|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |

**NoSQL Workshop 5 – DataStax DSE Graph**

Guido Schmutz   
Technology Manager   
May 2018

**In this workshop we will learn the basics of working with Gremlin. We will use a docker image to work with Gremlin on top of a DataSatax DSE Graph database.**

**All the code used in this workshop is also available in a text file. Please use this for copy-paste.**

# Prerequisites

For the workshop you need a docker environment with docker-compose installed on it. Docker Compose provides a way to start multiple docker container in once.

# Starting DataStax DSE Graph using Docker

In a Docker environment, create a new docker-compose.yml file with the following content:

version: '2'

networks:

default:

external:

name: demo

services:

dse:

image: "datastax/dse-server"

environment:

- DS\_LICENSE=accept

command:

-k

-s

-g

# Allow DSE to lock memory with mlock

cap\_add:

- IPC\_LOCK

ulimits:

memlock: -1

ports:

- "4040:4040"

- "7080:7080"

- "7199:7199"

- "8182:8182"

- "8983:8983"

- "9042:9042"

container\_name: "dse"

studio:

image: "datastax/dse-studio"

environment:

- DS\_LICENSE=accept

volumes:

- ./studio:/var/lib/datastax-studio

- ./conf/studio:/config

ports:

- "9091:9091"

container\_name: "studio"

graph-loader:

image: trivadisbds/dse-graphloader:6.0.0

depends\_on:

- dse

volumes:

- ./graph-examples:/graph-examples

container\_name: "graph-loader"

Create the network which is used by the docker-compose definition above.

docker network create demo

Create the workspace/datastax folder inside /home/cas

cd /home/cas

mkdir workspace

cd workspace

mkdir datastax

cd datastax

and download the graph-examples folder from dropbox.

wget "https://www.dropbox.com/s/b4y3pfttetan38n/graph-examples.zip?dl=0"

unzip graph-examples.zip\?dl=0

rm graph-examples.zip\?dl\=0

Start docker-compose with the following command.

sudo docker-compose up -d

If you get a timeout error, then just redo the docker-compose up -d command.

Based on the docker-compose definition, four services will be created, a single Cassandra node, an instance of OpsCenter, an instance of DSE Studio and an instance of the DSE Graph Loader.

To see the log file of the 4 services running, use the following command

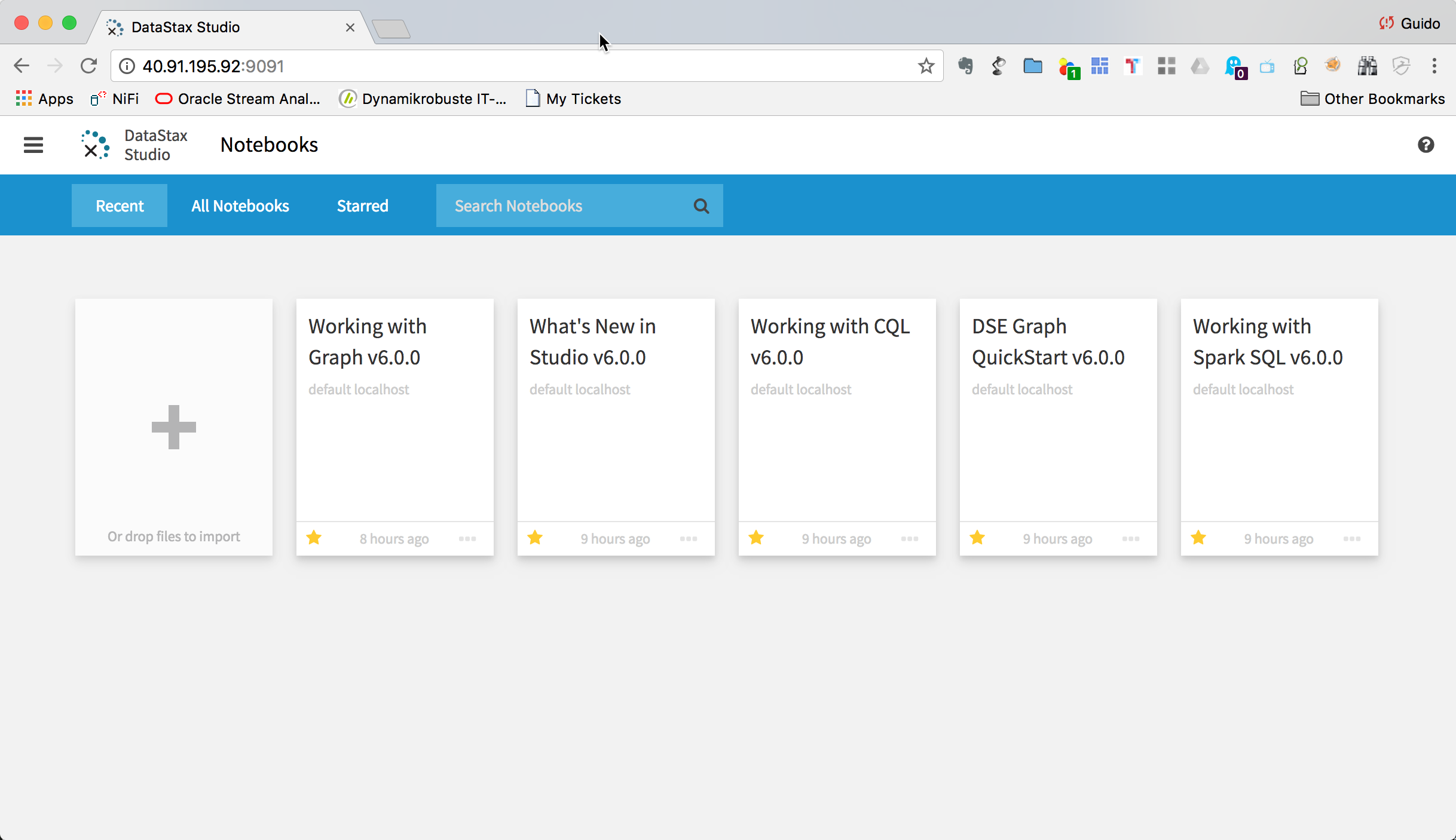
sudo docker-compose logs -f

# Create a new graph schema in DSE Studio

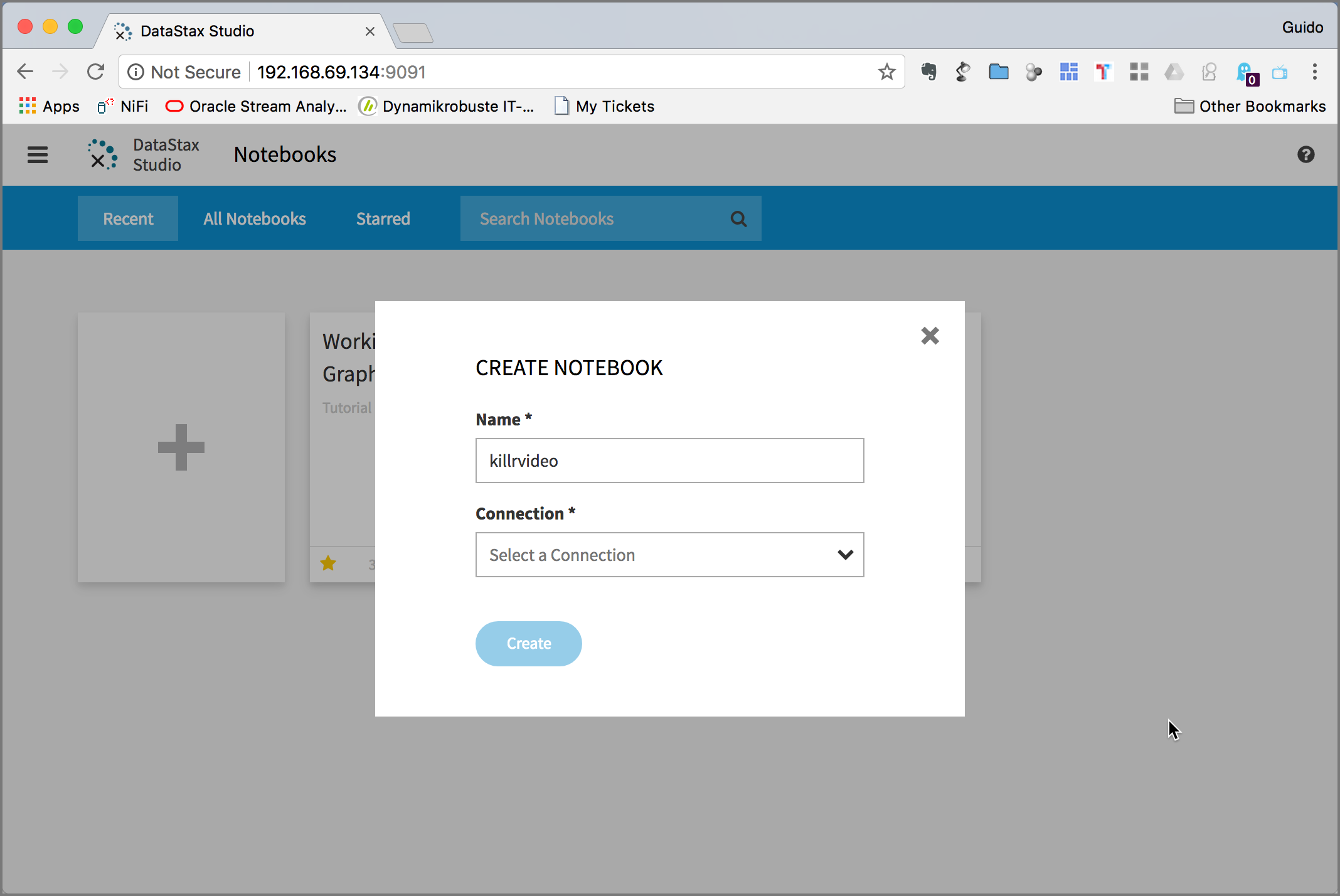
In order to work with DSE-Graph we will be using the graphical environment DSE Studio. It’s a notebook-style UI provided by DataStax.

