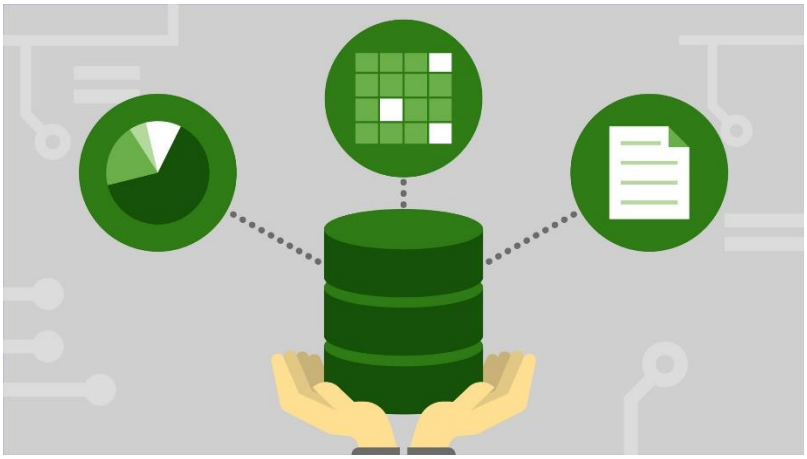
به نام خدا

دانشگاه تهران

پردیس دانشکده‌های فنی

دانشکده برق و کامپیوتر



آزمایشگاه پایگاه‌داده

دستورکار شماره 3

شماره دانشجویی

810196443

فروردین ۱۴۰۰

هومان چمنی

گزارش فعالیت‌های انجام شده

بخش نصب و آموزش اولیه در quackit:

پس از نصب نرم‌افزار در محیط Windows و آشنا شدن با رابط کاربری دیتابیس مذکور (که در به سادگی در Browser قابل مشاهده می‌باشد), موارد زیر که سعی شده است تا حد ممکن به ترتیب مشابه در آموزش باشند یاد گرفته شدند:

- آشنا شدن با مفاهیم اولیه مربوط به syntax زبان سایفر که به طور مثال node ها با پرانتز و رابطه‌ها با براکت مشخص می‌شوند.
- آشنا شدن با نحوه ایجاد کردن یک یا چند Node با استفاده از دستور Create و نحوه نشان دادن Node درست بعد از ساخته شدن با استفاده از دستور Return
- آشنا شدن با نحوه ساختن ارتباط بین دو Node که دو مرحله دارد. مرحله اول یافتن Node های مذکور با استفاده از دستورهای Match و Where و سپس ساختن یال با استفاده از دستور Create که دو Node را در پرانتز و یال را در براکت بین آن دو مشخص می‌کند.
- ساختن index برو روی یک property از یک Node که با استفاده از دستور Create index on مشخص می‌شود و بعد از آن دیتابیس در background اقدام به ساختن index مذکور می‌کند. با دستور schema نیز می‌توان تمامی index ها و constraint ها را مشاهده کرد. لازم به ذکر است که ساختن index هنگام زدن query نیز امکان پذیر بوده و با سینتکس using index بعد از عبارت match هندل می‌شود.
- آشنا شدن با Constraint ها. دو نوع محدودیت داریم. محدودیت یکتایی که مشخص می‌کند یک property در یک Node می‌بایست یکتا باشد (نسبت به property های مشابه در Node های هم‌جنس دیگر). محدودیت موجودیت نیز که فقط در نسخه enterprise دیتابیس ما قابل استفاده است مشخص می‌کند که یک property باید در همه Node های خاص یا رابطه‌های خاص وجود داشته باشد (همان بحث not null که قبلا داشتیم)
- آشنا شدن با قابلیت‌های مختلف Match. علاوه بر یافتن Node هایی که ویژگی خاصی دارند در دیتابیس مذکور می‌توان رابطه‌ها را نیز بررسی کرده و به طور مثال Node ای را که با Node مشخص دیگری ارتباط خاصی دارد برگردانیم. همچین می‌توان تمامی Node های موجود را برگرداند تا دید کلی نسبت به شمای کلی داشته باشیم (با مشخص نکردن Where و Return کردن همه موارد)
- آشنا شدن با نحوه کار با Csv ها و خواندن دیتا از آنان. در صورتی که حجم فایل اندک باشد این کار با استفاده از دستورهای Load و Create انجام شده اما درصورتی که حجم فایل بالا باشد استفاده کردن از Periodic commit باعث می‌شود که به طور مثال با خواندن هر ۱۰۰۰ ردیف موارد خوانده شده Commit شده تا memory overhead کمتر شود.
- آشنا شدن با نحوه drop کردن یک index یا constraint که با مشخص کردن Node یا رابطه مذکور به راحتی انجام می‌شود
- آشنا شدن با نحوه Delete کردن یک Node یا رابطه که با کمک گرفتن از Match انجام می‌شود.

