

Una vez sabemos cómo crear colecciones de documentos, vamos a ver detenidamente cómo insertar documentos en ellas. En las próximas lecciones, veremos cómo extraer documentos de las colecciones, cómo actualizarlos y suprimirlos. Todo ello desde dos puntos de vista distintos: la API simple y el lenguaje de consulta **AQL**.

Comenzamos la lección introduciendo qué es una consulta y cómo se puede realizar en **ArangoDB**. En nuestro caso, mediante la API simple, proporcionada por el *shell* y el lenguaje de consulta **AQL**, mediante el *shell* y la interfaz web. A continuación, se describe en qué consiste una inserción y se hace hincapié en los campos claves, **_key** e **_id**. Después, se describe detalladamente cómo realizar inserciones simples y múltiples mediante la API simple y **AQL**. Finalmente, se muestra cómo ejecutar consultas parametrizadas **AQL**.

Al finalizar la lección, el estudiante sabrá:

- Qué es una consulta.
- Qué es **AQL**.
- Cómo insertar documentos mediante la API simple y **AQL**.
- Cómo crear consultas parametrizadas **AQL**.
- Cómo ejecutar consultas **AQL** en la interfaz web de **ArangoDB**.
- Qué es la inyección de **AQL**.

Introducción

Una **consulta** (*query*) es un comando mediante el cual se realiza una determinada operación sobre la base de datos como, por ejemplo, la obtención de uno o más documentos y la inserción de un nuevo documento. En **ArangoDB**, las consultas principalmente se ejecutan mediante *drivers* como, por ejemplo, el de **JavaScript**, el *shell* de **ArangoDB** y la interfaz web.

ArangoDB dispone de dos maneras para ejecutar consultas, una API simple y su propio lenguaje de consulta conocido como **AQL**. La **API simple** (*simple API*) está formada por un conjunto de métodos que tiene el objeto colección, por ejemplo, del *shell* de **ArangoDB**. En cambio, **AQL** (*ArangoDB Query Language*, lenguaje de consulta de **ArangoDB**) es un lenguaje específico para realizar consultas de lectura/escritura sobre la base de datos. Es más rico y potente que la API de colección.

Uno de los elementos más importantes de un motor de bases de datos es su **lenguaje de consulta** (*query language*), aquel que se utiliza para acceder a una instancia de bases de datos. El lenguaje por excelencia es, sin ninguna duda, **SQL**, diseñado para motores relaciones. Debido a su éxito, muchos motores no relacionales crean lenguajes similares a él. **ArangoDB** no podía ser menos y proporciona su propio lenguaje, **AQL**. No es idéntico, pero sigue unas pautas muy similares. Favoreciendo su aprendizaje y su utilización.

Tal como acabamos de indicar, **ArangoDB** proporciona dos maneras distintas de acceder a una instancia: la API simple, orientada a **JavaScript**; y **AQL**, orientado a **SQL**. En vez de presentar cada forma de manera separada, lo haremos de manera conjunta. Así el estudiante verá los pros y los contras de cada una de ellas en cada situación.

DML, DDL y DCL

Los lenguajes o APIs usadas por los motores de gestión de bases de datos se dividen básicamente en varios conjuntos de operaciones relacionadas con su objetivo final:

- **DML** (*data manipulation language*, lenguaje de manipulación de datos). Formado por las operaciones y/o comandos que manipulan los datos para su extracción, inserción, modificación

o supresión. En nuestro caso, aquellas que se encargan de extraer, insertar, actualizar o suprimir documentos de las colecciones.

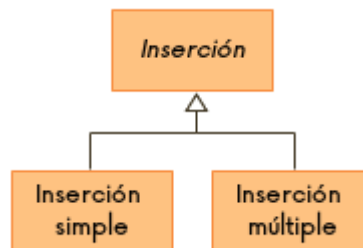
- **DDL** (*data definition language*, lenguaje de definición de datos). Formado por las operaciones y/o comandos que crean objetos en la base de datos como, por ejemplo, las colecciones y los índices.
- **DCL** (*data control language*, lenguaje de control de datos). Formado por las operaciones y/o comandos que conceden o revocan permisos a los usuarios sobre los recursos de la base de datos.

La API simple dispone de los tres lenguajes, mientras que **AQL** sólo de comandos para realizar operaciones **DML**. Algunos comandos **DDL** y **DCL** ya los hemos visto en lecciones anteriores como, por ejemplo, `db._createDatabase()`, `db._dropDatabase()`, `db._create()`, etc. En las próximas lecciones, vamos a centrarnos en el **DML** a través del cual extraer, insertar, actualizar y suprimir documentos. Comenzando con la inserción.