In a browser, connect to <http://localhost:9091> inside the Virtual Machine or use the IP Address of the Virtual Machine to connect from externally (i.e. from the host machine). In Azure don’t forget to open the port 9091 and 9042 on the Networking page (add an inbound rule on TCP for port 9091 and for port 9042).

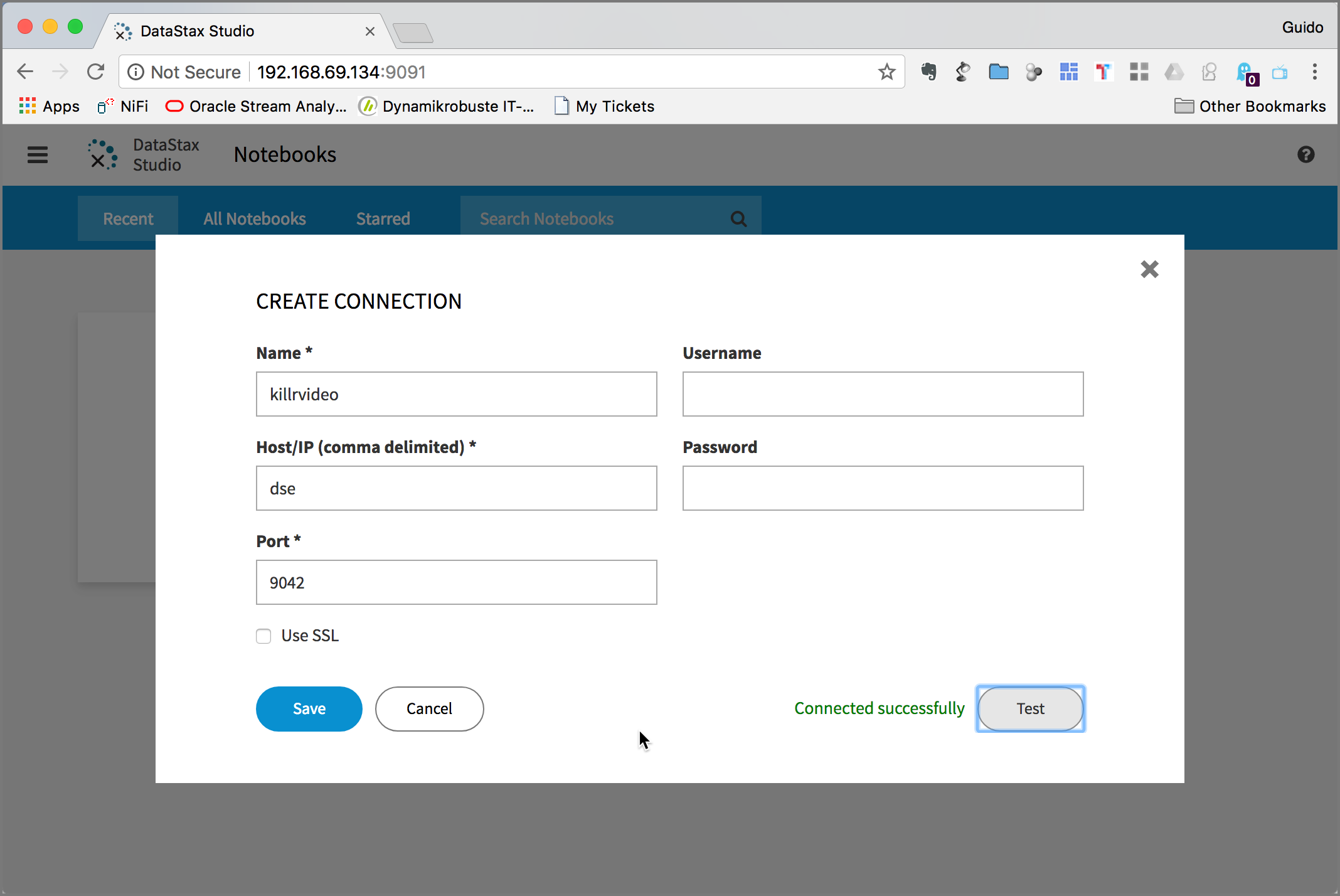
The following page should appear:



click on the **+** icon.

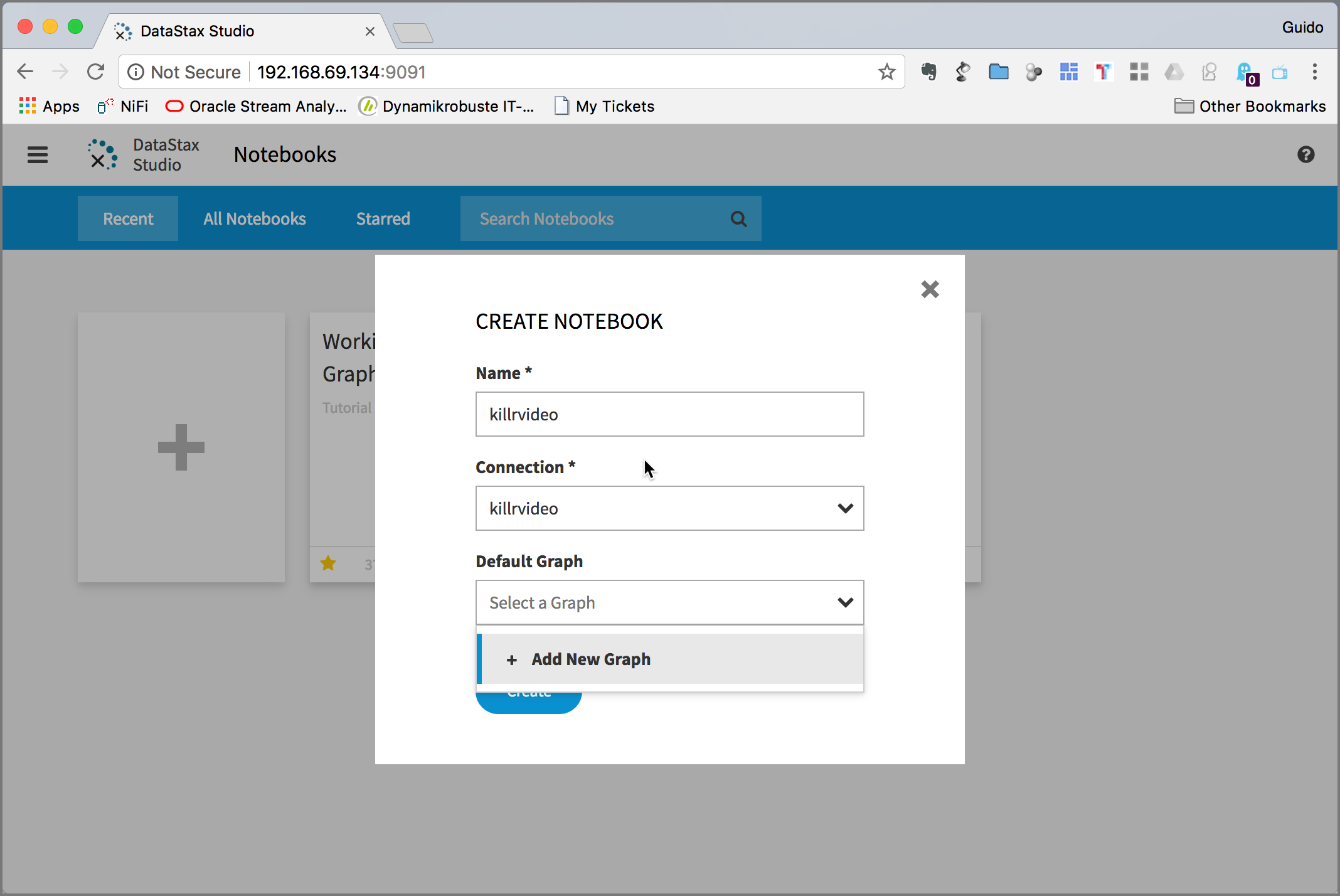


Click on **Connection** drop down and select **Add New Connection**.