بخش آموزش خود سایت و زبان cypher:

کووری که در عکس زیر مشاهده می‌کنید به دنبال یک Person گشته که فیلد name آن عبارت Tom Hanks بوده و بعد از یافتن این فرد آن را با Return نمایش می‌دهد.

```
neo4j$ MATCH (tom:Person) WHERE tom.name = "Tom Hanks" RETURN tom
```

Graph
Table
Text
Code

tom

1


```
{
  "identity": 79,
  "labels": [
    "Person"
  ],
  "properties": {
"name": "Tom Hanks",
"born": 1956
  }
}
```

کووری زیر نیز تقریبا کار مشابهی انجام می‌دهد با این تفاوت که دیگر از عبارت Where استفاده نکرده و شرطی که برای یافتن یک entity از جنس Movie که title آن (Could Atlas) می‌باشد را درون خود Match بیان کرده است که از لحاظ کارکردی با مدل قبل تفاوت ندارد.

```
1 MATCH (cloudAtlas:Movie {title: "Cloud Atlas"})
2 RETURN cloudAtlas
```

Graph
Table
Text
Code

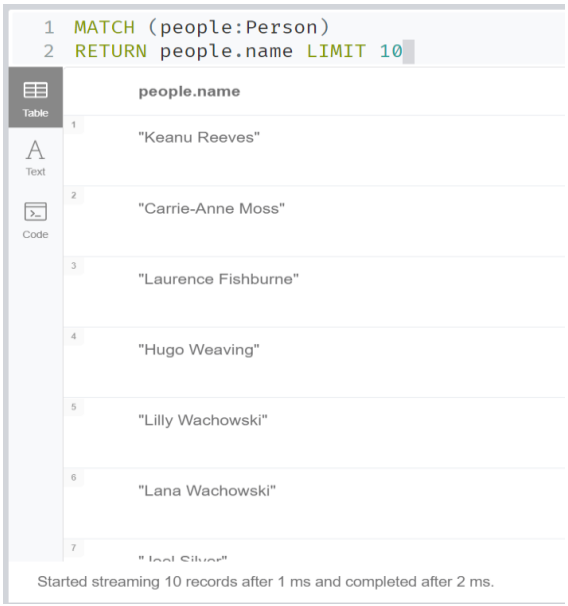
cloudAtlas

1


```
{
  "identity": 113,
  "labels": [
    "Movie"
  ],
  "properties": {
"tagline": "Everything is connected",
"title": "Cloud Atlas",
"released": 2012
  }
}
```


Started streaming 1 records after 1 ms and completed after 2 ms.

کووری زیر نام ۱۰ تا از Person هار را پیدا می‌کند. با استفاده از Match تمام Person ها در نظر گرفته شده و در قسمت Return که برای نشان دادن نتیجه کاربرد دارد نام این Person ها نمایش داده شده با این مورد که با استفاده از Limit این تعداد به ۱۰ محدود شده است. همچنین چون مقدار property بازگردانده شده و نه خود Node ها خروجی به صورت جدول در دسترس است و شکل نخواهیم داشت.



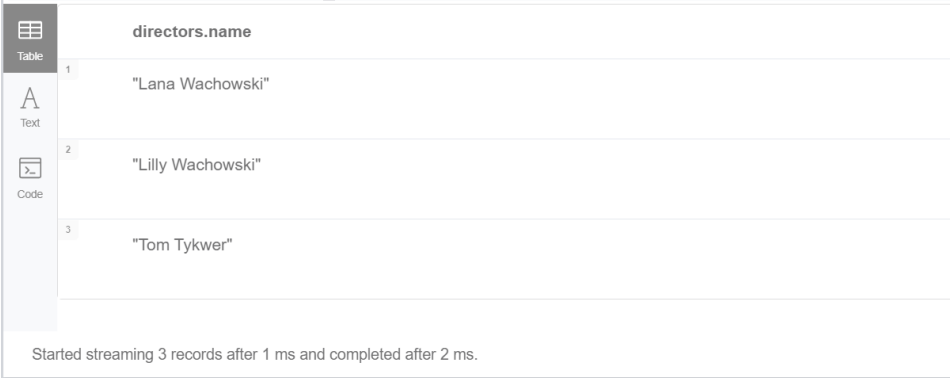
در کووری زیر ابتدا مشخص شده است که nineties از جنس Movie بوده و بعد در عبارت Where مشخص شده است که Movie هایی که (همان nineties) سال انتشار آنان بزرگتر از ۱۹۹۰ و کوچکتر از ۲۰۰۰ بوده (دهه ۹۰ میلادی) را انتخاب کن. در آخر نیز آن دسته که ویژگی بالا را دارند را در نظر گرفته و title آنان را چاپ کن. همچنین چون مقدار property بازگردانده شده و نه خود Node ها خروجی به صورت جدول در دسترس است و شکل نخواهیم داشت.

