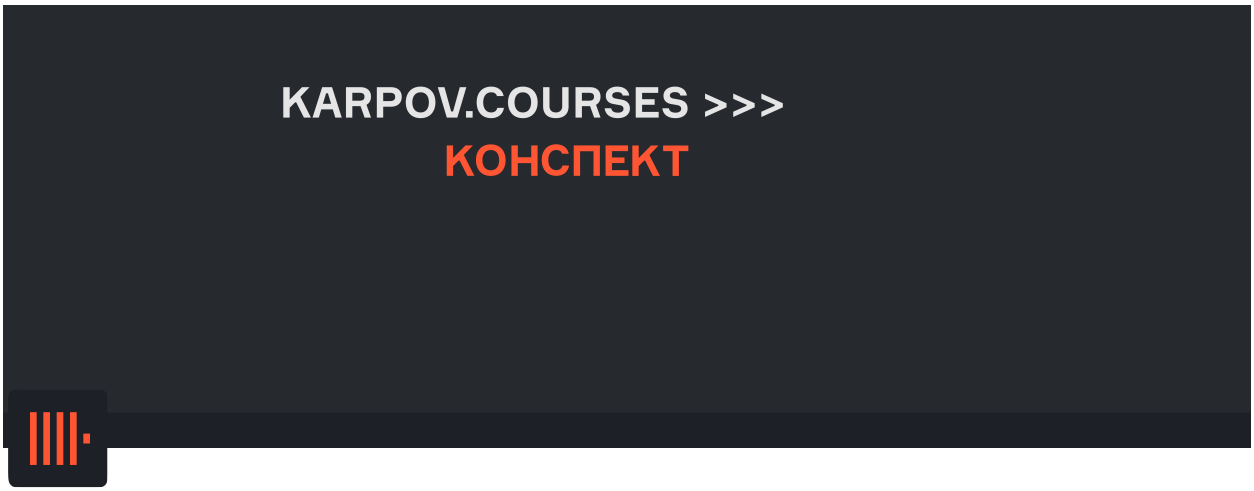



# > Конспект > 4 урок > SQL

## > Оглавление



## > Как зайти в redash?

1. Переходим на https://redash.lab.karpov.courses/
2. Заполняем поля
   **Логин:** ваш email
   **Пароль:** пароль от личного кабинета

Более детально изучим в следующих лекциях!

# > Int[8/16/32/64] / UInt[8/16/32/64]

Типы данных `Int` и `UInt` предназначены для хранения целых чисел фиксированной длины. [8/16/32/64] обозначают разрядность, чем она больше, тем больший диапазон значений может хранить.

Главное отличие между Int и Unit заключается в том, что первый может хранить как положительные, так и отрицательные значения.

## Диапазоны Int

- Int8 – [ -128 : 127 ]
- Int16 – [ -32768 : 32767 ]
- Int32 – [ -2147483648 : 2147483647 ]
- Int64 – [ -9223372036854775808 : 9223372036854775807 ]

```
SELECT
    CAST(-127 AS Int8) AS test,
    toTypeName(CAST(-127 AS Int8)) AS test_name
```

```
+------+-----------+
| test | test_name |
+------+-----------+
| -127 |      Int8 |
+------+-----------+
```

## Диапазоны UInt

- UInt8 – [ 0 : 255 ]
- UInt16 – [ 0 : 65535 ]
- UInt32 – [ 0 : 4294967295 ]
- UInt64 – [ 0 : 18446744073709551615 ]

```
SELECT
    CAST(127 AS UInt8) AS test,
    toTypeName(CAST(127 AS UInt8)) AS test_name
```

```
+------+-----------+
| test | test_name |
+------+-----------+
|  127 |      Int8 |
+------+-----------+
```

