Selección simple de documentos

Tiempo estimado: 25min

Ya sabemos insertar documentos en las colecciones, así pues ha llegado el momento de ver cómo extraerlos de ellas.

La lección comienza introduciendo el concepto de consulta de selección y haciendo hincapié en la secuencia de comandos que forman una consulta en AQL. A continuación, se presenta cómo llevar a cabo las operaciones de extracción, proyección, restricción, ordenación, limitación y supresión mediante la API simple y AQL. Después, se examina cómo soporta ArangoDB los almacenes clave-valor. Finalmente, se presenta detalladamente los cursores con los que acceder a los resultados devueltos por la API simple y las consultas AQL.

Al finalizar la lección, el estudiante sabrá:

- Cómo realizar consultas de selección mediante la API simple y AQL.
- Qué es un cursor.
- Cómo recorrer un cursor.

Introducción

Una consulta de selección (*select query* o *find query*) es aquella que devuelve al usuario documentos de una o más colecciones. Es posiblemente la operación más realizada en una base de datos. Al conjunto de documentos devuelto por una consulta, se le conoce formalmente como conjunto resultado (*result set*).

En ArangoDB, se puede realizar este tipo de consultas mediante la API simple, recordemos, mediante los métodos del objeto colección, o bien mediante AQL. Cuando la consulta es sencilla, en muchas ocasiones la API simple permite la consulta sin problemas. Pero cuando la consulta es compleja o empieza a complicarse, por ejemplo, porque tiene restricciones, aplica funciones, etc., hay que utilizar AQL, porque es muy, pero que muy potente en este aspecto.

Consultas de selección AQL

Antes de adentrarnos más a fondo en las consultas de selección en AQL, hay que hacer un pequeño parón para explicar cómo funcionan. Concretamente, una consulta AQL, independientemente de su tipo, es una secuencia de comandos que realiza una determinada operación para el usuario. Ya hemos visto el comando INSERT, mediante el cual se inserta un nuevo documento en una colección. Ahora vamos a ver otros como FOR, FILTER, LIMIT y SORT, orientados principalmente a extraer y procesar documentos de colecciones.

Así pues, una consulta de selección está formada por comandos. Existe uno, FOR, que tiene como objeto extraer los documentos de una colección y añadirlos al flujo de trabajo. Existe otro, RETURN, que tiene como objeto marcar lo que se devolverá al usuario. Y existe otros intermedios que trabajarán con el flujo para restringir lo que le llegará al RETURN.

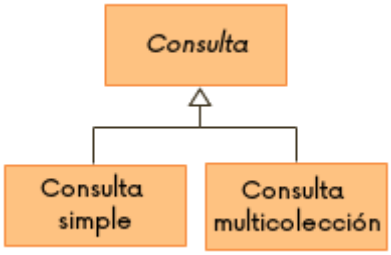
Hay que ver una consulta AQL como una secuencia de comandos que se van aplicando uno detrás de otro. Trabajan con los documentos que le llegan. Veamos un ejemplo ilustrativo:

```
FOR b in bands
FILTER b.origin == "Spain"
LIMIT 10
RETURN b
```

Esta consulta lo que hace es extraer los documentos de la colección bands cuyo campo origin sea Spain y devuelve los 10 primeros documentos tal cual se encuentran en la colección.

Extracción

La extracción (*extraction*) es la operación mediante la cual se sacan documentos de una colección para introducirlos en el flujo de trabajo o procesamiento de la consulta. Atendiendo al número de colecciones con las que se trabaja, se distingue entre consultas simples o multicolección.



Una consulta simple (*simple query*) es aquella cuyo origen de datos es una única colección. La consulta trabaja con documentos procedentes de una única colección de la base de datos. En cambio, si la consulta trabaja con documentos de varias colecciones, se habla de consulta multicolección (*multicollection query*). En esta lección, se habla únicamente de consultas simples, dejándose para una posterior las consultas multicolección.

Extracción mediante API simple

Un objeto colección proporciona el método all() para extraer todos los documentos de la colección:

```
all() : Cursor
```

Este método devuelve un cursor con el que recorrer los documentos de la colección.