در کووری زیر در ابتدا Person که نام آن Tom Hanks بوده را مشخص کرده و بعد هم تمامی Node ها از جنس Movie که این فرد با آنان رابطه Acted_in دارد را مشخص کرده و در آخر این فیلم‌ها و آن فرد را برگردانیم. یعنی در آخر Node فرد با نام Tom Hanks و تمامی فیلم‌هایی که این فرد در آنان بازی کرده است را برمی‌گردانیم. همچنین چون در تنظیمات دیتابیس مشخص شده است که result nodes به صورت connected باشند پس در شکل نهایی Node تام هنکس به Node های فیلم‌ها یال‌های جهت‌دار دارد. (در خود کووری نیز همانطور که معلوم است فلش از سمت تام هنکس به فیلم‌ها می‌باشد)

```
1 MATCH (tom:Person {name: "Tom Hanks"})-[:ACTED_IN]→(tomHanksMovies)
2 RETURN tom,tomHanksMovies
```

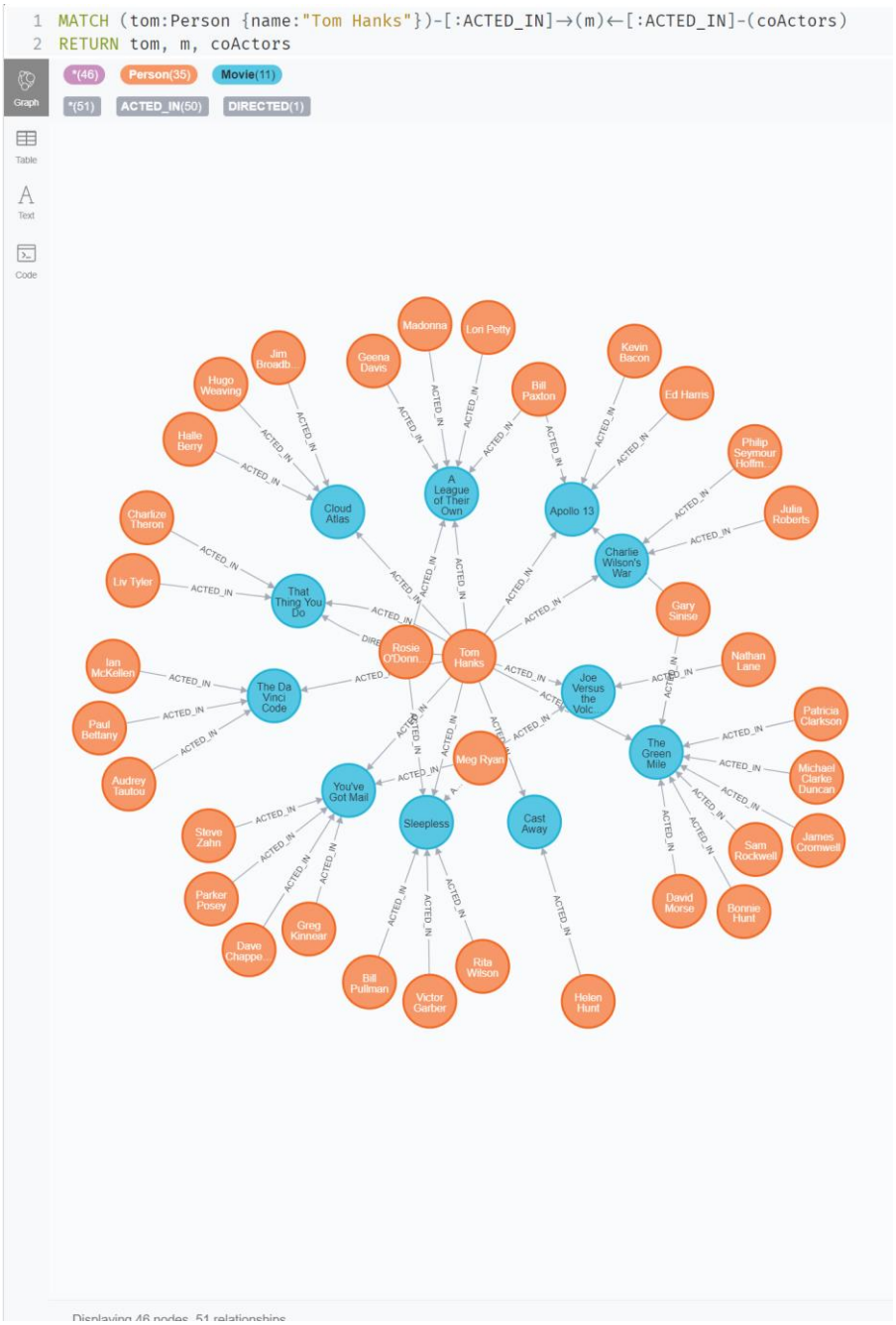
در کووری زیر ابتدا cloudAtlas که از جنس Movie بوده توسط فیلد title مشخص شده (یعنی بدون استفاده از Where مشخص کرده‌ایم) و بعد directors نیز طوری مشخص شده اند که شامل Node هایی باشد که رابطه directed با فیلم مذکور داشته باشد. (فلش از سمت directors به سمت فیلم می‌باشد). در آخر نیز فیلد نام این directors چاپ شده است. به طور خلاصه نام کارگردان‌های فیلم با نام cloudAtlas آورده شده و بازهم چون فیلد را برمیگردانیم خروجی به صورت جدول می‌باشد و Graph view نداریم.

```
1 MATCH (cloudAtlas:Movie {title: "Cloud Atlas"})←[:DIRECTED]-(directors)
2 RETURN directors.name
```