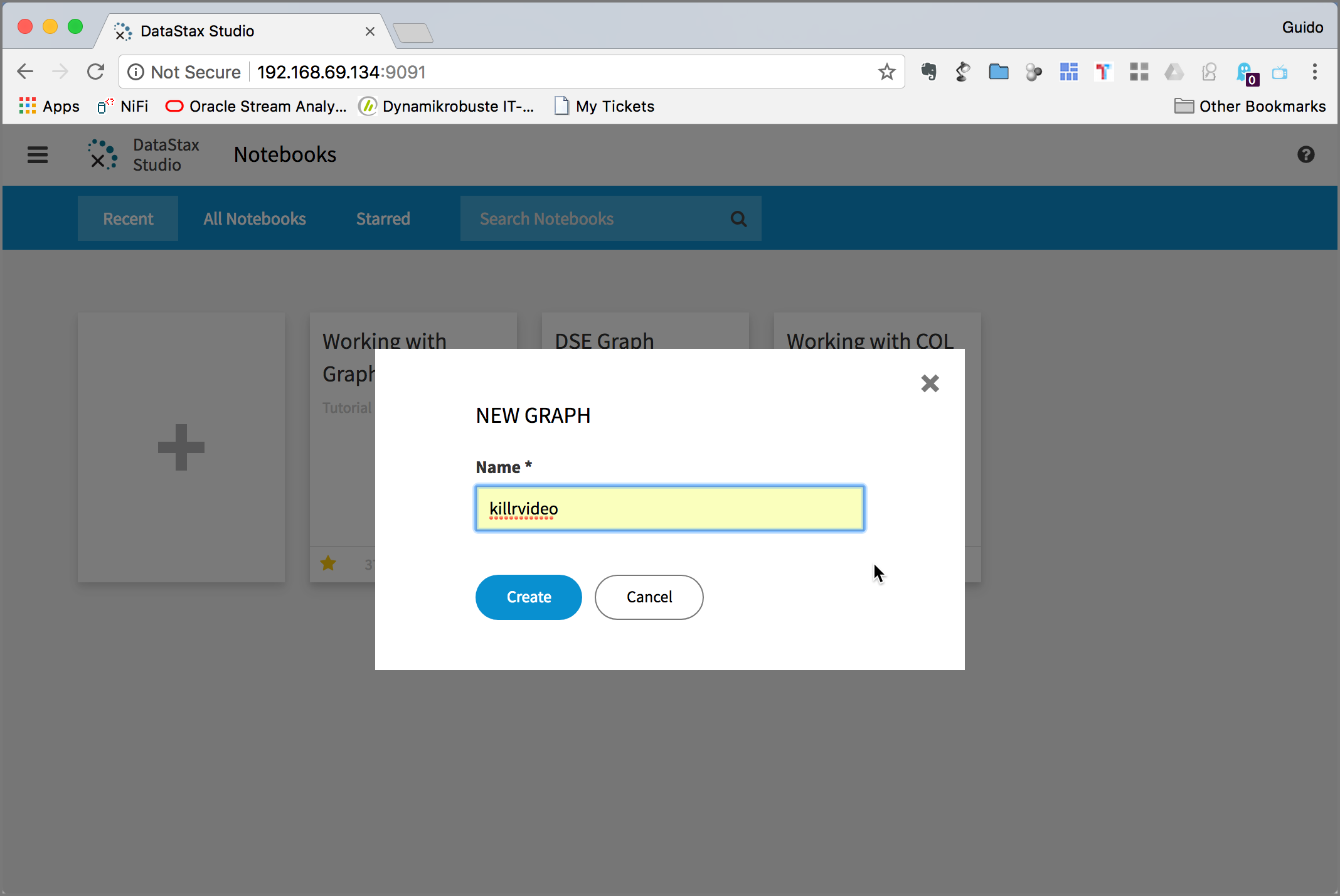


Enter **killrvideo** into the **Name** field and **dse** into the **Host/IP** field. Click on **Test** to test if connect to DSE Graph works successfully. If you get an error, you might have to make sure that port 9042 is reachable.

Click **Save** and you are back on the CREATE NOTEBOOK pop-up window.

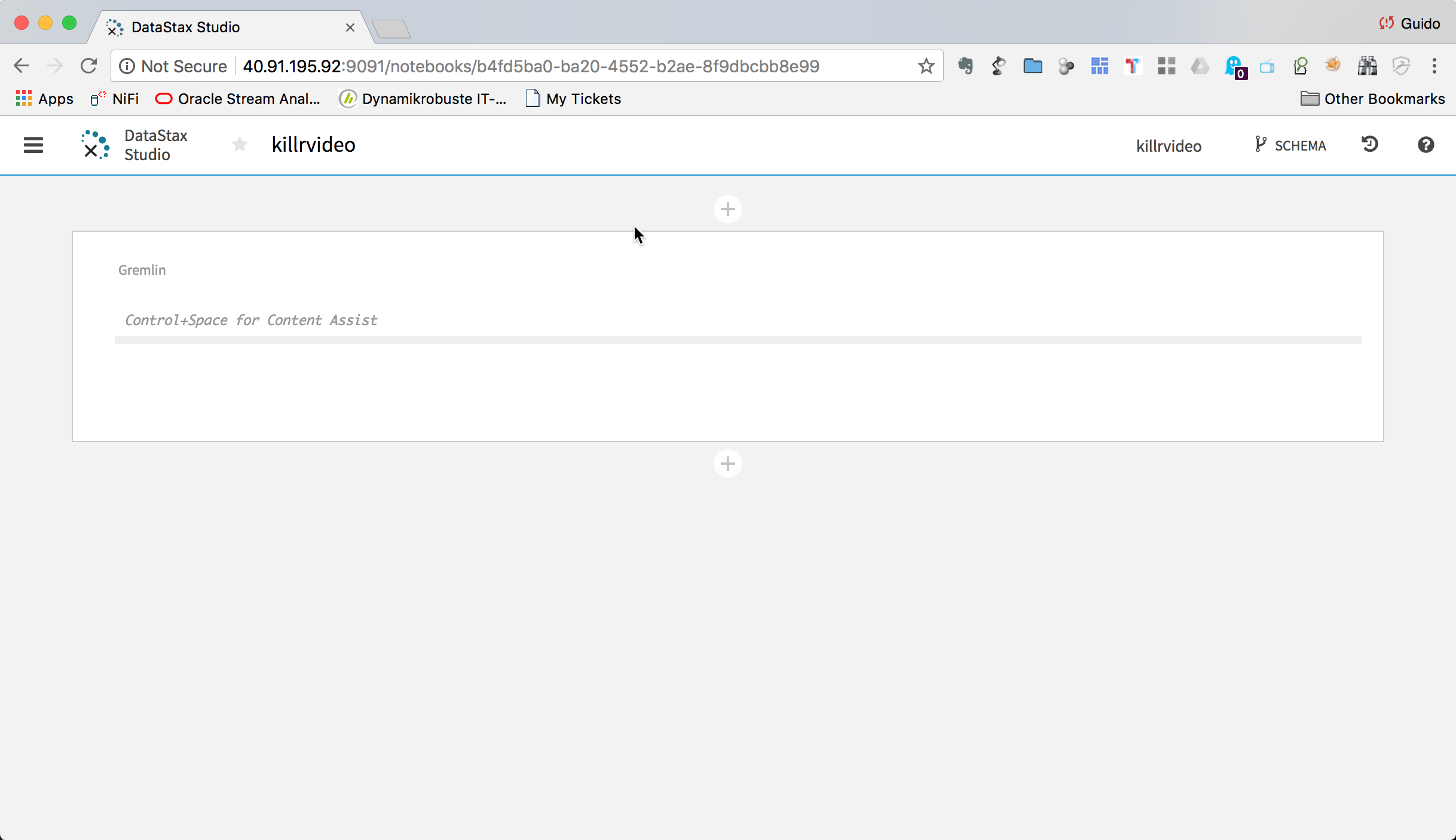


Click on the **Default Graph** drop-down and select **Add New Graph**.



Enter **killrvideo** into the **Name** field and then click **Create** and once again **Create**.

An empty notebook should appear. That’s your working space.



# Create the schema

Now we are ready to create the graph schema. The following diagram shows the graph model we are going to implement.



Copy the following code and paste it into the empty cell in the notebook.

// Either create the graph in studio and add the following in a cell

// or uncomment and run the following with the gremlin console

/\*

:remote config alias reset

system.graph("killrvideo").drop()

system.graph("killrvideo").ifNotExists().create()

:remote config alias g killrvideo.g

:remote config timeout max

\*/

// Define properties

schema.propertyKey("genreId").Text().create();

schema.propertyKey("personId").Text().create();

schema.propertyKey("userId").Text().create();

schema.propertyKey("movieId").Text().create();

schema.propertyKey("name").Text().create();

schema.propertyKey("age").Int().create();

schema.propertyKey("gender").Text().create();

schema.propertyKey("title").Text().create();

schema.propertyKey("year").Int().create();

schema.propertyKey("duration").Int().create();

schema.propertyKey("country").Text().create();

schema.propertyKey("production").Text().multiple().create();

schema.propertyKey("rating").Int().create();

// Define vertex labels

schema.vertexLabel("genre").properties("genreId","name").create();

schema.vertexLabel("person").properties("personId","name").create();

schema.vertexLabel("user").properties("userId","age","gender").create();

schema.vertexLabel("movie").properties("movieId","title","year","duration","country","production").create();

// Define edge labels

schema.edgeLabel("knows").connection("user","user").create();

schema.edgeLabel("rated").single().properties("rating").connection("user","movie").create();

schema.edgeLabel("belongsTo").single().connection("movie","genre").create();

schema.edgeLabel("actor").connection("movie","person").create(); // multiple() due to data

schema.edgeLabel("director").single().connection("movie","person").create();

schema.edgeLabel("composer").single().connection("movie","person").create();

schema.edgeLabel("screenwriter").connection("movie","person").create(); // multiple() due to data

schema.edgeLabel("cinematographer").single().connection("movie","person").create();

// Define vertex indexes

schema.vertexLabel("genre").index("genresById").materialized().by("genreId").add();

schema.vertexLabel("genre").index("genresByName").materialized().by("name").add();

schema.vertexLabel("person").index("personsById").materialized().by("personId").add();

schema.vertexLabel("person").index("personsByName").materialized().by("name").add();

schema.vertexLabel("user").index("usersById").materialized().by("userId").add();

schema.vertexLabel("user").index("usersByAge").secondary().by("age").add();

schema.vertexLabel("movie").index("moviesById").materialized().by("movieId").add();

schema.vertexLabel("movie").index("moviesByTitle").materialized().by("title").add();