He aquí un ejemplo ilustrativo:

```
db.bands.all()
```

También se puede utilizar el método iterate() de las colecciones, muy similar al método forEach() de los *arrays* de JavaScript:

```
iterate(fn)
```

Parámetro	Tipo de datos	Descripción
fn	function	Función de procesamiento de documentos: fn(doc, index).

El método tiene como objeto proporcionar cada documento individualmente a la función pasada como argumento. El parámetro doc representa el documento en curso; e index, el índice del documento. Ejemplo:

```
db.bands.iterate(function(doc) {
  print(doc.name);
})
```

Extracción de documentos mediante AQL

AQL es una mezcla entre SQL y JavaScript. Concretamente, funciona bajo un sistema similar a los bucles for-of de JavaScript. Para extraer documentos de una colección, en SQL se utiliza la cláusula FROM de una consulta SELECT. En AQL, se utiliza el comando FOR:

```
FOR variableIteradora IN colección
```

Este comando lo que hace es extraer todos los documentos de la colección indicada y los añade al flujo de trabajo, uno a uno. Para permitir trabajar con cada documento individualmente, proporciona la variable iteradora (*iterator variable*). Esta variable lo que hace es proporcionar un nombre con el que referenciar el documento actual, la cual se puede referenciar en los demás comandos de la consulta, por ejemplo, para hacer filtros, ordenar el resultado o proyectar campos.

Realmente, el comando espera el nombre de una colección de la base de datos o un valor de tipo *array*. En nuestro caso, vamos a centrarnos en las colecciones.

Toda consulta debe devolver un resultado. En SQL, el resultado queda implícito en la consulta. Estará formado por toda fila que cumpla lo indicado. En AQL, las cosas no son iguales. Hay que indicar explícitamente un comando RETURN, al final de la consulta, que indique lo que hay que devolver.

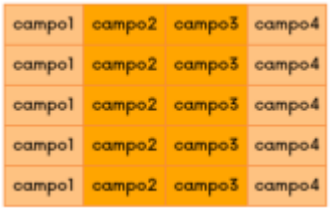
Generalmente, la variable iteradora, pero como veremos en breve, se puede restringir los campos a devolver.

Así pues, si deseamos acceder a todos los documentos de la colección bands mediante AQL, tendremos que hacerlo como sigue:

```
FOR b IN bands
RETURN b
```

Proyección

La proyección (*projection*) es la operación mediante la cual se indica qué campos queremos que presenten los documentos del conjunto resultado. Esta operación recibe como operando un conjunto de documentos y devuelve otro que contiene todos los documentos operandos, pero sólo con los campos especificados, suprimiendo del resultado todos aquellos que no se especifican explícitamente.



Proyección mediante API simple

En la API simple, se proyectan siempre todos los campos.

Proyección mediante AQL

En AQL, los campos a proyectar se especifican en el comando RETURN de la consulta. Podemos proyectar todos los campos o sólo algunos de ellos o incluso añadir otros generados dinámicamente.

Proyección de todos los campos

Si se desea proyectar todos los campos, no hay más que indicar la variable de iteración:

```
RETURN variable
```

Ejemplo:

```
FOR b IN bands
RETURN b
```

Proyección de columna específica

Para proyectar determinados campos o bien generar dinámicamente otros, hay que indicar el esquema de documento a devolver con una sintaxis similar a la siguiente:

```
RETURN {
  campo: valor,
  campo: valor,
  campo: valor…
}
```

El siguiente ejemplo ayudará a esclarecer las cosas:

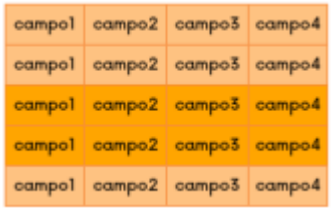
```
FOR b in bands
RETURN {name: b.name, origin: b.origin}
```

Es posible indicar como nombre de campo una expresión, generando así lo que se conoce como nombre de campo dinámico (*dynamic field name*). En este caso, al igual que en JavaScript, hay que delimitar la expresión entre corchetes. Ejemplo:

```
FOR b IN bands
RETURN {[b._key]: b.name}
```

Restricción

Mediante una restricción (*restriction*), se puede reducir los documentos que formarán parte del resultado final, a aquellos que cumplan una determinada condición.



