Funciones AQL predefinidas

Tiempo estimado: 35min

Antes de describir detenidamente las consultas multicolección, vamos a hacer una pequeña parada para introducir en familia las funciones AQL soportadas por ArangoDB.

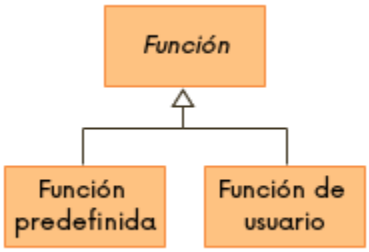
Comenzamos la lección introduciendo los dos tipos de función existentes: las funciones predefinidas y las funciones de usuario. A continuación, se presenta las principales funciones predefinidas que proporciona AQL.

Al finalizar la lección, el estudiante sabrá:

- Qué diferencia hay entre una función predefinida y una de usuario.
- Cuáles son las principales funciones predefinidas de AQL.

Introducción

ArangoDB permite que utilicemos funciones en AQL para realizar cálculos complejos, por ejemplo, en los comandos FILTER y UPDATE. Atendiendo a si vienen de fábrica con la instancia o las definimos nosotros mismos, distinguimos entre funciones predefinidas y funciones de usuario.



Una función predefinida (*built-in function*) es aquella que viene de fábrica con ArangoDB. Mientras que una función de usuario (*user function*), aquella que definimos nosotros, extendiendo así la funcionalidad de AQL.

En esta lección, prestamos atención a las predefinidas. En la siguiente, describiremos cómo definir nuestras propias funciones de usuario mediante JavaScript.

Invocación de función

Para invocar una función, basta con usar la misma sintaxis que en JavaScript:

```
función(argumento1, argumento2, argumento3…)
```

No se distingue entra mayúsculas y minúsculas en los nombres de función. Aunque por convenio y buenas prácticas, utilizaremos minúsculas.

Funciones de comprobación de tipo

Se puede consultar si un valor es de un tipo u otro mediante las siguientes funciones predefinidas:

```
is_null(value) : bool
is_bool(value) : bool
is_number(value) : bool
is_string(value) : bool
is_array(value) : bool
is_list(value) : bool
is_object(value) : bool
is_document(value) : bool
is_datestring(value) : bool
typename(value) : string
```

Parámetro	Tipo de datos	Descripción
value	any	Valor a comprobar.

Las funciones is_xxx() comprueban si un valor es de un tipo concreto. Mientras que typename() devuelve el tipo de un valor: null, bool, number, string, array u object. La función typename() es similar al operador typeof de JavaScript.

Funciones de conversión

Se puede convertir un valor de un tipo a otro, mediante las siguientes funciones predefinidas:

```
to_bool(value) : bool
to_number(value) : number
to_string(vlaue) : str
to_array(value) : array
to_list(value) : array
```

Parámetro	Tipo de datos	Descripción
value	any	Valor a convertir.

Funciones de cadena

AQL proporciona las siguientes funciones predefinidas para trabajar con cadenas de texto.

Longitud de cadena

Para conocer la longitud de una cadena, se puede utilizar:

```
length(value) : number
count(value) : number
```

Parámetro	Tipo de valor	Descripción
value	string	Texto a analizar.

Concatenación de cadenas

Para obtener la cadena que resulta de la concatenación de otras, se utiliza la función concat():

```
concat(value1, value2, value3…) : str
concat_separator(separator, value1, value2, value3…) : str
```

Parámetro	Tipo de datos	Descripción
separator	string	Texto a insertar entre las cadenas concatenadas.
valueX	string	Texto a concatenar.

En otros lenguajes de consulta, la concatenación se puede realizar mediante los operadores + o ||. En AQL, no es posible. Hay que utilizar la función concat().

La función concat() no añade ningún texto adicional entre las cadenas como separador. Si es necesario este separador, se puede utilizar la función concat_separator(). He aquí unos ejemplos ilustrativos:

```
127.0.0.1:8529@_system> db._query("RETURN concat('uno', 'dos', 'tres')").toArray()
[
  "unodostres"
]
127.0.0.1:8529@_system> db._query("RETURN concat_separator(':', 'uno', 'dos',
'tres')").toArray()
[
  "uno:dos:tres"
]
127.0.0.1:8529@_system>
```


Conversión a minúsculas y mayúsculas

Para obtener la cadena que resulta de convertir todas sus letras a minúsculas o mayúsculas, se utiliza las funciones lower() y upper(), respectivamente:

```
lower(value) : str
upper(value) : str
```

Parámetro	Tipo de datos	Descripción
value	string	Texto a convertir.