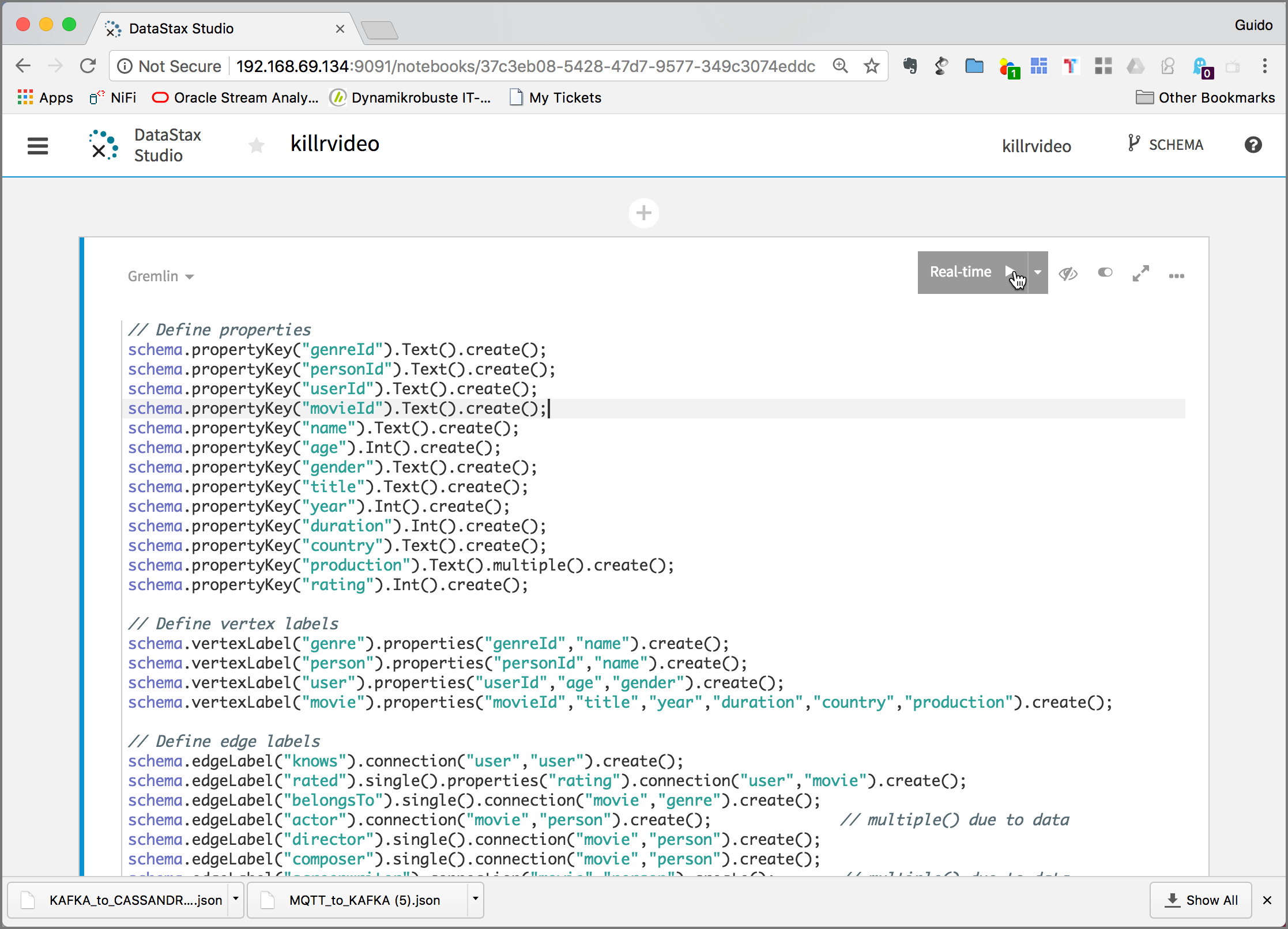
schema.vertexLabel("movie").index("moviesByYear").secondary().by("year").add();

// Define edge indexes

schema.vertexLabel("user").index("toMoviesByRating").outE("rated").by("rating").add();

schema.vertexLabel("movie").index("toUsersByRating").inE("rated").by("rating").add();

The notebook should now look like the diagram below:



Either click on the **Real-Time >** Button with the arrow in the upper right corner or hit **SHIFT-Enter** to execute the cell.

It will take a few seconds to create the schema.

# Load data into the graph

To load the data, we will use the DSE Graph Loader started previously with docker-compose. The folder with the example files has been mapped into the graph-loader container as **/graph-examples**. Navigate into the running graph-loader docker container

docker exec -ti graph-loader bash

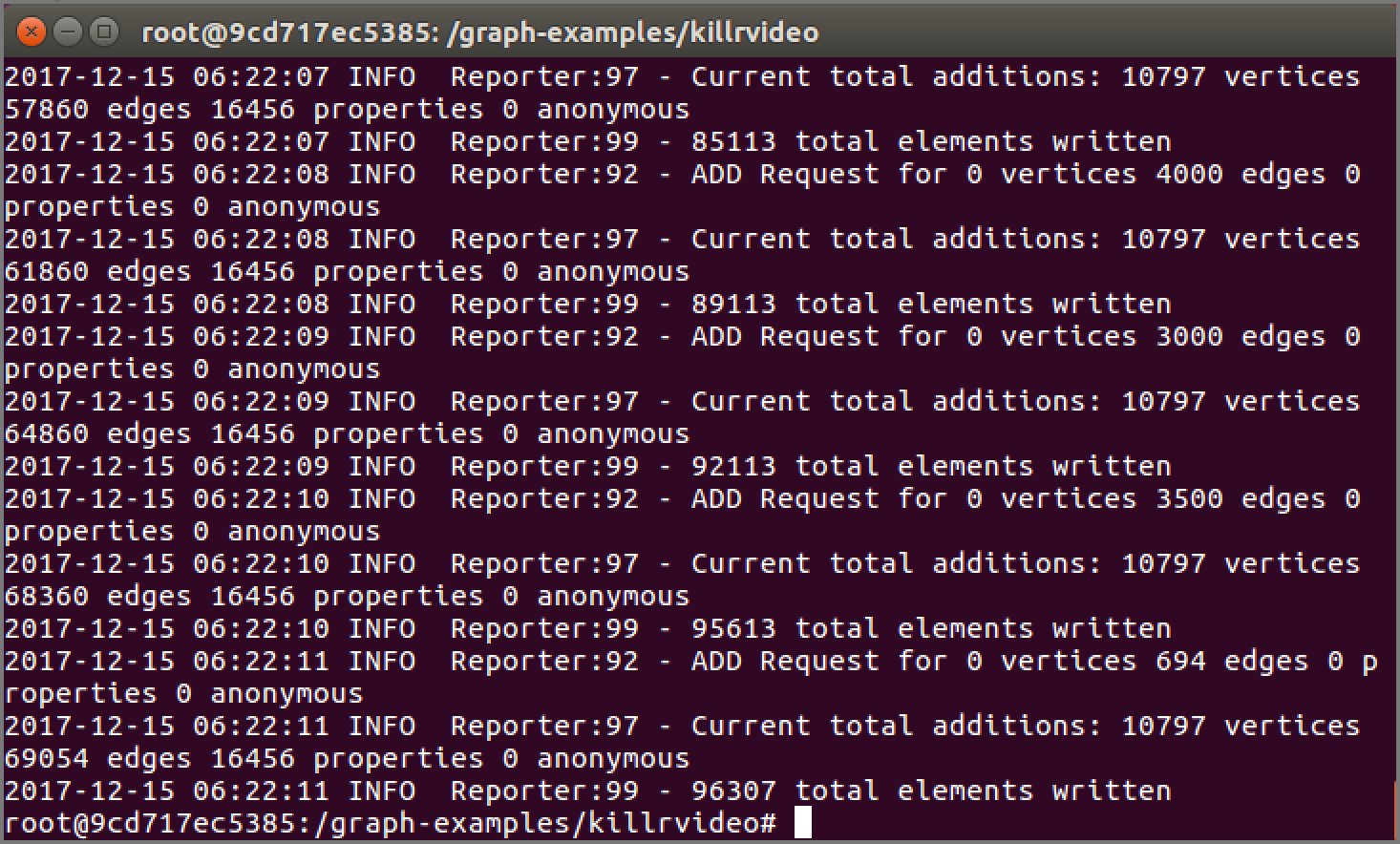
and change to the /graph-examples/killrvideo folder

cd /graph-examples/killrvideo/

Start the graph loader using the killrvideo-mapping.groovy mapping file.

/opt/dse-graph-loader/graphloader -address dse killrvideo-mapping.groovy -graph killrvideo -preparation false

It takes a few minutes for the load to finish.

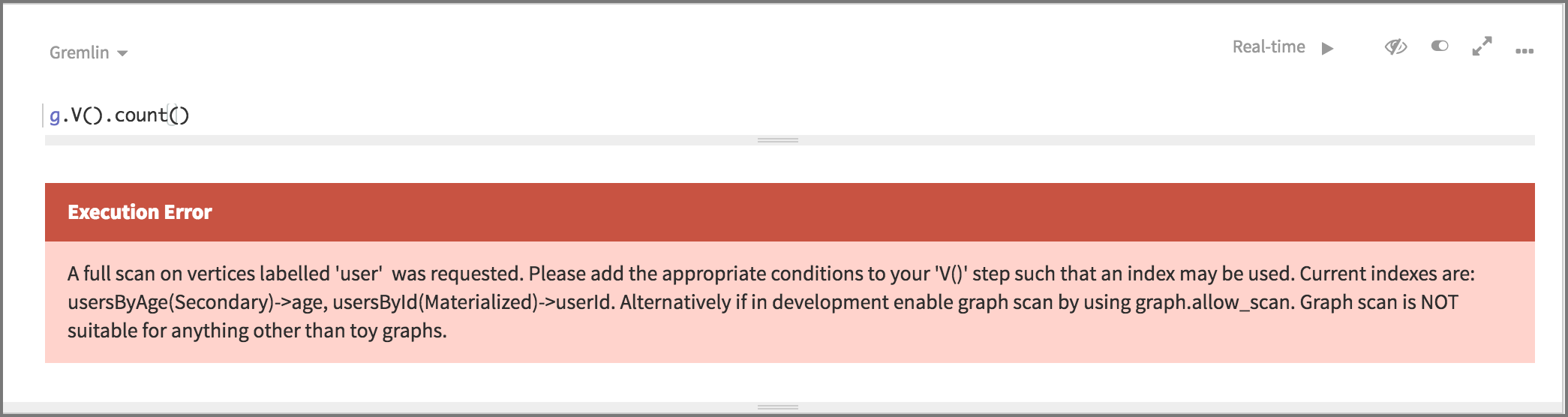


# Using the graph

Count the number of vertices in the graph.

g.V().count()

If will not work, because the standard settings of DSE Graph does not allow for scanning the whole graph.