Restricción mediante API simple

Cuando usamos la API simple, podemos extraer aquellos documentos de la colección que cumplan una determinada condición. En este caso, se utiliza el método byExample() del objeto colección:

```
byExample(exmaple) : Cursor
```

Parámetro	Tipo de datos	Descripción
example	object	Condición de filtro.

Por ejemplo, para extraer aquellos documentos cuyo campo origin sea Spain, podemos realizar una consulta como la siguiente:

```
db.bands.byExample({origin: "Spain"})
```

La restricción puede indicar tantos campos como sea necesario. Todos ellos se compararán por igualdad (operador ==).

Si necesitamos comparar la propiedad de un campo de tipo objeto, podemos hacer algo como:

```
db.bands.byExample({origin: {town: "Valencia"}})
```

Restricción mediante AQL

En AQL, para indicar una restricción, se utiliza el comando FILTER. Su sintaxis es la siguiente:

```
FILTER condición
```

Ejemplo ilustrativo:

```
FOR b IN bands
FILTER b.name == "The National"
RETURN b
```

En una consulta AQL, se puede especificar tantos FILTERs como sea necesario. Y además, podemos hacerlo en cualquier punto de la consulta. Veamos un ejemplo:

```
FOR b IN bands
FILTER b.origin.country == "Spain"
LIMIT 10
FILTER b.origin.town == "Valencia"
RETURN b
```

Primero se filtra por país. A continuación, se extrae los 10 primeros documentos que cumplen la restricción anterior. Y a continuación, se vuelve a hacer otro filtro que restringe el resultado, de los 10 seleccionados, a aquellos cuya ciudad sea Valencia.

El orden de los factores puede alterar el producto. Un filtro se aplica a los documentos que le llegan, los cuales dependerán de los comandos que le preceden. Así pues, el ejemplo anterior no tiene por qué devolver lo mismo que la siguiente consulta:

```
FOR b IN bands
FILTER b.origin.country == "Spain"
FILTER b.origin.town == "Valencia"
LIMIT 10
RETURN b
```

Ni que esta otra:

```
FOR b IN bands
LIMIT 10
FILTER b.origin.country == "Spain"
FILTER b.origin.town == "Valencia"
RETURN b
```

La flexibilidad de AQL a este respecto es total. Mucho más que SQL.

Ordenación

La ordenación (*sorting*) se encarga de poner en orden los documentos. Esta operación recibe un conjunto de documentos como operando y devuelve otro como resultado, pero con los documentos ordenados según se le haya indicado.

Ordenación mediante API simple

Actualmente, no se puede ordenar los documentos mediante la API simple.

Ordenación de documentos mediante AQL

Para ordenar los documentos del flujo de trabajo de una consulta AQL, antes de su paso por el comando RETURN, se puede utilizar el comando SORT, cuya sintaxis es la siguiente:

```
SORT expresión
SORT expresión modificador
```

La ordenación puede ser de dos tipos: ascendente o descendente. Mediante la ordenación ascendente (*ascendent sort*), se ordenan los documentos según el campo especificado de menor a mayor valor. Se especifica mediante el modificador ASC, o bien la omisión de modificador. Es el comportamiento predeterminado cuando no se indica nada. En cambio, para ordenar de mayor a menor valor, es decir, para llevar a cabo una ordenación descendente (*descendent sort*), hay que utilizar explícitamente el modificador DESC.

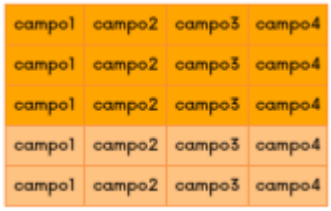
He aquí un ejemplo ilustrativo:

```
FOR b IN bands
FILTER b.origin == "UK"
SORT b.name ASC
RETURN b
```

Si lo necesitamos, podemos indicar varios campos de ordenación. Basta con separarlos unos de otros por comas.

Limitación de documentos

La limitación (*limit*) restringe el número máximo de documentos que pasan a través suyo. Todos aquellos que se encuentren más allá del límite indicado no se devolverán. Esta operación recibe como operando los documentos que han pasado los comandos anteriores y devuelve otro como resultado, pero sin los documentos de más.