Omisión de caracteres iniciales y finales

Para obtener la cadena que resulta de suprimir los espacios en blanco iniciales y/o finales, se puede utilizar las siguientes funciones:

```
trim(value) : str
trim(value, chars) : str
ltrim(value) : str
ltrim(value, chars) : str
rtrim(value) : str
rtrim(value, chars) : str
```

Parámetro	Tipo de datos	Descripción
value	string	Texto a analizar.
chars	number	Caracteres a suprimir. Valor predeterminado: el espacio en blanco, el tabulador y el salto de línea.

trim() suprime tanto a izquierda como a derecha, o sea, al comienzo y al final de la cadena dada. En cambio, ltrim() sólo a la izquierda o inicio. Y rtrim() a derecha o final.

Subcadenas

Para obtener una subcadena, se puede utilizar:

```
substring(value, start) : str
substring(value, start, length) : str
left(value, length) : str
right(value, length) : str
```

Parámetro	Tipo de datos	Descripción
value	string	Texto a analizar.
start	number	Carácter en el que comenzar. El primer carácter es el cero.
length	number	Número de caracteres a recuperar.

La función substring() permite extraer un intervalo de caracteres indicando el primer carácter en el que comenzar a extraer. left() siempre comienza en el carácter cero. Y right() en el último.

Veamos unos ejemplos ilustrativos:

```
127.0.0.1:8529@_system> db._query("RETURN substring('an emotional fish',
3)").toArray()
[
  "emotional fish"
]
127.0.0.1:8529@_system> db._query("RETURN substring('an emotional fish', 3,
9)").toArray()
[
  "emotional"
]
127.0.0.1:8529@_system> db._query("RETURN left('an emotional fish', 12)").toArray()
[
  "an emotional"
```



```
]
127.0.0.1:8529@_system> db._query("RETURN right('an emotional fish', 14)").toArray()
[
  "emotional fish"
]
127.0.0.1:8529@_system>
```

Conversión de objetos JSON

Para obtener una cadena en formato JSON de un valor o viceversa, se puede utilizar:

```
json_parse(text) : object
json_stringify(value) : str
```

Parámetro	Tipo de datos	Descripción
text	string	Representación textual del valor JSON.
value	object	Valor JSON del que extraer su representación textual.

json_parse() es similar a la función JSON.parse() de JavaScript. Y json_stringify() a JSON.stringify().

Búsqueda de texto

Para comprobar si existe un texto dentro de otro, se puede usar las siguientes funciones:

```
contains(text, search) : bool
contains(text, search, returnIndex) : number
find_first(text, search) : number
find_first(text, search, start) : number
find_first(text, search, start, end) : number
find_last(text, search) : number
find_last(text, search, start) : number
find_last(text, search, start, end) : number
regex_test(text, search) : bool
regex_test(text, search, caseInsensitive) : bool
```

Parámetro	Tipo de datos	Descripción
text	string	Texto a analizar.
search	string	Texto a buscar.
start	number	Índice en el que comenzar la búsqueda.
end	number	Índice en el que finalizar la búsqueda.
caseInsensitive	bool	¿No distinguir entre mayúsculas y minúsculas?

La función contains() comienza la búsqueda al comienzo de la cadena y devuelve si el texto existe.

Inversión de texto

Para obtener una cadena en orden inverso:

```
reverse(value) : str
```

Obtención de huellas

Para obtener la huella de un texto, se puede utilizar las funciones md5() y sha1():

```
md5(text) : str
sha1(text) : str
```

Parámetro	Tipo de datos	Descripción
text	string	Texto del que obtener su huella.

División de texto

Para obtener un *array* con fragmentos de texto, usando un determinado separador de fragmento, podemos utilizar la función split():


```
split(value, separator) : str[]
split(value, separator, limit) : str[]
```

Parámetro	Tipo de datos	Descripción
value	string	Texto a analizar.
separator	string o string[]	Separador(es) a usar.
limit	number	Número máximo de fragmentos a devolver.