## > Булевы значения

Хранение булевых значений в ClickHouse реализовано через `UInt8`, отдельный тип данных для них не выделен. Сами значения такого формата принимают значение либо 1, либо 0.

```
SELECT
    3 = 3, -- сравниваем значения
    toTypeName(3=3) -- проверяем тип данных с помощью toTypeName()
```

```
+--------------+-------------------------+
| equals(3, 3) | toTypeName(equals(3, 3)) |
+--------------+-------------------------+
|            1 |                   UInt8 |
+--------------+-------------------------+
```

Аналогичный пример, когда мы ожидаем получить False (0):

```
SELECT
    3 = 4,
    toTypeName(3=4)
```

```
+--------------+-------------------------+
| equals(3, 4) | toTypeName(equals(3, 4)) |
+--------------+-------------------------+
|            0 |                   UInt8 |
+--------------+-------------------------+
```

## > Переполнение типов

Что же произойдет, если наши данные хранились с типом `Int8`, но максимальное значение увеличилось на единичку и превысило 127?

```
SELECT
    CAST(127 AS Int8) + 1 AS test,
    toTypeName(CAST(127 AS Int8) + 1) AS test_name
```

Тип данных автоматически увеличит битность (с 8 на 16):

```
+------+-----------+
| test | test_name |
+------+-----------+
```

```
|  128 |     Int16 |
+------+-----------+
```

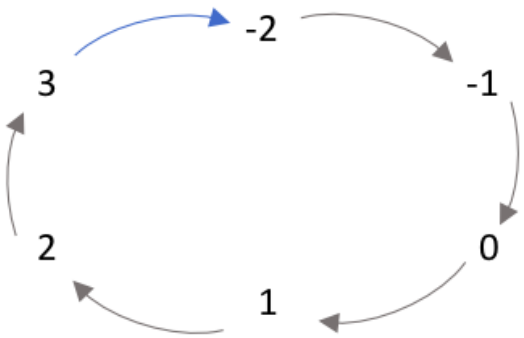
А если мы возьмем максимальное значение для `Int64`?

```
SELECT
    CAST(9223372036854775807 AS Int64) + 1 AS test,
    toTypeName(CAST(9223372036854775807 AS Int64) + 1) AS test_name
```

Откуда взялся минус? SQL попытался прибавить единичку, но вышел за границу, и, в некотором смысле пройдя по кругу, вернулся к начальной границе типа.

```
+----------------------+-----------+
|                 test | test_name |
+----------------------+-----------+
| -9223372036854775808 |     Int64 |
+----------------------+-----------+
```

Более наглядный пример: предположим у нас есть свой тип данных, принимающий всего 6 значений от -2 до 3. Когда мы прибавим к 3 единицу, тип переполнится, и значение станет -2. Если сделаем 3+2, то будет -1, и т.д.




# > Nullable(TypeName)

`Nullable(TypeName)` – позволяет вместе хранить значения как определенного типа (где TypeName – название нужного типа), так и NULL.

# > String

`String` – тип данных для хранения строк произвольной длины.

Иногда цифры или даты тоже могут быть записаны как текст. В таком случае можно привести значения к подходящему типу следующим образом:

```
SELECT -- число 22 записано как строка, поэтому приводим к Int8 -> можем произвести операцию сложения
CAST('22' AS Int8) + 17 AS sum_test
```

```
+----------+
| sum_test |
+----------+
|       39 |
+----------+
```

## replaceAll()

Представим ситуацию, когда сумма покупок записана в формате строки и со значком доллара. Для того, чтобы получить возможность применять агрегатные функции, нужно избавиться от лишнего, используя `replaceAll()`:

```
SELECT-- оборачиваем значение в replaceAll(), затем приводим к Float64
CAST(replaceAll('22.3$', '$', '') AS Float64) AS num_test
```

```
+----------+
| num_test |
+----------+
|     22.3 |
+----------+
```

