Operadores AQL

Tiempo estimado: 15min

En la lección anterior, presentamos las consultas de selección, pero no hicimos nada de hincapié en los operadores soportados por el lenguaje AQL. Ha llegado el momento de hacerlo.

La lección comienza introduciendo los conceptos de expresión, operador, operando, aridad, precedencia y asociatividad. Y a continuación, se van presentando los distintos operadores de AQL en grupos.

Al finalizar la lección, el estudiante sabrá:

- Qué es un operador.
- Qué es la aridad.
- Qué es la precedencia y cuál es la tabla de precedencia de AQL.
- Qué operadores se pueden utilizar en las expresiones de AQL.

Introducción

Una expresión (*expression*) es una proposición formada por una combinación de términos que, tras ser evaluada, devuelve un valor. Principalmente, están formadas por identificadores, operadores, valores literales y/o subexpresiones como, por ejemplo, `1 - 2 + 3`.

Un operador (*operator*) es un símbolo o palabra reservada que realiza una determinada operación como, por ejemplo, la suma, la multiplicación o la división. Los parámetros de los operadores se conocen formalmente como operandos (*operands*), los cuales pueden ser valores literales, variables o subexpresiones. AQL proporciona un conjunto muy rico de operadores para facilitar las consultas, tanto de lectura como de escritura.

Todo operador tiene una aridad, una precedencia y una asociatividad.

La aridad (*arity*) indica el número de operandos que espera el operador. Teniendo en cuenta este número, se pueden clasificar en: operadores unarios (*unary operators*), un único operando; operadores binarios (*binary operators*), dos operandos; operadores ternarios (*ternary operators*), tres operandos; y operadores n-arios (*n-ary operators*), *n* operandos. Salvando algunas excepciones, tal como veremos en su momento, las sintaxis de los operadores unarios y binarios, respectivamente, son como sigue:

```
operador operando
operando operador operando
```

Como una expresión puede contener varios operadores, es necesario que el motor de AQL pueda determinar, sin temor a equivocarse, el orden en que debe ejecutarlos. No puede ejecutarlos aleatoriamente, porque el orden de los factores puede alterar el producto. Por esta razón, los operadores presentan una precedencia (*precedence*), esto es, una prioridad del operador con respecto a los demás dentro de la misma expresión. Así, los operadores con mayor precedencia se ejecutarán antes que los de menor precedencia.

La siguiente tabla indica la precedencia de los operadores de AQL, de *mayor a menor precedencia*, junto a su aridad y asociatividad:

Precedencia	Operador	Aridad	Asociatividad
1	[]	Binario	Por la izquierda
2	.	Binario	Por la izquierda
3	fn()	Binario	Por la izquierda
4	[*]	Binario	Por la izquierda
5	!, not, +, -	Unario	Por la derecha

6	*, /, %	Binario	Por la izquierda
7	+, -	Binario	Por la izquierda
8	<, <=, >, >=	Binario	Por la izquierda
9	in, not in	Binario	Por la izquierda
10	==, !=	Binario	Por la izquierda
11	&&, and	Binario	Por la izquierda
12	||, or	Binario	Por la izquierda
13	?:	Ternario	Por la derecha

Cuando deseamos dejar claro cuál es el orden, se suele utilizar los paréntesis para delimitar subexpresiones que marcan que su contenido debe ejecutarse primero. Al igual que se hace en JavaScript y en otros lenguajes de consulta como, por ejemplo, SQL.

Operadores de comparación

Un operador de comparación (*comparison operator*) es aquel que compara dos operandos entre sí para comprobar si son iguales o distintos, devolviendo un valor booleano como resultado. La siguiente tabla enumera los operadores de comparación de AQL, muy similares a los de JavaScript:

Operador	Aridad	Descripción	Ejemplo
==	Binario	Igualdad	x == y
!=	Binario	Desigualdad	x != y
<	Binario	Menor que	x < y
<=	Binario	Menor o igual que	x <= y
>	Binario	Mayor que	x > y
>=	Binario	Mayor o igual que	x >= y

Operadores de patrones

Un patrón (*pattern*) es un texto que define la forma de algo. En AQL, se puede utilizar los siguientes operadores de comparación de patrón:

Operador	Aridad	Descripción	Ejemplo
like	Binario	Comparación de cumplimiento de patrón simple	valor like patrón
=~	Binario	Comparación de cumplimiento con expresión regular	valor =~ expreg
!~	Binario	Comparación de no cumplimiento con expresión regular	valor !~ expreg

El operador like es similar al de SQL, utiliza los siguientes marcadores o comodines:

- % para representar cualquier número de caracteres arbitrario.
- _ para representar un carácter cualquiera.

Por su parte, los patrones de expresiones regulares son más completos:

- . representa una carácter cualquiera.
- \d representa un dígito cualquiera.
- \s representa un carácter de espacio en blanco.
- \S representa un carácter que no sea el espacio en blanco.
- \t representa el carácter de tabulado.
- \r representa el carácter de retorno de carro.
- \n representa el carácter de nueva línea.
- [abc] representa uno de los caracteres indicados entre los corchetes.