در کووری زیر مانند مثال‌های مشابه قبلی ابتدا Person با نام Tom Hanks مشخص شده و بعد تمام فیلم‌هایی که این فرد در آنان بازی کرده است (رابطه Acted_in از سمت تام هنکس به سمت فیلم‌ها) مشخص شده (در متغیر m) و بعد از آن نیز تمام افرادی که با فیلم‌ها رابطه مشابه Acted_in دارند نیز در coAactors مشخص شده‌اند. در آخر نیز Node های مربوط به تام هنکس, فیلم‌هایی که در آن بازی کرده و تمامی افراد دیگری که در این فیلم‌ها بازی کرده‌اند بازگردانده شده‌اند. به طور خلاصه یعنی فرد با نام Tom Hanks و تمام فیلم‌هایی که بازی کرده و به ازای هر کدام از آن فیلم‌ها تمام افراد دیگری که در آن فیلم بازی کرده‌اند مشخص شده‌اند. به نوعی یعنی فیلم‌ها و هم‌بازی‌های Tom Hanks مشخص شده‌اند.

```
1 MATCH (tom:Person {name:"Tom Hanks"})-[:ACTED_IN]→(m)←[:ACTED_IN]-(coActors)
2 RETURN tom, m, coActors
```



در کووری زیر تمامی افرادی از جنس Person که هر نوع رابطه‌ای با Movie که نام آن Cloud Atlas می‌باشد دارند مشخص شده و پس از آن نام این افراد و نوع رابطه‌ای که با فیلم مذکور دارند و اطلاعات آن رابطه بازگردانده شده است. اطلاعات آن رابطه یعنی همان property های آن و نوع رابطه نیز با type آن مشخص می‌شود. به طور مثال اولین ردیف این جدول نویسنده نام نویسنده فیلم و اطلاعاتی درباره آن رابطه در اختیار ما قرار می‌دهد. بازهم چون فیلد را برمیگردانیم خروجی به صورت جدول می‌باشد و Graph view نداریم.

