In this chapter, we will consider a few AQL Example Queries on an **Actors and Movies** Database. These queries are based on graphs.

Problem

Given a collection of actors and a collection of movies, and an actIn edges collection (with a year property) to connect the vertex as indicated below −

**[Actor] <- act in -> [Movie]**

How do we get −

* All actors who acted in "movie1" OR "movie2"?
* All actors who acted in both "movie1" AND "movie2”?
* All common movies between "actor1" and "actor2”?
* All actors who acted in 3 or more movies?
* All movies where exactly 6 actors acted in?
* The number of actors by movie?
* The number of movies by actor?
* The number of movies acted in between 2005 and 2010 by actor?

Solution

During the process of solving and obtaining the answers to the above queries, we will use Arangosh to create the dataset and run queries on that. All the AQL queries are strings and can simply be copied over to your favorite driver instead of Arangosh.

Let us start by creating a Test Dataset in Arangosh. First, download [this file](file:///C:\Apache24\htdocs\arangodb\dataset.js) −

# wget -O dataset.js

<https://drive.google.com/file/d/0B4WLtBDZu_QWMWZYZ3pYMEdqajA/view?usp=sharing>

Output

...

HTTP request sent, awaiting response... 200 OK

Length: unspecified [text/html]

Saving to: ‘dataset.js’

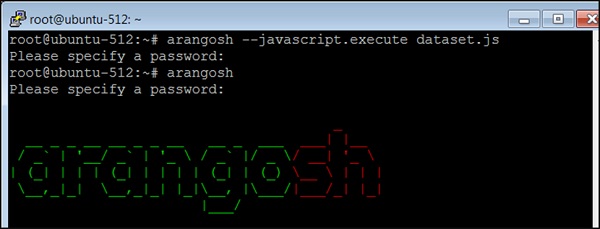
dataset.js [ <=> ] 115.14K --.-KB/s in 0.01s

2017-09-17 14:19:12 (11.1 MB/s) - ‘dataset.js’ saved [117907]

You can see in the output above that we have downloaded a JavaScript file **dataset.js.** This file contains the Arangosh commands to create the dataset in the database. Instead of copying and pasting the commands one by one, we will use the **--javascript.execute** option on Arangosh to execute the multiple commands non-interactively. Consider it the life saver command!

Now execute the following command on the shell −

$ arangosh --javascript.execute dataset.js



Supply the password when prompted as you can see in the above screenshot. Now we have saved the data, so we will construct the AQL queries to answer the specific questions raised in the beginning of this chapter.

First Question

Let us take the first question: **All actors who acted in "movie1" OR "movie2"**. Suppose, we want to find the names of all the actors who acted in "TheMatrix" OR "TheDevilsAdvocate" −

We will start with one movie at a time to get the names of the actors −

127.0.0.1:8529@\_system> db.\_query("FOR x IN ANY 'movies/TheMatrix' actsIn

OPTIONS {bfs: true, uniqueVertices: 'global'} RETURN x.\_id").toArray();

Output

We will receive the following output −

[

"actors/Hugo",

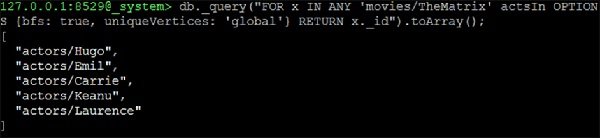
"actors/Emil",

"actors/Carrie",

"actors/Keanu",

"actors/Laurence"

]



Now we continue to form a UNION\_DISTINCT of two NEIGHBORS queries which will be the solution −

127.0.0.1:8529@\_system> db.\_query("FOR x IN UNION\_DISTINCT ((FOR y IN ANY

'movies/TheMatrix' actsIn OPTIONS {bfs: true, uniqueVertices: 'global'} RETURN

y.\_id), (FOR y IN ANY 'movies/TheDevilsAdvocate' actsIn OPTIONS {bfs: true,

uniqueVertices: 'global'} RETURN y.\_id)) RETURN x").toArray();

Output

[

"actors/Charlize",

"actors/Al",

"actors/Laurence",

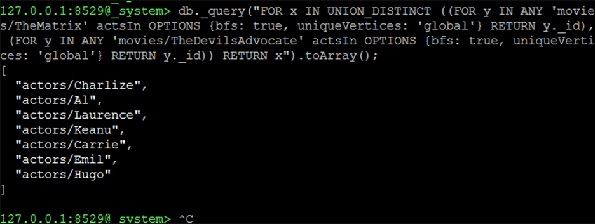
"actors/Keanu",

"actors/Carrie",

"actors/Emil",

"actors/Hugo"

]



Second Question

Let us now consider the second question: **All actors who acted in both "movie1" AND "movie2"**. This is almost identical to the question above. But this time we are not interested in a UNION but in an INTERSECTION −

127.0.0.1:8529@\_system> db.\_query("FOR x IN INTERSECTION ((FOR y IN ANY

'movies/TheMatrix' actsIn OPTIONS {bfs: true, uniqueVertices: 'global'} RETURN

y.\_id), (FOR y IN ANY 'movies/TheDevilsAdvocate' actsIn OPTIONS {bfs: true,

uniqueVertices: 'global'} RETURN y.\_id)) RETURN x").toArray();