- [^abc] representa un carácter que no sea uno de los indicados entre los corchetes.
- [a-z] representa un carácter en el rango indicado entre los corchetes.
- [^a-z] representa un carácter fuera del rango indicado entre los corchetes.
- (a|b) representa una de las opciones indicadas entre los paréntesis.
- ^ representa el inicio de la cadena.
- $ representa el fin de la cadena.

He aquí unos ejemplos ilustrativos:

```
b.origin like "%, UK"
b.origin =~ ".*, UK$"
```

Operadores de array

En las restricciones, es muy común comparar si un determinado valor se encuentra en un *array*. Para este fin, se puede utilizar el operador binario in:

```
valor in array
valor not in array
```

Ejemplo:

```
"indie" in b.genres
```

También es posible comparar *arrays* con *arrays* con los siguientes operadores binarios:

Operador	Ejemplo	Descripción
array all in array	b.tags all in ["indie", "folk"]	Todos los elementos del *array* izquierdo deben encontrarse en el derecho.
array none in array	b.tags none in ["indie", "folk"]	Ninguno de los elementos del *array* izquierdo debe encontrarse en el derecho.
array any in array	b.tags any in ["indie", "folk"]	Alguno de los elementos del *array* izquierdo debe encontrarse en el derecho.
array any == valor array any != valor array any > valor array any >= valor array any < valor array any <= valor	b.tags == "indie" b.tags != "indie" b.tags > "indie" b.tags >= "indie" b.tags < "indie" b.tags <= "indie"	Alguno de los elementos del *array* debe ser igual, distinto, menor, menor o igual que, mayor o mayor o igual que el indicado.
array none == valor array none != valor array none > valor array none >= valor array none < valor array none <= valor	b.tags none == "indie" b.tags none != "indie" b.tags none > "indie" b.tags none >= "indie" b.tags none < "indie" b.tags none <= "indie"	Ninguno de los elementos del *array* debe ser igual, distinto, mayor, mayor o igual que, menor o menor o igual que el indicado.
array all == valor array all != valor array all > valor array all >= valor array all < valor array all <= valor	b.tags all == "indie" b.tags all != "indie" b.tags all > "indie" b.tags all >= "indie" b.tags all < "indie" b.tags all <= "indie"	Todos los elementos del *array* deben ser igual, distinto, mayor, mayor o igual que, menor o menor o igual que el indicado.

Operador de expansión de array

El operador de expansión de array (*array expansion operator*) itera por todos los elementos de un *array* y devuelve un *array* con los valores de un determinado campo o propiedad de cada elemento iterado. Su sintaxis es como sigue:

```
array[*].miembro
```

Supongamos que tenemos el siguiente *array*:


```
bands = [
  {name: "Bell X1", origin: "Ireland"},
  {name: "Blur", origin: "England"},
  {name: "The Boxer Rebellion", origin: "England"}
]
```
La expresión bands[*].name devolverá ["Bell X1", "Blur", "The Boxer Rebellion"].

Operadores lógicos

Un operador lógico (*logical operator*) devuelve un resultado booleano para las operaciones *AND*, *OR* y *NOT* lógicas. Son muy utilizados en las expresiones de restricción o filtro:

Operador	Aridad	Descripción	Ejemplo
&&, and	Binario	AND lógico	x and y
||, or	Binario	OR lógico	x or y
!, not	Unario	NOT lógico	not x

Los operadores binarios lógicos son en corto circuito (*short-circuit*), esto significa que, si a partir del operando de la izquierda pueden deducir su resultado, no evaluarán el segundo. Esto es muy importante. Por ejemplo, el operador && devolverá true sólo si ambos operandos son true. Si el primer operando *no* devuelve true, no hace falta evaluar el segundo, pues no puede devolver true bajo ningún concepto. Por su parte, el operador || devolverá true si al menos uno de los dos operandos es true. Si el primero lo es, no hace falta evaluar el segundo, pues va a devolver true sí o sí; en cambio, si devuelve un valor distinto de true, tiene que evaluar el segundo para determinar el valor final.

Operadores aritméticos

Los operadores aritméticos (*arithmetic operators*) son aquellos que realizan operaciones aritméticas con números como, por ejemplo, la suma, la resta o la multiplicación. Según el operador, puede ser unario o binario. La siguiente tabla presenta los operadores aritméticos de AQL:

Operador	Aridad	Descripción	Ejemplos
+	Unario	Conversión a número.	+x
-	Unario	Negación numérica.	-x
+	Binario	Suma.	x + y
-	Binario	Resta.	x - y
*	Binario	Multiplicación.	x * y
/	Binario	División.	x / y
%	Binario	Módulo, resto de la división.	x % y

Operador .?

El operador ternario .?, también conocido como operador condicional (*conditional operator*), evalúa el primer operando y si es verdadero, true, devuelve el segundo; en otro caso, el tercero. Su comportamiento y sintaxis es similar al de JavaScript:

```
condición .? siCierto : siFalso
```

Operador de rango

El operador de rango (*range operator*) devuelve un *array* con un elemento para cada número entero indicado por un rango, ambos extremos incluidos:

```
inicio..fin
```
Así pues, la expresión 3..7 devolverá [3, 4, 5, 6, 7].