You can change this with the allow\_scan option

schema.config().option('graph.allow\_scan').set('true')

Now try it again, the count the number of vertices should be returned.

g.V().count()

For each vertex in the graph, emit its label, then group and count each distinct label

g.V().label().groupCount()

For each rated-edge in the graph, emit its stars property value and compute the average value.

Let’s start with a simple traversal to return the director of the Movie titled “Alice in Wonderland”:

g.

V().

has("title","Alice in Wonderland").

has("year",2010).

out("director").

values("name")



g.

V().

has("title","Alice in Wonderland").

has("year",2010).

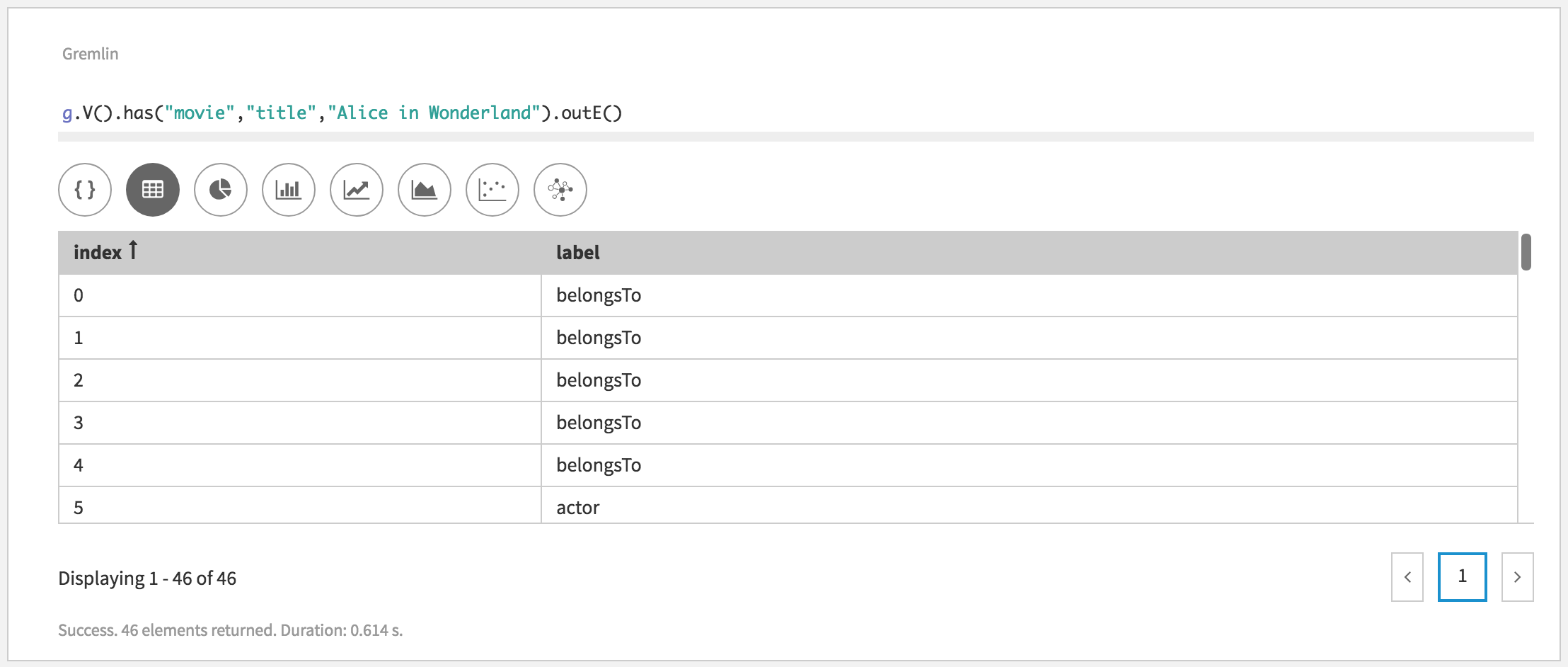
union(\_\_.out("director"),

out("screenwriter")).

values("name")

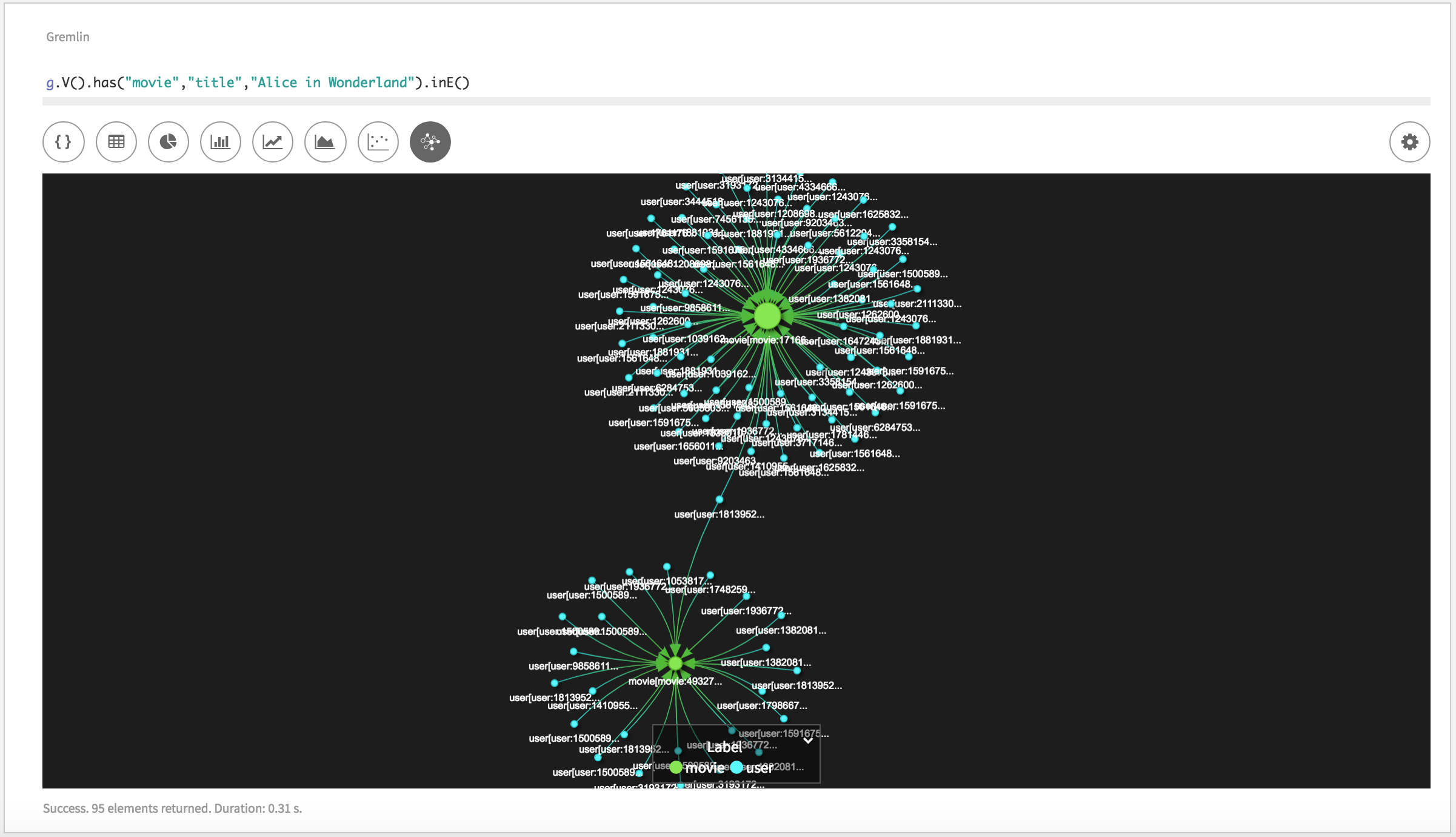
Now let’s get the movie titled “Alice in Wonderland” and show all out-going Edges:

g.V().has("movie","title","Alice in Wonderland").outE()