Limitación mediante API simple

Para limitar el resultado de un método de consulta, se puede utilizar el método limit() del cursor devuelto. Veamos un ejemplo:

```
db.bands.all().limit(10).toArray()
```

Limitación mediante AQL

El comando LIMIT establece un tope de documentos que puede pasar a través suyo:

```
LIMIT count
LIMIT offset, count
```

De manera predeterminada, se indica el número máximo de documentos que devolverá. Mientras que offset indica a partir de qué documento comenzar a devolver, omitiéndose todos los anteriores de su resultado; si no se indica, comenzará en cero.

Ejemplos:


```
FOR b IN bands
LIMIT 10
RETURN b

FOR b IN bands
LIMIT 5, 10
RETURN b
```

En ambos casos, el comando devolverá como *máximo* 10 documentos. En el primero, devolverá los 10 primeros. En el segundo, los 10 que siguen al quinto.

Supresión de duplicados

Mediante la supresión de duplicados (*distinct*), se suprime del resultado aquellos documentos que son duplicados de otro. Así, nunca se obtendrá el mismo documento dos o más veces. Esta operación recibe cero, uno o más documentos como operando y devuelve otro conjunto, pero sin los duplicados.

Supresión de duplicados mediante API simple

No se puede suprimir duplicados mediante la API simple.

Supresión de duplicados mediante AQL

Para indicar la supresión de duplicados en una consulta AQL, se utiliza la cláusula DISTINCT del comando RETURN:

```
RETURN DISTINCT expresión
```

Ejemplos:

```
FOR b IN bands
RETURN DISTINCT b.origin
```

Almacenes clave-valor

¿Recuerda que ArangoDB es multimodelo? ¿Recuerda también que dijimos que soportaba tres modelos: clave-valor, documentos y grafos? Hasta el momento, hemos presentado una colección como un almacén de documentos. Veamos por qué se habla también de almacén clave-valor.

Un almacén clave-valor (*key-value store*) es un tipo de sistema NoSQL que almacena datos en forma de tabla o *array* asociativo. Los datos se agrupan en pares, donde un par (*pair*) no es más que un objeto o registro de datos formado por dos elementos: una clave y un valor. La clave (*key*) representa el identificador a utilizar para acceder al objeto o registro. Y el valor (*value*) contiene los datos.

La clave suele ser de tipo texto, pero el valor puede ser de cualquier otro tipo como, por ejemplo, un número, un objeto, un *array* u otro texto.

Decimos que ArangoDB soporta el modelo clave-valor porque los documentos se identifican *siempre* de manera única mediante una clave. Esta clave puede ser el campo _key, si deseamos identificar el documento de manera única en una colección. Siendo posible que dos o más documentos tengan el mismo valor _key, pero siempre en colecciones distintas. También es posible utilizar el campo _id, en cuyo caso se identifica el documento de manera única en toda la base de datos. Nunca dos o más documentos tendrán el mismo _id en la misma base de datos.

Debido a esto, se puede utilizar una colección como un almacén clave-valor. Cada documento representa un par, donde la clave es el campo clave que deseemos usar, _key o _id. Y el valor, el documento al completo.

Acceso mediante clave

Si necesitamos acceder a un documento mediante una de sus claves, podemos utilizar los métodos siguientes de la API simple:


```
db._document(id)
db.colección.document(key)
```

Parámetro	Tipo de datos	Descripción
id	string u object	Identificador del documento a acceder.
key	string u object	Clave del documento a acceder.

También es posible utilizar el método documents() de las colecciones:

```
db.colección.documents(keys)
```

Parámetro	Tipo de datos	Descripción
keys	object[]	Claves de los documentos a acceder.

En AQL, podemos usar una consulta con FILTER o bien la función document():

```
document(collection, key) : object
```

Parámetro	Tipo de datos	Descripción
collection	string	Nombre de la colección a consultar.
key	string	Clave a consultar.

Ejemplos:

```
FOR b IN bands
FILTER b._key == "athlete"
RETURN b

RETURN document("bands", "athlete")
```

Comprobación de clave

Para comprobar si una clave existe, se puede utilizar los métodos siguientes de la API simple:

```
db._exists(id) : boolean
db.colección.exists(key) : boolean
```

Parámetro	Tipo de datos	Descripción
id	string u object	Identificador del documento a consultar.
key	string u object	Clave del documento a consultar.