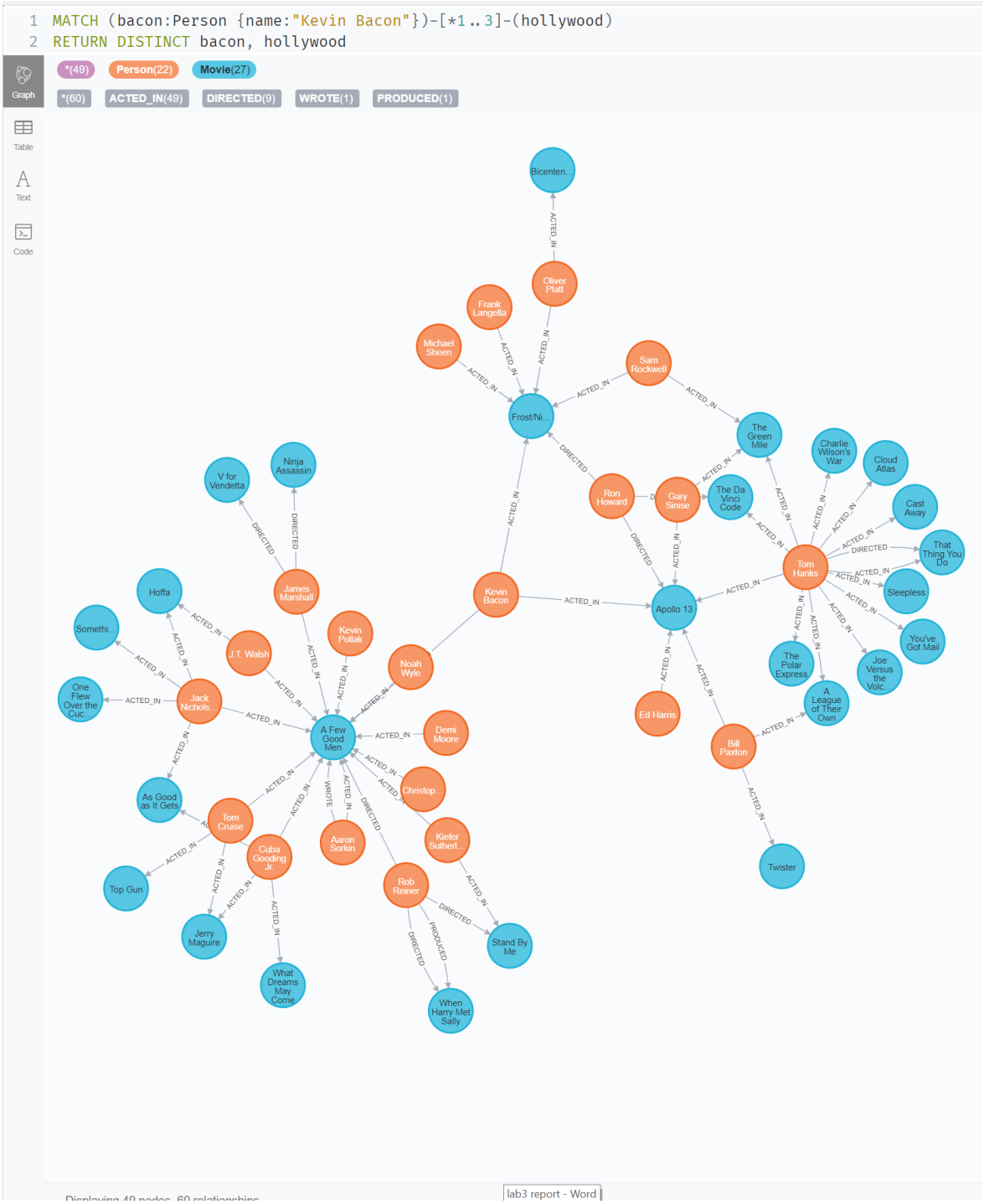
```
1 MATCH (people:Person)-[relatedTo]-(:Movie {title: "Cloud Atlas"})
2 RETURN people.name, type(relatedTo), relatedTo
```

	people.name	type(relatedTo)	relatedTo
1	"David Mitchell"	"WROTE"	{ "identity": 154, "start": 117, "end": 113, "type": "WROTE", "properties": { } }
2	"Lana Wachowski"	"DIRECTED"	{ "identity": 153, "start": 14, "end": 113

Started streaming 10 records after 1 ms and completed after 2 ms.

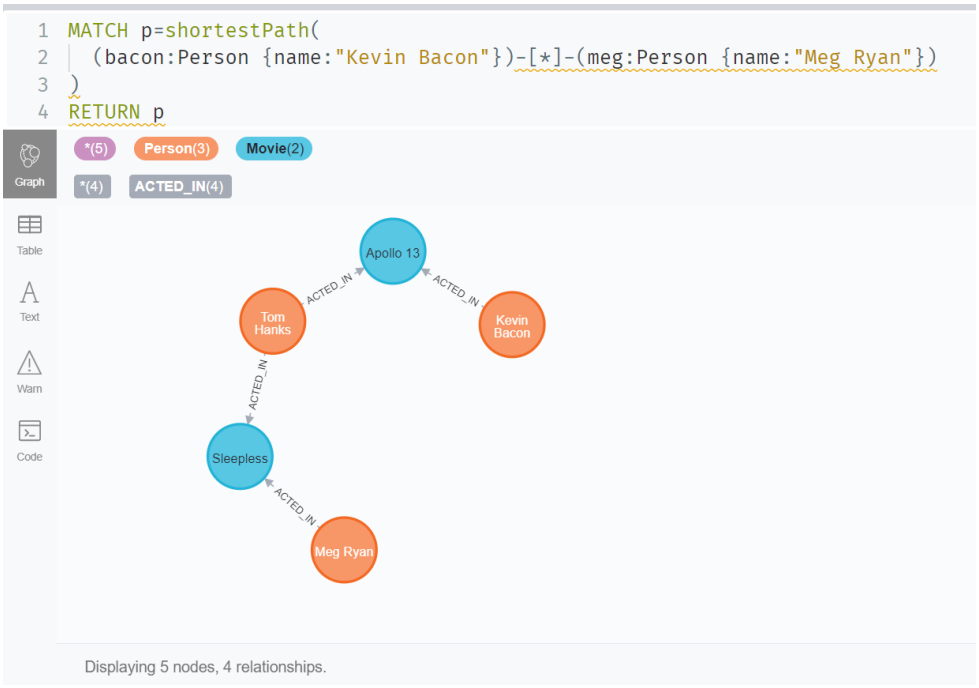
کووری که در صفحه بعد مشاهده می‌کنید در ابتدا bacon را مشخص کرده که از جنس Person بوده و فیلد name آن برابر Kevin Bacon بوده (این کار بدون استفاده از Where و در خود Match انجام شده است) و بعد از آن تمامی Entity هایی (از جنس Person یا از جنس Movie) که ارتباط آنان با Bacon بین ۱ تا ۳ جهش فاصله دارد مشخص شده‌اند. این کووری درواقع یک کاربرد از کووری shortest path که با نام Bacon Path می‌باشد را نشان می‌دهد. لازم به ذکر است که موقع بازگرداندن نتایج از عبارت Distinct استفاده شده تا هر Node صرفا یکبار چاپ شود. شاید یک سری از Node ها بیشتر از یک مسیر به Bacon داشته باشند و در صورتی که از Distinct استفاده نشود ممکن است Node هایی که بیش از یک مسیر (به طول ۱ تا ۳) به Bacon دارند بیشتر از یکبار چاپ شوند.

```
1  MATCH (bacon:Person {name:"Kevin Bacon"})-[*1..3]-(hollywood)
2  RETURN DISTINCT bacon, hollywood
```