# > Date / DateTime / Interval

`Date` – используется для хранения даты

```
2019-01-01
```

`DateTime` – тип данных для хранения даты и времени

```
2019-01-01 00:00:00
```

`Interval` – тип данных для интервалов дат и времени

```
SELECT
    INTERVAL 4 DAY as test,
    toTypeName(INTERVAL 4 DAY) as test_type
```

```
+------+-------------+
| test |   test_type |
+------+-------------+
|    4 | IntervalDay |
+------+-------------+
```

# > Float[32/64]

`Float` – числа с плавающей запятой. Их использование в ClickHouse не рекомендовано, т.к. можно столкнуться с ошибками округления. В некоторых случаях точность подсчетов может быть критична, например, при работе с большими денежными показателями.

## DECIMAL(P, S)

`DECIMAL` – тип данных, который используется для хранения дробных чисел и позволяет избежать проблем, которые возникают при использовании Float.

Параметры:

- P (precision, 0:32) – определяет, сколько десятичных знаков может содержать число, включая дробную часть
- S (scale, 0:P) – определяет, сколько десятичных знаков содержится в дробной части числа

Документация

# > Array

`Array` – массив.

```
SELECT
    array('one', 'two', 'three') AS test_array,
    toTypeName(test_array),
    test_array[1] AS first_element  -- индексация начинается с 1, доступ к элементу через квадратные скобки
```

```
+---------------+---------------------------+---------------+
|    test_array | toTypeName(array(1, 2, 3)) | first_element |
+---------------+---------------------------+---------------+
| one,two,three |              Array(String) |           one |
+---------------+---------------------------+---------------+
```

## groupArray()

`groupArray` – еще одна агрегатная функция.

Например: для каждого пользователя считаем сумму покупок (`sum_rub`), а в dates записываем массив из дат (`dates`), когда он совершал покупки

```
SELECT
    UserID,
    groupArray(BuyDate) AS dates,
    SUM(Rub) AS sum_rub
FROM
    checks
GROUP BY
    UserID
ORDER BY
    sum_rub DESC
LIMIT 1
```

```
+---------------------+------------------------------------------+---------+
|              UserID |                                    dates | sum_rub |
+---------------------+------------------------------------------+---------+
| 15605251414578190336 |['2019-06-08', '2019-06-04', '2019-06-03'] |  181146 |
+---------------------+------------------------------------------+---------+
```

# > orNull

Для обработки случаев, когда в данных могут встретиться `NULL`, используется дополнение `orNull`:

```
SELECT
    toInt64OrNull('123'),  -- к обычному toInt64 добавляется OrNull
    toInt8OrNull('123$')
```

```
+---------------------+---------------------+
| toInt64OrNull('123') | toInt8OrNull('123$') |
+---------------------+---------------------+
|                 123 |                NULL |
+---------------------+---------------------+
```

## coalesce

`coalesce` – слева-направо проверяет являются ли переданные аргументы `NULL` и возвращает первое не `NULL` значение

```
SELECT
    coalesce(toInt32OrNull('$435'), null, null, 33, null, 77) as first_non_null
```

```
+----------------+
| first_non_null |
+----------------+
|             33 |
+----------------+
```

# > geoDistance

`geoDistance` – функция для работы с координатами, рассчитывающая расстояние между двумя точками в метрах.

Параметры:

- `lon1Deg` — долгота первой точки в градусах
- `lat1Deg` — широта первой точки в градусах
- `lon2Deg` — долгота второй точки в градусах
- `lat2Deg` — широта второй точки в градусах

Все значения долготы должны быть в диапазоне [-180°, 180°], широты – [-90°, 90°].

Пример:

```
SELECT-- lon1Deg, lat1Deg, lon2Deg, lat2Deg
    geoDistance(0.1278, 51.5074, 30.3609, 59.9311) AS dist_m,
    dist_m / 1000 AS dist_km
```

```
+-----------+-------------+
|    dist_m |     dist_km |
+-----------+-------------+
| 2087583.4 | 2087.583375 |
+-----------+-------------+
```