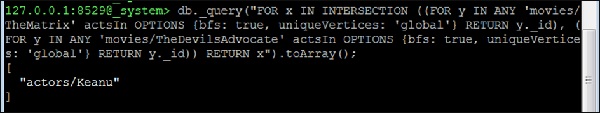
Output

We will receive the following output −

[

"actors/Keanu"

]



Third Question

Let us now consider the third question: **All common movies between "actor1" and "actor2"**. This is actually identical to the question about common actors in movie1 and movie2. We just have to change the starting vertices. As an example, let us find all the movies where Hugo Weaving ("Hugo") and Keanu Reeves are co-starring −

127.0.0.1:8529@\_system> db.\_query(

"FOR x IN INTERSECTION (

(

FOR y IN ANY 'actors/Hugo' actsIn OPTIONS

{bfs: true, uniqueVertices: 'global'}

RETURN y.\_id

),

(

FOR y IN ANY 'actors/Keanu' actsIn OPTIONS

{bfs: true, uniqueVertices:'global'} RETURN y.\_id

)

)

RETURN x").toArray();

Output

We will receive the following output −

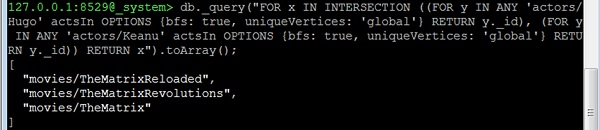
[

"movies/TheMatrixReloaded",

"movies/TheMatrixRevolutions",

"movies/TheMatrix"

]



Fourth Question

Let us now consider the fourth question. **All actors who acted in 3 or more movies**. This question is different; we cannot make use of the neighbors function here. Instead we will make use of the edge-index and the COLLECT statement of AQL for grouping. The basic idea is to group all edges by their **startVertex** (which in this dataset is always the actor). Then we remove all actors with less than 3 movies from the result as here we have included the number of movies an actor has acted in −

127.0.0.1:8529@\_system> db.\_query("FOR x IN actsIn COLLECT actor = x.\_from WITH

COUNT INTO counter FILTER counter >= 3 RETURN {actor: actor, movies:

counter}"). toArray()

Output

[

{

"actor" : "actors/Carrie",

"movies" : 3

},

{

"actor" : "actors/CubaG",

"movies" : 4

},

{

"actor" : "actors/Hugo",

"movies" : 3

},

{

"actor" : "actors/Keanu",

"movies" : 4

},

{

"actor" : "actors/Laurence",

"movies" : 3

},

{

"actor" : "actors/MegR",

"movies" : 5

},

{

"actor" : "actors/TomC",

"movies" : 3

},

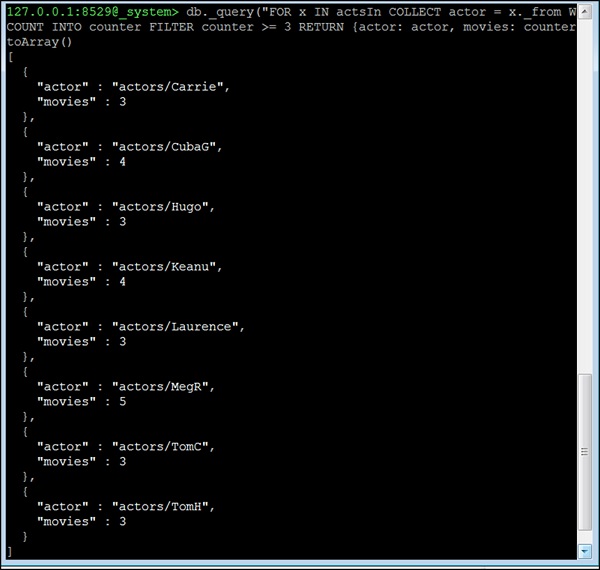
{

"actor" : "actors/TomH",

"movies" : 3

}

]



For the remaining questions, we will discuss the query formation, and provide the queries only. The reader should run the query themselves on the Arangosh terminal.

Fifth Question

Let us now consider the fifth question: **All movies where exactly 6 actors acted in**. The same idea as in the query before, but with the equality filter. However, now we need the movie instead of the actor, so we return the **\_to attribute** −

db.\_query("FOR x IN actsIn COLLECT movie = x.\_to WITH COUNT INTO counter FILTER

counter == 6 RETURN movie").toArray()

The number of actors by movie?

We remember in our dataset **\_to** on the edge corresponds to the movie, so we count how often the same **\_to** appears. This is the number of actors. The query is almost identical to the ones before but **without the FILTER after COLLECT** −

db.\_query("FOR x IN actsIn COLLECT movie = x.\_to WITH COUNT INTO counter RETURN

{movie: movie, actors: counter}").toArray()

Sixth Question

Let us now consider the sixth question: **The number of movies by an actor**.

The way we found solutions to our above queries will help you find the solution to this query as well.

db.\_query("FOR x IN actsIn COLLECT actor = x.\_from WITH COUNT INTO counter

RETURN {actor: actor, movies: counter}").toArray()