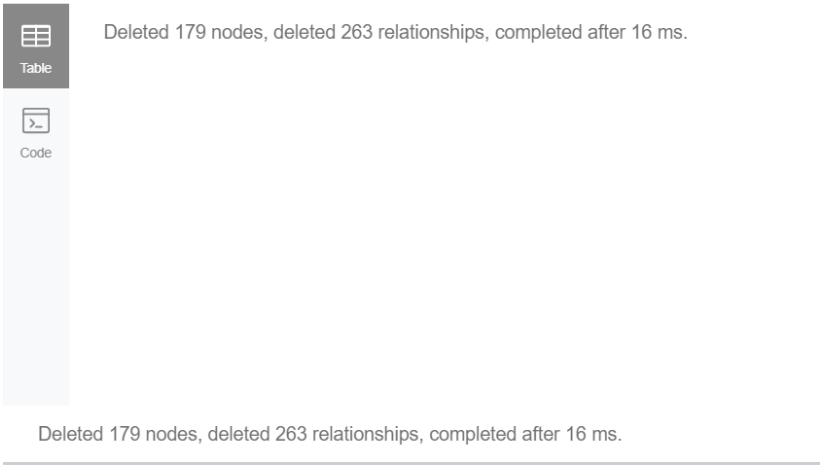
در کووری زیر از تابع shortestPath استفاده شده است که در آن دو Node مشخص شده و با استفاده از [*] مسیر شامل کمترین یال‌ها بین این دو Node مشخص می‌شود. در این کووری کوتاه‌ترین مسیر بین Node با نام bacon که درواقع Person با فیلد نام Kevin Bacon بوده و Node با نام meg که درواقع Person با فیلد نام Meg Ryan می‌باشد مشخص شده است. بازگرداندن این مسیر نیز مسیر مورد نظر شامل نودها, یال‌ها و جهت‌هایشان را به ما نشان می‌دهد. لازم به ذکر است که محاسبه این کووری که با [*] مشخص شده در دیتابیس‌های حجیم ممکن است که از لحاظ زمانی اندکی طول بکشد.

```
MATCH p=shortestPath(
  (bacon:Person {name:"Kevin Bacon"})-[*]-(meg:Person {name:"Meg Ryan"})
)
RETURN p
```

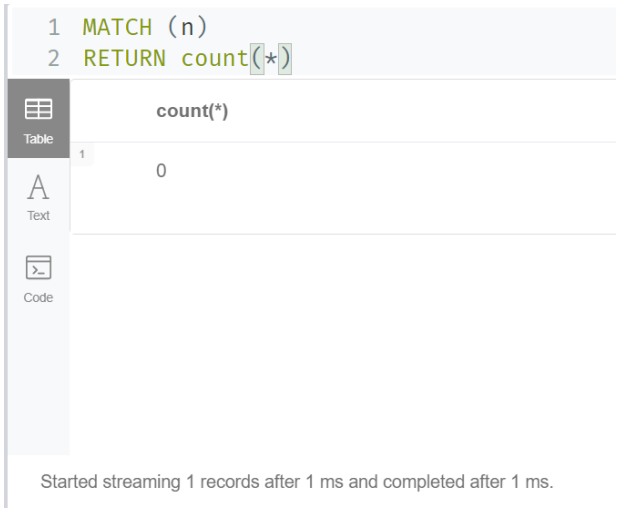


در کووری زیر با استفاده از Match تمامی Node های موجود در دیتابیس مذکور مشخص شده و بعد با استفاده از Detach و Delete این نودها و ارتباطات بین آنان پاک می‌شوند.

```
MATCH (n)
DETACH DELETE n
```



در کووری زیر تمامی Node های موجود در دیتابیس مشخص شده و بعد از آن تعداد این نودها برگردانده شده که مشاهده می‌شود هیچ نودی باقی نمانده است زیرا در عکس قبلی تمامی نودها و رابطه‌های بین آنان را پاک کرده‌ایم.