Unos ejemplos ilustrativos:

```
127.0.0.1:8529@_system> db._query("RETURN split('flesh for lulu', ' ')").toArray()
[
  [
    "flesh",
    "for",
    "lulu"
  ]
]
127.0.0.1:8529@_system> db._query("RETURN split('flesh for lulu', ' ', 2)").toArray()
[
  [
    "flesh",
    "for"
  ]
]
127.0.0.1:8529@_system>
```

Reemplazo de texto

Para obtener un texto tras reemplazar partes de él por otro, se puede utilizar la función substitute():

```
substitute(value, search, replace) : string
substitute(value, search, replace, limit) : str
```

Parámetro	Tipo de datos	Descripción
value	string	Texto a analizar.
search	string o string[]	Texto(s) a reemplazar.
replace	string	Texto de reemplazo.
limit	number	Número máximo de reemplazos.

Funciones numéricas

Para trabajar con números, se dispone de las siguientes funciones.

Valor absoluto

Para obtener el valor absoluto de un número, se puede utilizar la función abs():

```
abs(value) : number
```

Parámetro	Tipo de datos	Descripción
value	number	Número a analizar.

Mínimo, intermedio y máximo

Para obtener el valor mínimo, intermedio o máximo, podemos utilizar las siguientes funciones:

```
max(numArray) : number
median(numArray): number
min(numArray) : number
```

Parámetro	Tipo de datos	Descripción
numArray	number[]	Números a analizar.

Promedio

Para obtener el promedio de varios números, se puede utilizar:

```
average(numArray) : number
```

Parámetro	Tipo de datos	Descripción
numArray	number[]	Números a analizar.

Redondeo

Para redondear un número real, podemos utilizar las siguientes funciones:

```
ceil(value) : number
floor(value) : number
round(value) : number
```

Parámetro	Tipo de datos	Descripción
value	number	Número a analizar.

La función ceil() redondea al alza. floor() a la baja. Y round() al entero más próximo.

Veamos unos ejemplos ilustrativos:

```
127.0.0.1:8529@_system> db._query("RETURN ceil(2.5)").toArray()
[
  3
]
127.0.0.1:8529@_system> db._query("RETURN floor(2.5)").toArray()
[
  2
]
127.0.0.1:8529@_system> db._query("RETURN round(2.5)").toArray()
[
  3
]
127.0.0.1:8529@_system> db._query("RETURN round(2.4)").toArray()
[
  2
]
127.0.0.1:8529@_system>
```

Número aleatorio

Para obtener un número aleatorio, se puede utilizar:

```
rand() : number
```

Funciones varias

He aquí un conjunto de funciones varias:

```
sin(value) : number
cos(value) : number
tan(value) : number
acos(value) : number
asin(value) : number
atan(value) : number
atan2(y, x) : number
exp(value) : number
exp2(value) : number
log(value) : number
log2(value) : number
log10(value) : number
degrees(rad) : number
percentile(numArray, n, method) : number
pi() : number
pow(base, exp) : number
radians(deg) : number
range(start, stop) : number[]
```



```
range(start, stop, step) : number[]
sqrt(value) : number
stddev_population(nunmArray) : number
stddev_sample(numArray) : number
sum(numArray) : number
variance_population(numArray) : number
variance_sample(numArray) : number
```

Funciones de array

AQL también proporciona funciones para trabajar con *arrays*.

Tamaño de un array

Para conocer el tamaño de un *array*:

```
count(array) : number
length(array) : number
```

Parámetro	Tipo de datos	Descripción
array	array	*Array* a analizar.

Obtención de elementos

Para obtener el primer y el último elemento de un *array*, podemos usar:

```
first(array) : any
last(array) : any
```

Parámetro	Tipo de datos	Descripción
array	array	*Array* a analizar.

Para saber si un elemento se encuentra en el *array* u obtener un elemento a partir de su índice, se puede utilizar el método position():

```
position(array, search) : bool
position(array, search, returnIndex) : bool|number
```

Parámetro	Tipo de datos	Descripción
array	array	*Array* a analizar.
search	any	Elemento a buscar.
returnIndex	bool	¿Devolver índice donde se encuentra?

La primera sobrecarga devuelve si el elemento se encuentra en el *array*. Si deseamos saber en qué posición, hay que utilizar la segunda pasando true como tercer argumento. En este último caso, si el elemento no se encuentra en el *array*, devolverá -1.