Número de documentos

Se puede consultar el número de documentos que tiene una colección mediante la API simple y AQL.

Número de documentos mediante API simple

Si usamos la API simple, se puede utilizar el método count():

```
count() : number
```

Ejemplo:

```
db.bands.count()
```

Número de documento mediante AQL

En el caso de AQL, se puede utilizar las funciones count() y length():

```
count(collection) : number
length(collection) : number
```

Parámetro	Tipo de datos	Descripción
collection	ArangoCollection	Colección a consultar.

Ejemplo ilustrativo:

```
RETURN count(bands)
```


Cursores

Los métodos de consulta de la API simple suelen devolver un cursor y el método _query() con el que ejecutar las consultas AQL también. Aunque cada uno lo devuelve de un tipo distinto.

Un cursor es un objeto para recorrer los documentos del resultado de una consulta. Ayuda a recorrer los resultados de manera más óptima, sobre todo cuando puede tener muchos documentos, pues no los solicita todos de golpe, sino que los va solicitando poco a poco, reduciéndose así el consumo de recursos en el servidor y en el cliente. Se utiliza en muchos sistemas de gestión de bases de datos como, por ejemplo, ArangoDB, MongoDB, PostgreSQL y SQL Server.

Existe algunos métodos muy útiles en los objetos de tipo cursor. Por un lado, tenemos el método toArray() que devuelve todo el contenido del cursor, o sea, de la consulta, en forma de *array*:

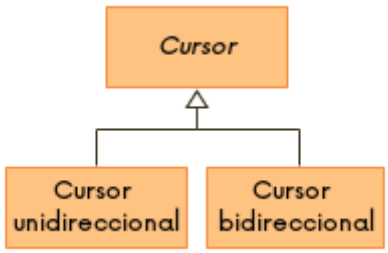
```
toArray() : object[]
```

Ejemplo:

```
127.0.0.1:8529@prueba> db.bands.all()
SimpleQueryAll(bands)
127.0.0.1:8529@prueba> db.bands.all().toArray()
[
  {
    "_key" : "la",
    "_id" : "bands/la",
    "_rev" : "_UULh3kC---",
    "name" : "L.A."
  },
  {
    "_key" : "lori-meyers",
    "_id" : "bands/lori-meyers",
    "_rev" : "_UULgc_K---",
    "name" : "Lori Meyers"
  }
]
127.0.0.1:8529@prueba>
```

Recorrido de un cursor

A la operación mediante la cual se recorre o accede un cursor documento a documento, se le conoce formalmente como extracción (*fetch*). Los cursores se pueden recorrer adelante y/o atrás. Atendiendo a las direcciones de recorrido, se distingue entre cursores unidireccionales y bidireccionales.



Un cursor unidireccional (*unidirectional cursor* o *forward-only cursor*) es aquel que permite recorrer cada documento una única vez. Una vez el documento ha sido accedido, ya no se puede volver a acceder a través suyo. En cambio, un cursor bidireccional (*bidirectional cursor* o *scroll cursor*) puede recorrerse adelante y atrás, pudiendo revisitarse un documento en cualquier momento.

En ArangoDB, los cursores son unidireccionales. Sólo se pueden acceder hacia delante. Una vez accedido uno, ya no se puede volver a acceder mediante el mismo cursor.

La manera más sencilla de recorrer un cursor es mediante un bucle for-of. He aquí un ejemplo ilustrativo:

```
127.0.0.1:8529@prueba> for (let b of db.bands.all()) print(b.name);
L.A.
Lori Meyers
127.0.0.1:8529@prueba>
```

Otra posibilidad es mediante los métodos hasNext() y next() del cursor. El primero comprueba si el cursor tiene documentos pendientes de recorrer. Y el segundo devuelve el siguiente documento a

acceder. Ejemplo:

```
127.0.0.1:8529@prueba> bands = db.bands.all()
SimpleQueryAll(bands)
127.0.0.1:8529@prueba> while (bands.hasNext()) print(bands.next().name)
L.A.
Lori Meyers
127.0.0.1:8529@prueba>
```