Inserción de documentos

La **inserción de documentos** (*document insert*) es el comando u operación mediante la cual se añade uno o más documentos a una colección. Tanto la API simple como **AQL** proporcionan medios con los que llevarla a cabo.

Atendiendo al número de documentos insertados, se distingue entre inserción simple o múltiple.



La **inserción simple** (*simple insert*) es aquella que inserta un único documento. Mientras que la **inserción múltiple** (*multipleinsert*), varios.

Claves

Recordemos que todo documento almacenado en una colección dispone de dos claves: la clave primaria y el identificador. La **clave primaria** (*primary key*) es el campo `_key` y contiene un valor que debe ser único en la colección donde se encuentra almacenado el documento. Nunca dos o más documentos tendrán el mismo valor de clave principal en la misma colección. Por su parte, el **identificador** (*identifier*), representado por el campo `_id`, contiene un valor único para toda la base de datos. Nunca dos o más documentos de la base de datos tendrán el mismo identificador.

La clave primaria la puede fijar el usuario. En caso de no hacerlo, será el motor de **ArangoDB** quien lo fije aleatoriamente. En cambio, el identificador lo fija siempre el motor, teniendo como valor una cadena con el siguiente formato `nombreDeColección/clavePrimaria`.

Inserción simple

Mediante la inserción simple, se puede insertar un documento en una colección. Se puede hacer mediante la API simple o bien mediante **AQL**.

Permisos

Para insertar un documento, es necesario que el usuario tenga concedido el permiso de lectura/escritura sobre la base de datos donde se encuentra la colección.

Inserción simple mediante API simple

Para insertar un nuevo documento en una colección, se puede usar los métodos `insert()` y `save()`, son

sinónimos el uno del otro:

```
insert(doc)
insert(doc, opts)
save(doc)
save(doc, opts)
```

Parámetro	Tipo de datos	Descripción
-----------	---------------	-------------

<code>doc</code>	object	Documento a insertar.
<code>opts</code>	object	Opciones de inserción: <ul style="list-style-type: none">• <code>waitForSync</code> (boolean). ¿Inserción síncrona? <code>true</code>, sí; <code>false</code>, no, asíncrona. Valor predeterminado: el fijado en la propiedad homónima de la colección.• <code>silent</code> (boolean). ¿El método debe devolver algo? <code>true</code>, no; <code>false</code>, sí.• <code>returnNew</code> (boolean). ¿El método debe devolver el documento completo insertado? <code>true</code>, sí; <code>false</code>, no.

Si el documento no proporciona un valor para el campo `_key`, el motor de **ArangoDB** se lo creará automáticamente. El motor siempre fija los valores de los campos `_id` y `_rev`. Si se indican en el documento, sus valores serán omitidos.

Ejemplo:

```
db.bands.insert({
  _key: "bmx-bandits",
  name: "BMX Bandits",
  genres: ["indie rock", "indie pop"]
});
```

Inserción simple mediante AQL

Para insertar un documento en una colección mediante **AQL**, hay que utilizar el comando **INSERT**, cuya sintaxis básica es:

```
INSERT documento
INTO colección
```

He aquí un ejemplo ilustrativo:

```
INSERT {
  _key: "bmx-bandits",
  name: "BMX Bandits",
  genres: ["indie rock", "indie pop"]
} INTO bands
```

Inserción múltiple

La **inserción múltiple** (*multiple insert*) permite añadir varios documentos mediante una única operación. Se puede realizar mediante los métodos `insert()` y `save()` de la API simple y el comando **INSERT** de **AQL**.

Permisos

Para insertar varios documentos mediante una única instrucción, es necesario que el usuario tenga concedido el permiso de lectura/escritura sobre la base de datos donde se encuentra la colección.

Inserción múltiple mediante API simple

Para insertar varios documentos mediante `insert()` y `save()`, no hay más que pasarlos en un *array*:

```
insert(docs)
insert(docs, opts)
save(docs)
save(docs, opts)
```

Parámetro	Tipo de datos	Descripción
-----------	---------------	-------------

<code>docs</code>	object[]	Documentos a insertar.
-------------------	----------	------------------------

opts

object

Opciones de inserción. Las mismas que la versión simple.

Veamos un ejemplo ilustrativo:

```
db.bands.insert([
  { _key: "moby", name: "Moby"},
  { _key: "leonard-cohen", name: "Leonard Cohen"}
])
```

Inserción múltiple mediante AQL

En **AQL**, no se puede insertar varios documentos pasándolos mediante un *array*. Pero sí se puede insertar los resultantes de un comando de selección, tal como muestra el siguiente ejemplo:

```
FOR b IN bands
FILTER b.name LIKE "%a%"
INSERT b
INTO artists
```

Nota. En la próxima lección, presentamos la extracción de datos. Por ahora, sólo recuerde que puede insertar los documentos procedentes de una consulta de selección o extracción.