Veamos unos ejemplos:

```
127.0.0.1:8529@_system> db._query("RETURN position(['uno', 'dos', 'tres'],
'dos')").toArray()
[
  true
]
127.0.0.1:8529@_system> db._query("RETURN position(['uno', 'dos', 'tres'],
'cuatro')").toArray()
[
  false
]
127.0.0.1:8529@_system> db._query("RETURN position(['uno', 'dos', 'tres'], 'dos',
true)").toArray()
[
  1
]
127.0.0.1:8529@_system> db._query("RETURN position(['uno', 'dos', 'tres'], 'cuatro',
true)").toArray()
```



```
[
  -1
]
127.0.0.1:8529@_system>
```

Para obtener el valor de un determinado elemento:

```
nth(array, position) : any
```

Parámetro	Tipo de datos	Descripción
array	array	*Array* con el que trabajar.
position	number	Índice del elemento.

Si lo que se necesita es obtener un fragmento de un *array*:

```
slice(array, start) : array
slice(array, start, length) : array
```

Parámetro	Tipo de datos	Descripción
array	array	*Array* con el que trabajar.
start	number	Índice del elemento en el que comenzar.
length	number	Número de elementos a obtener.

Añadidura de elementos

Para añadir uno o más elementos a un *array,* se puede usar las funciones siguientes:

```
append(array, value) : array
push(array, value) : array
unshift(array, value) : array
append(array, value, unique) : array
push(array, value, unique) : array
unshift(array, value, unique) : array
append(array, values) : array
push(array, values) : array
append(array, values, unique) : array
push(array, values, unique) : array
```

Parámetro	Tipo de datos	Descripción
array	array	*Array* con el que trabajar.
value	any	Elemento a añadir.
values	array	Elementos a añadir.
unique	bool	¿Añadir sólo si no existe ya?

Las funciones append() y push() añaden el elemento al final del *array,* mientras que unshift() al comienzo.

Ejemplos:

```
127.0.0.1:8529@_system> db._query("RETURN append([1, 3, 5], 2)").toArray()
[
  [
    1,
    3,
    5,
    2
  ]
]
127.0.0.1:8529@_system> db._query("RETURN append([1, 3, 5], [2, 4])").toArray()
[
  [
    1,
    3,
    5,
```

```
    2,
    4
  ]
]
127.0.0.1:8529@_system> db._query("RETURN append([1, 3, 5], [1, 2, 3, 4, 5, 6],
true)").toArray()
[
  [
    1,
    3,
    5,
    2,
    4,
    6
  ]
]
127.0.0.1:8529@_system> db._query("RETURN unshift([1, 3, 5], 2)").toArray()
[
  [
    2,
    1,
    3,
    5
  ]
]
127.0.0.1:8529@_system>
```

Supresión de elemento

Para suprimir el primer y el último elementos de un *array*, se puede utilizar las funciones shift() y pop(), respectivamente:

```
shift(array) : array
pop(array) : array
```

Parámetro	Tipo de datos	Descripción
array	array	*Array* con el que trabajar.

A continuación, unos ejemplos ilustrativos:

```
127.0.0.1:8529@_system> db._query("RETURN pop([1, 2, 3, 4, 5])").toArray()
[
  [
    1,
    2,
    3,
    4
  ]
]
127.0.0.1:8529@_system> db._query("RETURN shift([1, 2, 3, 4, 5])").toArray()
[
  [
    2,
    3,
    4,
    5
  ]
]
127.0.0.1:8529@_system>
```

Si el objetivo es suprimir un elemento de una determinada posición, se puede utilizar la función remove_nth():

```
remove_nth(array, position) : array
```

Parámetro	Tipo de datos	Descripción
array	array	*Array* con el que trabajar.
position	number	Índice del elemento a suprimir.

Las índices comienzan en cero y se puede utilizar tanto valores positivos como negativos, donde -1 hace referencia al último elemento; -2 al penúltimo; y así sucesivamente.

Para suprimir determinado(s) valor(es), se puede utilizar las siguientes funciones:

```
remove_value(array, value) : array
remove_value(array, value, limit) : array
remove_values(array, values) : array
```

Parámetro	Tipo de datos	Descripción
array	array	*Array* con el que trabajar.
value	any	Valor a suprimir.
values	array	Valores a suprimir.
limit	number	Número máximo de veces que debe suprimirse el valor.