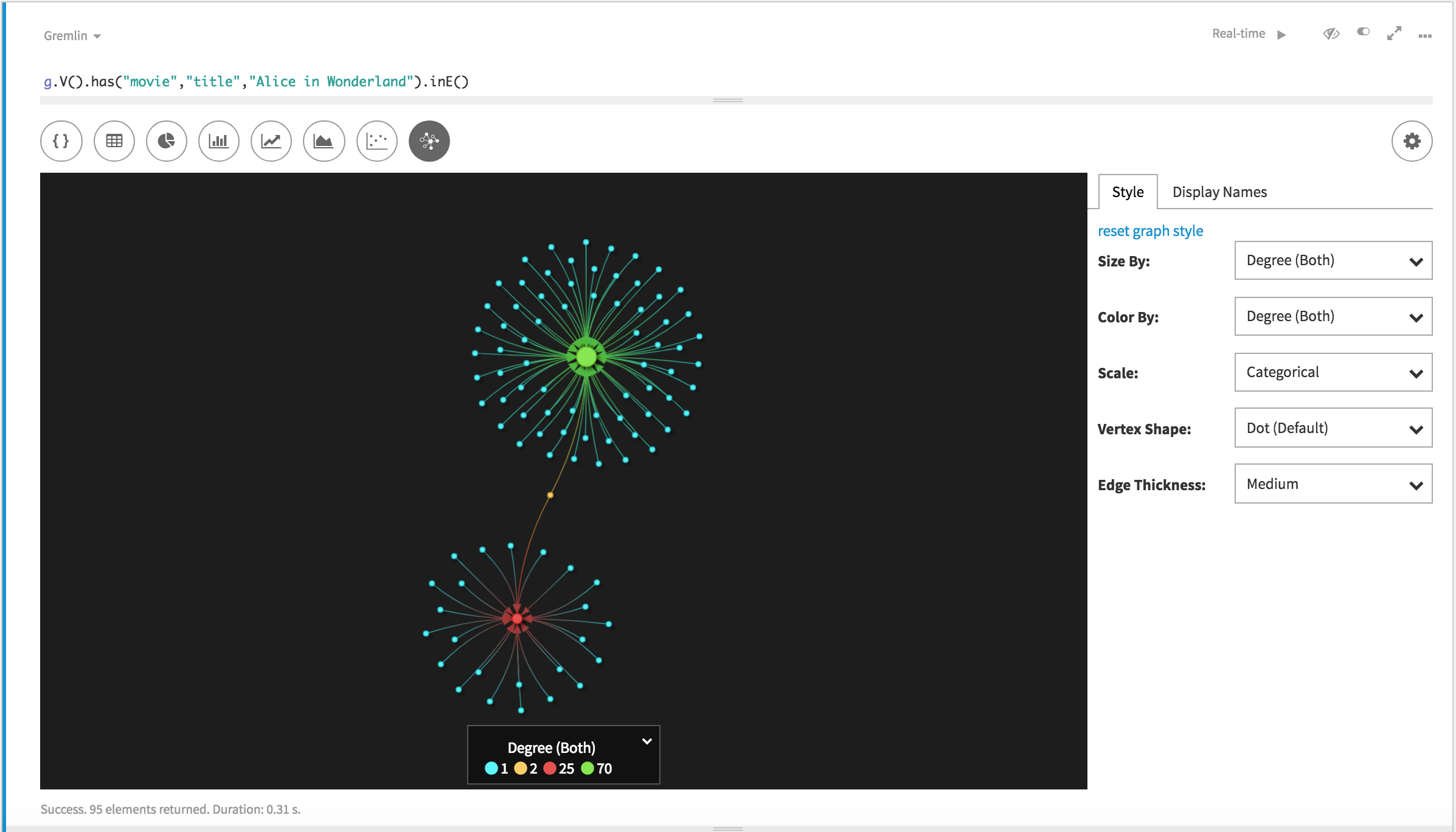
****

Next let’s get the movie titled “Alice in Wonderland” and show all in-coming Edges:

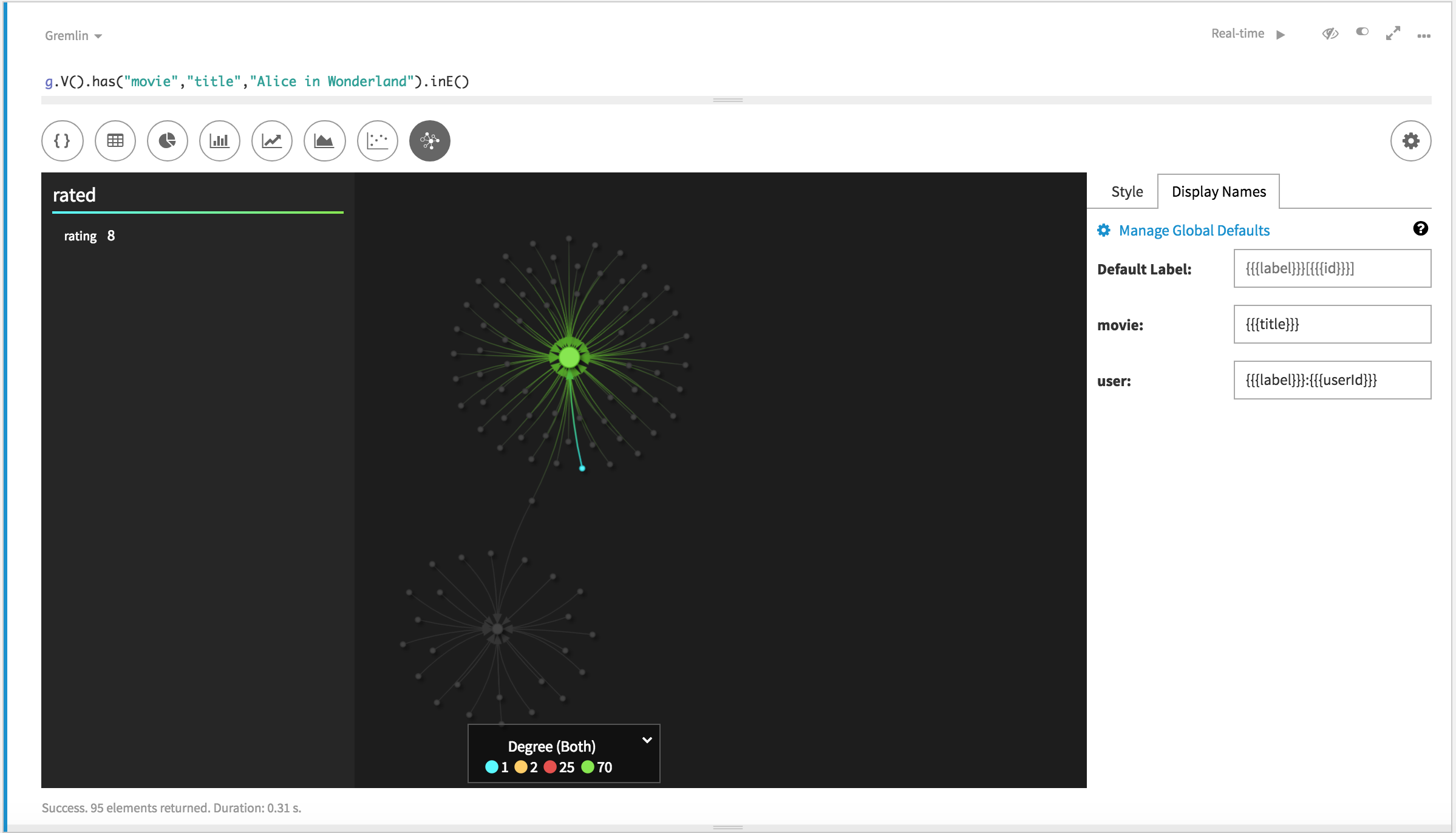
g.V().has("movie","title","Alice in Wonderland").inE()

****

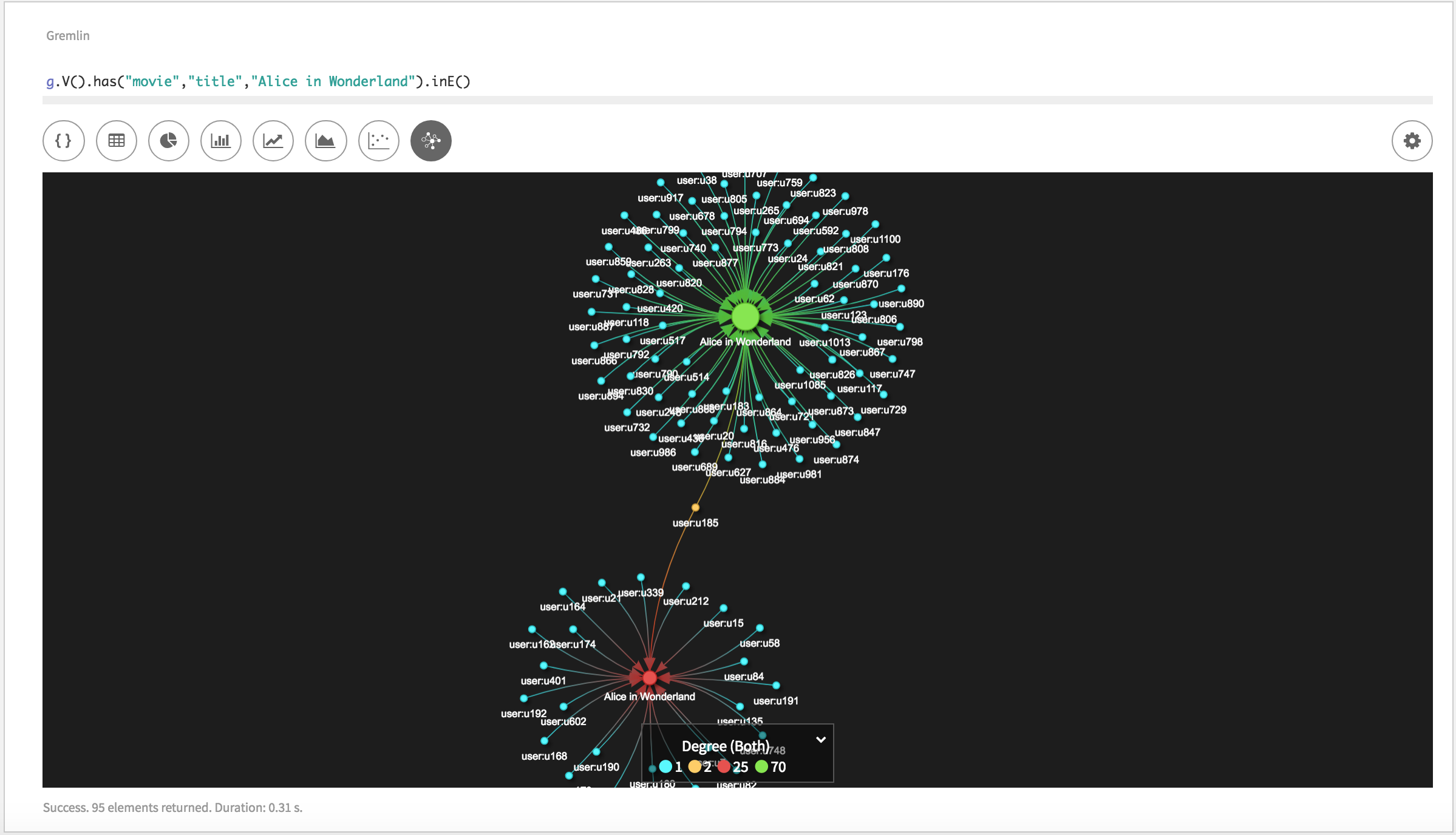
Click on **Graph Settings** icon on the right to show the graph preferences.