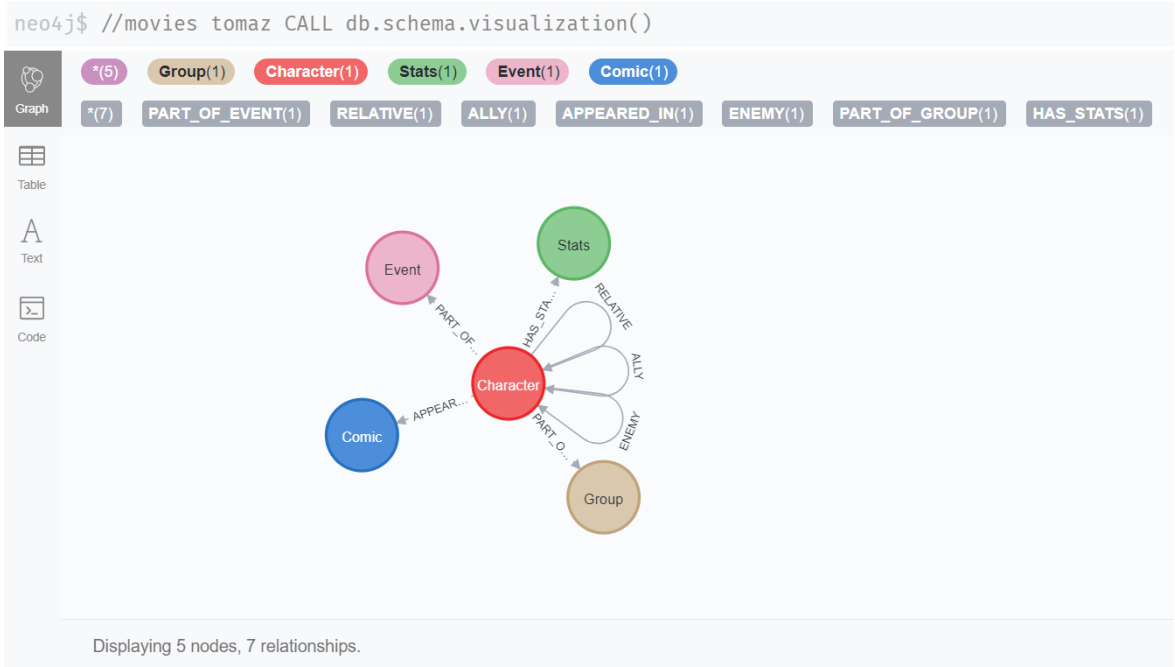


بخش کار روی مقاله انتخابی:

مقاله marvel-universe که به بررسی قهرمان‌های مارول و رابطه بین آنان پرداخته انتخاب شده است. این مقاله در اصل به این صورت شکل گرفته که نویسنده یک مدخل در سایت Kaggle دیده که درباره دنیای مارول بود. متاسفانه در آن دیتاست که آنجا قرار داشت فقط کمیک‌ها و کاراکترها دارای matching ID هستند و بقیه دیتاست طوری نیست که بتوان یک گراف کامل که اجزای آن باهم به درستی مرتبط هستند درست کرد. سپس نویسنده با استفاده کردن از Marvel API داده‌های لازم را جمع‌آوری کرده و بقیه مواردی که توسط API قابل گرفتن نبودند (مانند اطلاعات درباره کاراکترها) توسط Web crawling روی خود سایت مارول جمع‌آوری شده است. پس از لود کردن دیتاست نیز تحلیل‌های روبرو انجام شده است: بررسی event هایی که بیشترین تعداد کاراکتر در آنان بوده‌اند (که در آن مشاهده شده دوره یکی از event ها اشتباه بوده و به خود marvel اطلاع داده شده) / بررسی گروه‌ها و مرتب‌سازی بر اساس تعداد اعضای گروه و غیره (که سر هر تکه کد توضیح داده شده است)

نحوه import داده‌ها نیز از طریق ۱۰ بخش Load CSV می‌باشد که آدرس فایل‌ها نیز در گیت‌هاب است. در هنگام لود کردن دیتاست‌ها ارتباطات (خارجی) نیز شکل داده‌شده است. (به طور مثال عوض گروه بودن و یا عضو رویداد بودن). با استفاده از apoc.assert نیز مشخص شده است که هر موجودیت چه فیلدی را به عنوان کلید استفاده می‌کند. لینک روبرو کد import کردن را نشان می‌دهد : gist.github