Operaciones de conjuntos

La función unique() devuelve los valores únicos de un *array*:

```
unique(array) : array
```

Parámetro	Tipo de datos	Descripción
array	array	*Array* a analizar.

Para obtener la intersección, la unión y la diferencia de los elementos de varios *arrays*, hay que utilizar las funciones intersection(), minus() y union():

```
intersection(array1, array2, array3…) : array
minus(array1, array2, array3…) : array
union(array1, array2, array3…) : array
union_distinct(array1, array2, array3…) : array
```

Parámetro	Tipo de datos	Descripción
arrayX	array	*Array* con el que trabajar.

Las funciones intersection(), minus() y union_distinct() suprimen duplicados; union() no lo hace.

Para saber los elementos que se encuentran en todos los *arrays*, se usa:

```
outersection(array1, array2, array3…) : array
```

Parámetro	Tipo de datos	Descripción
arrayX	array	*Array* con el que trabajar.

Inversión de array

Para obtener un *array* con sus elementos en orden inverso, podemos usar reverse():

```
reverse(array) : array
```

Parámetro	Tipo de datos	Descripción
array	array	*Array* con el que trabajar.

Funciones de objeto

También existe funciones específicas para trabajar con objetos.

Obtención de nombres de campos

Para obtener los nombres de los campos de un documento:

```
attributes(doc) : str[]
attributes(doc, removeInternal) : str[]
attributes(doc, removeInternal, sort) : str[]
```


Parámetro	Tipo de datos	Descripción
doc	object	Documento a analizar.
removeInternal	bool	¿No devolver los campos de sistema _key, _id, etc.?
sort	bool	¿Devolver el *array* de nombres ordenado?

Para conocer el número de campos, se puede utilizar las funciones count() y length():

```
count(doc) : number
length(doc) : number
```

Parámetro	Tipo de datos	Descripción
doc	object	Documento a analizar.

Obtención de valores de campos

Si lo que se desea es obtener los valores de los campos, se puede utilizar:

```
values(doc) : array
values(doc, removeInternal) : array
```

Parámetro	Tipo de datos	Descripción
doc	object	Documento a analizar.
removeInternal	bool	¿No devolver los campos de sistema _key, _id, etc.?

Existencia de campo

Para comprobar si un documento tiene un determinado campo:

```
has(doc, attr) : bool
```

Parámetro	Tipo de datos	Descripción
doc	object	Documento a analizar.
attr	string	Nombre del campo a comprobar.

Supresión de campos

Para suprimir campos, se puede utilizar las funciones siguientes:

```
keep(doc, attr1, attr2, attr3…) : object
keep(doc, attrs) : object
unset(doc, attr1, attr2, attr3…) : object
unset(doc, attrs) : object
```

Parámetro	Tipo de datos	Descripción
doc	object	Documento con el que trabajar.
attrX	string	Nombre de campo a suprimir.
attrs	string[]	Nombres de los campos a suprimir.

La función keep() suprime todos los campos de un documento salvo los indicados. En cambio, unset() suprime los indicados.

Mezcla de documentos

Se puede obtener el documento que resulta de la mezcla de los campos de otros documentos mediante las siguientes funciones:

```
merge(doc1, doc2, doc3…) : object
merge_recursive(doc1, doc2, doc3…) : object
```

Parámetro	Tipo de datos	Descripción

docX	object	Documento con el que trabajar.

merge() es similar a la función Object.assign() de JavaScript.

Funciones de fecha

Para trabajar con fechas, se dispone de las siguientes funciones predefinidas. En ArangoDB, el tipo fecha no está soportado, pero se puede representar mediante su valor numérico.

Obtención de fecha

Para obtener la fecha actual, se utiliza la función date_now():

```
date_now() : number
```

Conversión de fechas

Para convertir fechas, se puede utilizar:

```
date_timestamp(iso8601) : number
date_timestamp(year, month, day, hh, mm, sec, ms) : number
date_iso8601(date) : str
```

Parámetro	Tipo de datos	Descripción
iso8601	str	Fecha en formato ISO 8601.
year	number	Año.
month	number	Mes.
day	number	Día del mes.
hh	number	Hora.
mm	number	Minuto.
sec	number	Segundo.
ms	number	Milisegundo.
date	number	Fecha en su representación numérica.