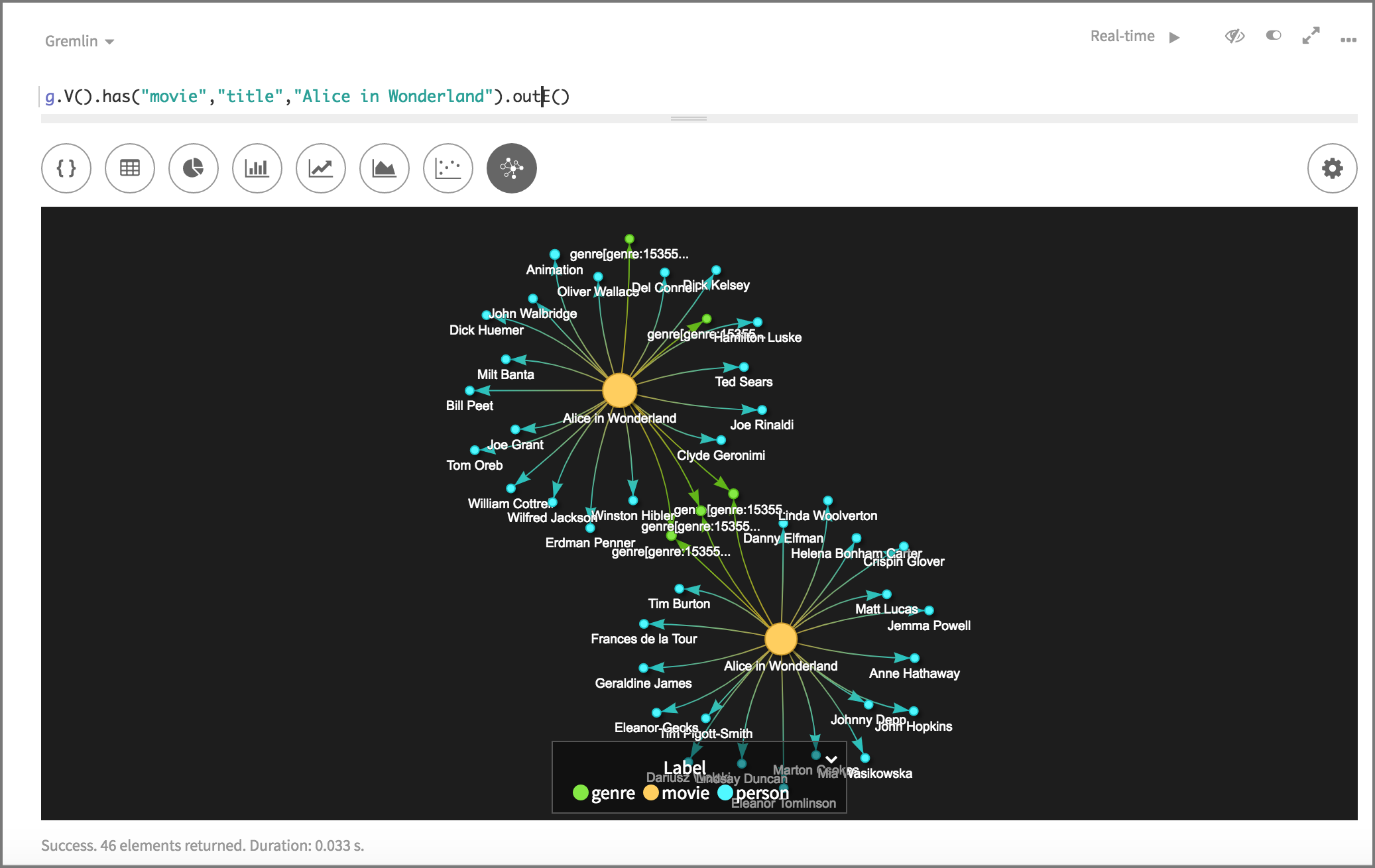


Click on the Display Names tab to change the information displayed in the graph visualization. Enter {{{title}}} into the **movie** field and {{{label}}}:{{{userId}}} into the **user** field.



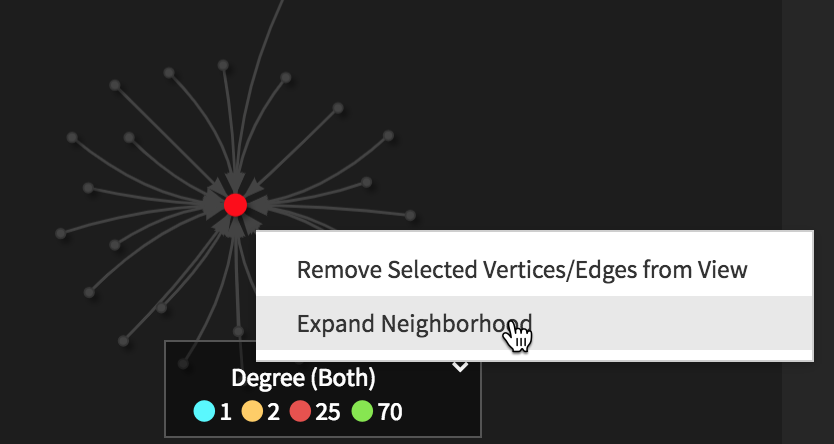
After you close the settings by clicking the Graph Settings again, the visualization will change.



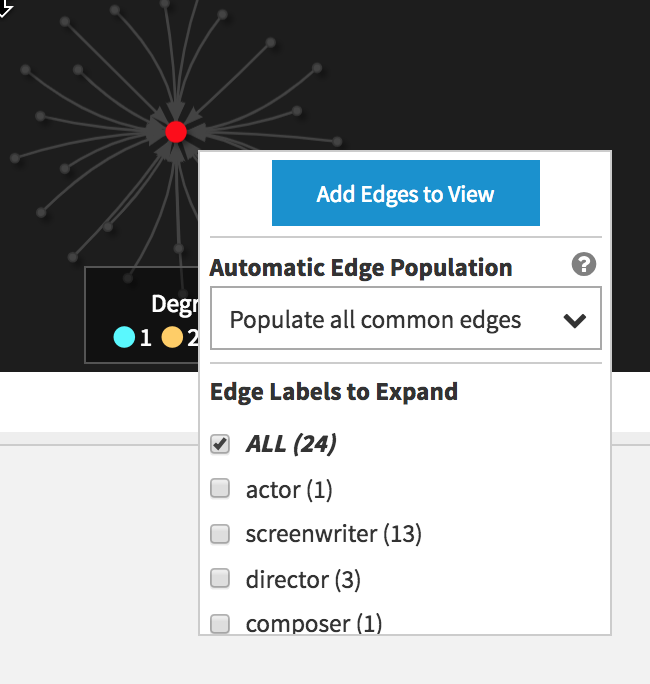
****

The visualization will start on the two movies (there are two movies with tile “Alice in Wonderland”, the first produced in 1951 and the second in 2010) and show all incoming edges and the vertices they come from.

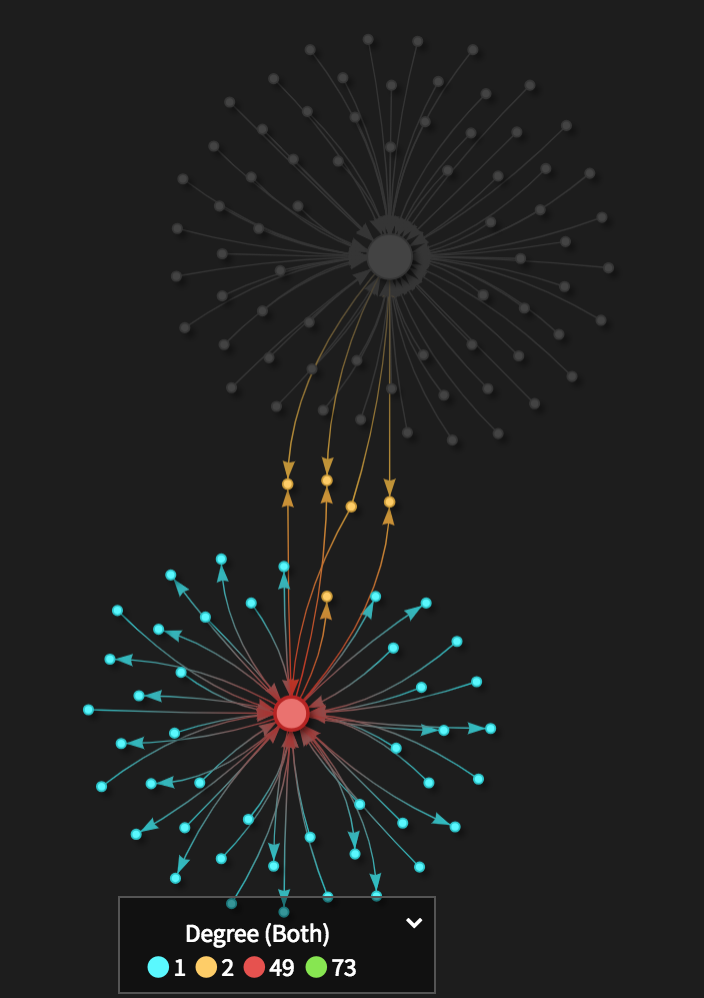
If you want to navigate in the graph, you can right click on a node and select **Expand Neighborhood**.