Ejecución de consultas AQL

La API simple proporciona métodos específicos para cada operación. Por ejemplo, la inserción se realiza mediante los métodos `insert()` y `save()`, la actualización mediante `update()` o `replace()` y la supresión mediante `remove()`. En cambio, **AQL** es un lenguaje de consulta que describe sus comandos en forma de texto. Para su ejecución, hay que utilizar el método `_query()` del objeto `db`:

```
_query(cmd)
_query(cmd, params)
```

Parámetro	Tipo de datos	Descripción
<code>cmd</code>	string	Texto del comando a ejecutar.
<code>params</code>	object	Parámetros de la consulta.

He aquí un ejemplo ilustrativo:

```
db._query('INSERT { _key: "the-mission", name: "The Mission"} INTO bands')
```

Consultas dinámicas

En **AQL**, se puede ejecutar consultas de dos maneras distintas, estática o dinámicamente.



Cuando se usa una **consulta AQL estática** (*static AQL query*), el texto de la consulta se conoce en tiempo de compilación. No hay que hacer ningún procesamiento especial previo a la ejecución. Mientras que cuando se utiliza una **consulta AQL dinámica** (*dynamic AQL query*), el texto de la consulta *no* se conoce en tiempo de compilación y hay que esperar a la ejecución para tenerlo.

El uso de **AQL** dinámico es útil en muchas ocasiones, pero hay que utilizarlo con cuentagotas porque presenta, principalmente, las siguientes desventajas:

- Las consultas dinámicas no se validan en tiempo de compilación. Si la consulta presenta algún error sintáctico o hace referencia a un objeto inexistente, habrá que esperar a su ejecución para detectarlo.
- El motor de consultas debe generar el plan de ejecución de la consulta dinámica cada vez que la ejecuta. A diferencia de las consultas estáticas donde el plan de ejecución podría estar ya creado si la consulta se ejecutó anteriormente.

- Las consultas dinámicas simples pueden desembocar en inyección de **AQL** malicioso, algo que no puede producirse en la ejecución de consultas estáticas.

Consultas parametrizadas

Una **consulta parametrizada** (*parameterized query*) es un tipo especial de consulta dinámica que contiene parámetros en el texto del comando, donde cada **parámetro** (*parameter*), también conocido formalmente como **variable ligada** (*bind var*), representa un valor a insertar en la posición donde se encuentra. Los parámetros se representan mediante nombres precedidos por una arroba (@):

@parámetro

Los valores de los parámetros se pasan como segundo argumento del método `_query()`, mediante un objeto donde cada propiedad representa un parámetro. No hay nada como recapitular todo en un ejemplo ilustrativo:

```
db._query("INSERT {_key: @key, name: @name} INTO bands", {
  key: "the-psychedelic-furs",
  name: "The Psychedelic Furs"
})
```

Consultas plantillas

El método `_query()` espera la información de una consulta. Por un lado, su texto y, por otro lado, si es necesario, los parámetros. Existe una manera adicional de crear consultas parametrizadas. Una **consulta plantilla** (*template query*) es un tipo especial de consulta parametrizada que proporciona toda la información de la consulta mediante un literal plantilla de **JavaScript**.

Veamos primero un ejemplo ilustrativo:

```
127.0.0.1:8529@prueba> var key = "the-stone-roses"
127.0.0.1:8529@prueba> var name = "The Stone Roses"
127.0.0.1:8529@prueba> db._query(aql`INSERT {_key: ${key}, name: ${name}} INTO bands`)
[object ArangoQueryCursor, count: 0, hasMore: false]
127.0.0.1:8529@prueba>
```

Analicemos la consulta. Por un lado, observe que ahora los parámetros se especifican mediante expresiones, `$f`. Cada vez que la plantilla presenta una expresión de este tipo, la analiza, la ejecuta y el valor resultante lo añade a la consulta. Igual que las plantillas de **JavaScript**. Por otro lado, observe el controlador de plantillas **aql**, una función que recibe una plantilla y devuelve un objeto con la información que necesita `_query()` para ejecutar la consulta.

Inyección de AQL

La **inyección de AQL** (*AQL injection*) es una técnica mediante la cual se introduce código **AQL** en una consulta dinámica. Esta inyección de código podría desembocar en la ejecución de consultas no deseadas, permitiendo a un usuario malicioso vulnerar la seguridad del sistema. Las consultas estáticas no sufren de esta vulnerabilidad, sólo las consultas dinámicas. Es más, sólo son vulnerables las consultas dinámicas simples; las parametrizadas, no.