Formateo de fecha

Para obtener partes de una fecha, se puede utilizar las siguientes funciones:

```
date_dayofweek(date) : number
date_dayofyear(date) : number
date_year(date) : number
date_month(date) : number
date_day(date) : number
date_hour(date) : number
date_second(date) : number
date_millisecond(date) : number
date_isoweek(date) : number
date_leapyear(date) : bool
date_quarter(date) : number
date_days_in_month(date) : number
```

Parámetro	Tipo d datos	Descripción
date	number	Fecha a analizar.

Para formatear una fecha numérica en una cadena de texto particular, podemos utilizar la función date_format():

```
date_format(date, format) : str
```

Parámetro	Tipo de datos	Descripción
date	number	Fecha a formatear.

format	string	Formato.

Los marcadores de fecha que se pueden utilizar en el parámetro de formato son:

- %t. Formato numérico de la fecha.
- %z. La fecha en formato ISO 8601.
- %w. Día de la semana.
- %y. Año.
- %yy. Año con dos dígitos.
- %yyyy. Año con cuatro dígitos.
- %m. Mes.
- %mm. Mes con dos dígitos.
- %d. Día del mes.
- %dd. Día del mes con dos dígitos.
- %h. Hora.
- %hh. Hora con dos dígitos.
- %i. Minuto.
- %ii. Minuto con dos dígitos.
- %s. Segundo.
- %ss. Segundo con dos dígitos.
- %f. Milisegundo.
- %fff. Milisegundo con tres dígitos.
- %x. Día del año.
- %xxx. Día del año con tres dígitos.
- %k. Número de semana.
- %kk. Número de semana con dos dígitos.
- %l. Si año bisiesto, 1; en otro caso, 0.
- %q. Trimestre.
- %a. Días que tiene el mes de la fecha.
- %mmm. Mes con tres letras.
- %mmmm. Mes con todas sus letras.
- %www. Día de la semana con tres letras.
- %wwww. Día de la semana con todas sus letras.
- %%. El carácter %.

Modificación de fecha

Para modificar una fecha, podemos utilizar:

```
date_add(date, amount, unit) : str
date_subtract(date, amount, unit) : str
```

Parámetro	Tipo de datos	Descripción
date	number	Fecha con la que trabajar.
amount	number	Unidades a sumar o restar.

unit	string	Medida de suma o resta: • y, años. • m, meses. • w, semanas. • d, días. • h, horas. • i, minutos. • s, segundos. • f, milisegundos.

date_add() se utiliza principalmente para obtener fechas del futuro; mientras que date_subtract(), del pasado. Ambas funciones devuelven la fecha resultante en formato ISO 8601.

Veamos cómo calcular la fecha de hace dos meses y de dentro de dos meses:

```
127.0.0.1:8529@_system> db._query("RETURN date_format(date_now(), '%z')").toArray()
[
  "2017-01-07T09:52:01.035Z"
]
127.0.0.1:8529@_system> db._query("RETURN date_add(date_now(), 2, 'm')").toArray()
[
  "2017-03-07T09:52:30.364Z"
]
127.0.0.1:8529@_system> db._query("RETURN date_subtract(date_now(), 2,
'm')").toArray()
[
  "2016-11-07T09:54:24.780Z"
]
127.0.0.1:8529@_system>
```

Diferencia entre fechas

La función date_diff() devuelve el tiempo que hay entre dos fechas dadas:

```
date_diff(date1, date2, unit) : number
date_diff(date1, date2, unit, asFloat) : number
```

Parámetro	Tipo de datos	Descripción
date1	number	Primera fecha.
date2	number	Segunda fecha.
unit	string	Unidad de diferencia: y, m, w, d, h, i, s o f.
asFloat	bool	¿Devolver un valor real en vez de entero?

A continuación, se muestra cómo calcular cuántos días hay entre dos fechas:

```
127.0.0.1:8529@_system> db._query("RETURN date_diff(date_now(), 1613383402973,
'd')").toArray()
[
  1499
]
127.0.0.1:8529@_system> db._query("RETURN date_diff(date_now(), 1613383402973, 'd',
true)").toArray()
[
  1499.9984386921296
]
127.0.0.1:8529@_system>
```

Funciones de nulo

Mediante la función not_null(), se puede obtener el primer parámetro cuyo valor es distinto de null:

```
not_null(value1, value2, value3…) : any
```

Parámetro	Tipo de datos	Descripción
valueX	any	Valor a comprobar.

Es similar a la función coalesce() de SQL