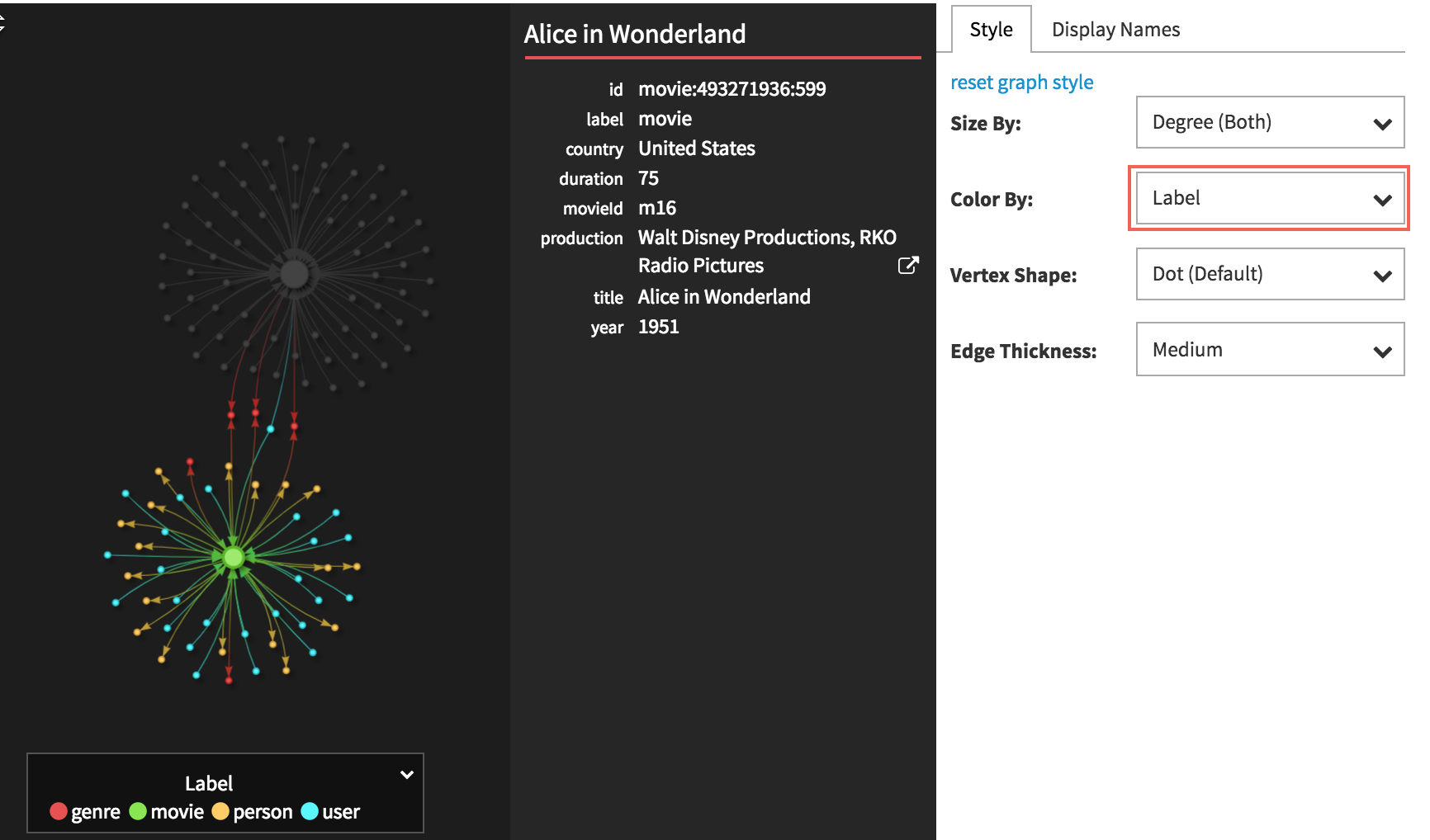
On the pop-up window, click on **Add Edges to View**.



We selected **ALL** edges to expand, which will result in the following new visualization.



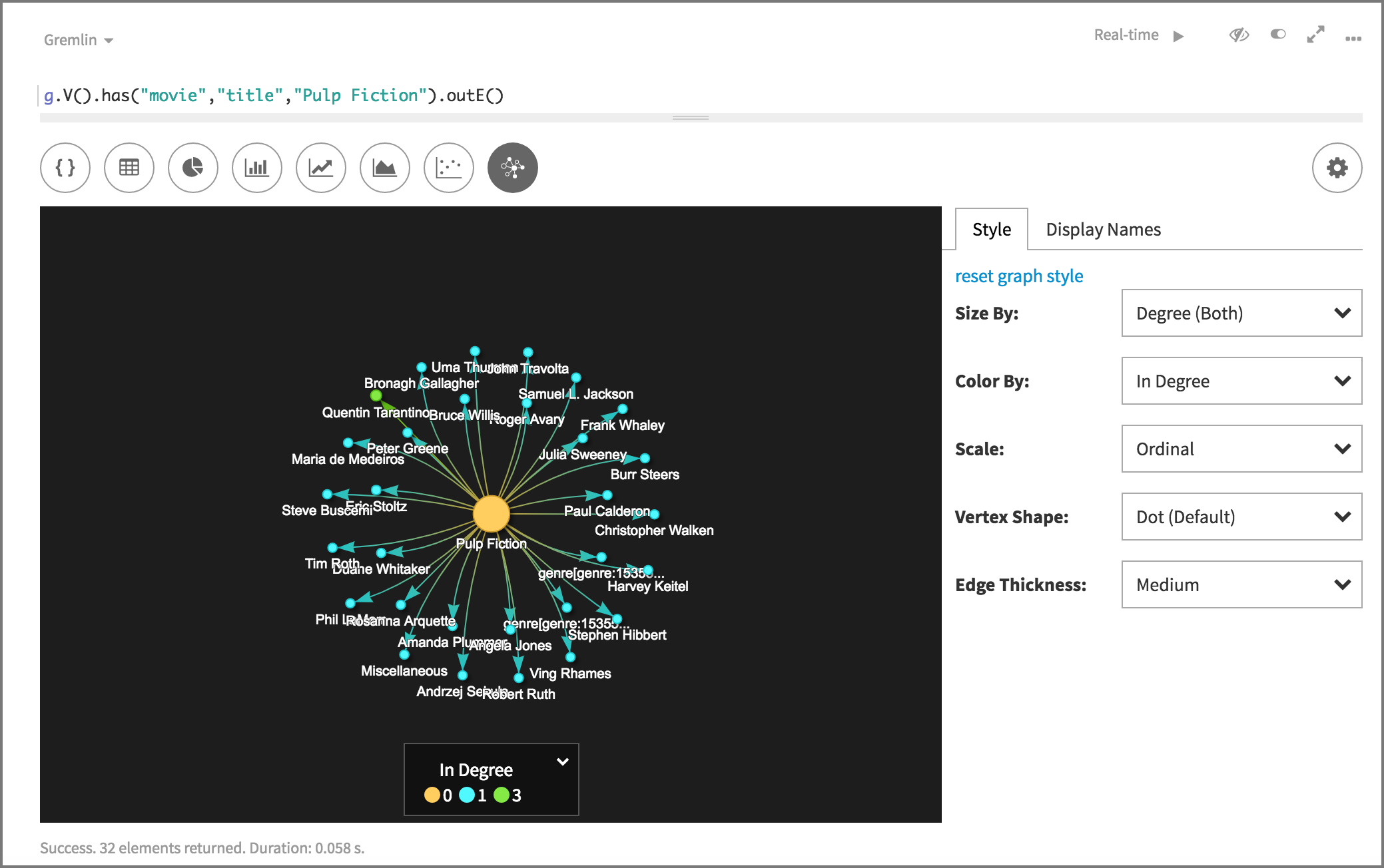
To see the different nodes, let’s change the color of the nodes to not represent the degree but the label.



Now we can see that due to the expand ALL, we can also see all links to **Genre** and **Movie** vertices.

Now let’s do it for another movie, the movie “Pulp Fiction”.

g.V().has("movie","title","Pulp Fiction").outE()

****

Next some more queries you can use on the movie graph.

Find user rating for Jonny Depp’s movies released in 2010 or later

g.V().

  hasLabel("person").

  has("name","Johnny Depp").

  in("actor").

  has("year",gte(2010)).

  inE("rated").

  values("rating")

Find Johnny Depp’s movie titles

g.V().has("person","name","Johnny Depp").

      as("actor").

      in("actor").as("movie").

      select("actor","movie").

      by("name").

      by("title").

      sample(1)

Find Johnny Depp’s movie titles, years, and genres

g.V().has("person","name","Johnny Depp").

      as("a").

      in("actor").as("t","y").

      out("belongsTo").as("g").

      select("a","t","y","g").

      by("name").by("title").by("year").by("name").

      sample(3)

Find directors who appeared in their own movies

g.V().match(

\_\_.as("m").out("actor").as("p"),

\_\_.as("m").out("director").as("p")).

select("p").by("name").dedup().

sample(10)