Por ejemplo, supongamos que tenemos una consulta en la que deseamos que el usuario indique una colección a consultar, algo como:

```
coll = "bands";
db._query("FOR doc IN " + coll + " RETURN doc");
```

El ejemplo anterior es bien intencionado y desembocaría en:

```
FOR doc IN bands RETURN doc
```

Hasta aquí todo bien. Ahora, suponga que el usuario en vez de proporcionar una colección, proporciona lo siguiente:

```
coll = "bands REMOVE doc IN bands";
```

Entonces la consulta ejecutada sería:

```
FOR doc IN bands REMOVE doc IN bands RETURN doc
```

Esto desembocaría en una consulta maliciosa e indeseada. Para resolver el problema, para evitar que los valores se inmiscuyan en el código, lo mejor es utilizar una consulta parametrizada como, por ejemplo:

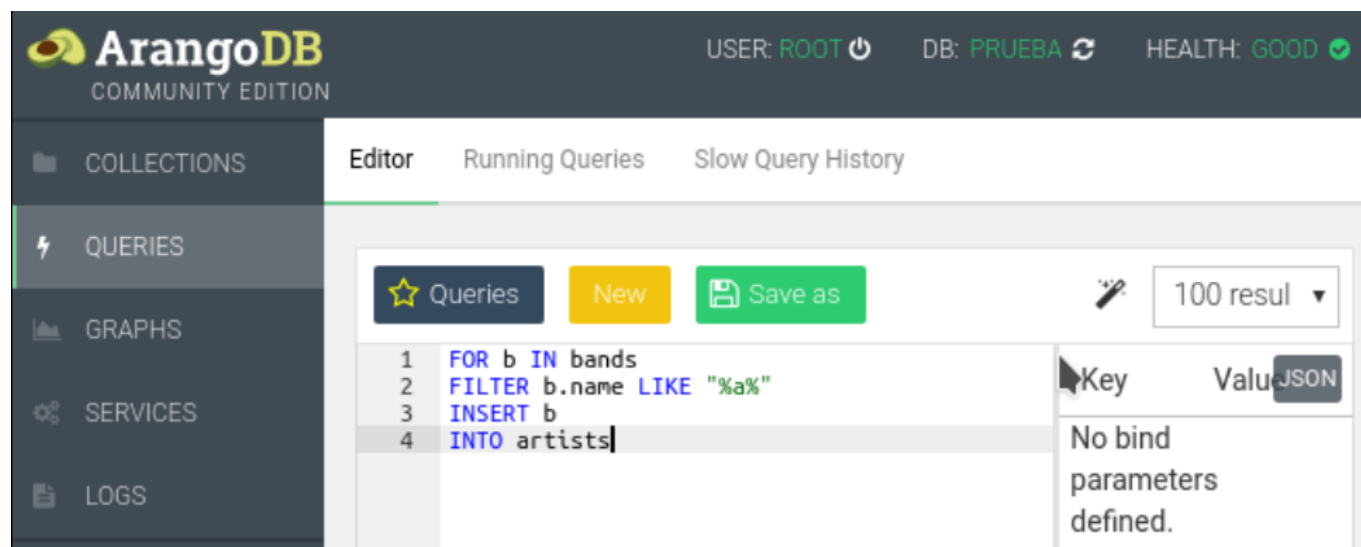
```
db._query("FOR doc IN @@coll RETURN doc", {"@coll": coll});
```

Así cuando pasásemos como valor de coll el código de inyección, el método `_query()` insertará el valor como el nombre de una colección, la cual obviamente no existe. Lo que hará fallar la consulta.

Cuando una consulta vaya a utilizar valores almacenados en variables o parámetros, se recomienda encarecidamente el uso de las consultas parametrizadas. No lo olvide. Son más seguras y no dan pie a la inyección de AQL malicioso.

Ejecución de consultas AQL mediante interfaz web

Las consultas AQL, independientemente de su tipo, se pueden ejecutar directamente en la interfaz web de ArangoDB. Ir a **QUERIES**, escribir la consulta en el editor y hacer clic en **Execute** para su ejecución:



The screenshot shows the ArangoDB web interface. The top bar displays the ArangoDB logo, 'COMMUNITY EDITION', and system status: 'USER: ROOT', 'DB: PRUEBA', and 'HEALTH: GOOD'. The left sidebar contains navigation links: 'COLLECTIONS', 'QUERIES' (highlighted), 'GRAPHS', 'SERVICES', and 'LOGS'. The main area is titled 'Editor' and contains a query editor with the following AQL code:

```
1 FOR b IN bands
2 FILTER b.name LIKE "%a%"
3 INSERT b
4 INTO artists
```

Below the editor, there are buttons for 'Queries', 'New', and 'Save as'. A dropdown menu is open, showing '100 resul' and a 'JSON' button. A tooltip message reads: 'No bind parameters defined.'