کووری زیر با استفاده از schema و تابع visualization تمامی Node هایی که در دیتابیس وجود داشته به همراه ارتباطات مختلفی که برای هر کدام نسبت به یکدیگر (یا حتی نسبت به Node های هم‌جنس) تعریف شده است را نشان می‌دهد. به طور مثال Node مربوط به کاراکتر سه رابطه درون خود دارد که یعنی هر کاراکتر با کاراکترهای دیگری می‌تواند سه نوع ارتباط مختلف داشته باشد. (آشنا / همرزم / دشمن). همچنین هر کاراکتر یک وضعیت داشته که همان ارتباط با Node با نام Stats می‌باشد (مواردی مانند سرعت و مهارت‌ها) و درون Event عضو می‌باشد. کاراکتر می‌تواند در یک یا چند Comic ظاهر شده و عضو یک Group باشد.



در قسمت بعدی با استفاده از کتابخانه apoc که در این مقاله استفاده شده است تابع stats را فراخوانی می‌کنیم. این تابع یک summary از اطلاعات مختلف دیتابیس مانند تعداد Node یا رابطه یا نوع روابط و غیره می‌دهد. بازگرداندن label از این تابع به ما تعداد اعضای موجود از هر Node را می‌دهد. یعنی تعداد به اعضای هر label که اعداد در شکل زیر قابل مشاهده هستند.

```
1 CALL apoc.meta.stats() YIELD labels
2 return labels
```

Table | Text | Code

labels

```
{
  "Stats": 470,
  "Group": 92,
  "Event": 74,
  "Character": 1105,
  "Comic": 38875
}
```

Started streaming 1 records after 2 ms and completed after 6 ms.

در قسمت بعدی می‌خواهیم که کاراکترهایی که بیشترین تعداد ظاهر شدن در کامیک‌ها را دارند بیابیم. ابتدا c از جنس Character را مشخص کرده و بعد برای بازگرداندن در ابتدا نام کاراکتر و جلوی آن نیز size (همان تعداد) روابطی که آن کاراکتر به شکل Appeared_in دارد را بازمیگردانیم و اسم بخش دوم خروجی را هم comics میزاریم (ربطی به موجودیت comic ندارد). البته خروجی بر اساس تعداد روابط که با comics مشخص شده است به صورت نزولی مرتب شده (order by) و صرفا ۵ ردیف اول آن (Limit) نوشته شده است.

```
1 MATCH (c:Character)
2 RETURN c.name as character,
3        size((c)-[:APPEARED_IN]→()) as comics
4 ORDER BY comics DESC
5 LIMIT 5
```

Table | Text | Code

	character	comics
1	"Spider-Man (1602)"	3357
2	"Tony Stark"	2354
3	"Logan"	2098
4	"Steve Rogers"	2019
5	"Thor (Marvel: Avengers Alliance)"	1547

Started streaming 5 records after 2 ms and completed after 41 ms.

در قسمت بعدی بررسی روی Event ها انجام شده است. این بررسی پرجمعیت ترین Event ها را به ترتیب نشان می‌دهد. در ابتدا e که از جنس Event می‌باشد در Match در نظر گرفته شده و بعد به ازای ردیف ۵ ستون ایجاد شده است. اولین ستون نام Event و در ستون بعدی تعداد کاراکترهایی که با آن Event رابطه PART_OF_EVENT دارند (فلش از سمت کاراکتر به Event بوده) و این تعداد نیز که با Size محاسبه شده است نامش count_of_heroes است. ستون بعدی date آغاز آن Event و ستون بعدی هم date پایان آن (هر دو property از خود Event هستند). در آخر نیز توضیح Event که property می‌باشد نوشته شده. نتایج مانند مثال قبلی به صورت نزولی و طبق تعداد کاراکترها مرتب شده و صرفا ۵ تا از پرجمعیت ترین Event ها نمایش داده شده اند. لازم به ذکر است که بعد از بررسی مشخص شد که زمان رخداد Acts of Vengeance اشتباه بوده (در خود Marvel API نیز اشتباه است) و این موضوع به Marvel اطلاع داده شده است.

```
MATCH (e:Event)
RETURN e.title as event,
       size((e)<-[:PART_OF_EVENT]-()) as count_of_heroes,
       e.start as start,
       e.end as end,
       e.description as description
ORDER BY count_of_heroes DESC
LIMIT 5
```

Table | Text | Code

	event	count_of_heroes	start	end	description
1	"Fear Itself"	132	"2011-04-16 00:00:00"	"2011-10-18 00:00:00"	"The Serpent, God of Fear and brother to the Allfather Odin, rises to challenge can Thor, Captain America, Iron Man and the Avengers turn back the tide of fea
2	"Dark Reign"	128	"2008-12-01 00:00:00"	"2009-12-31 12:59:00"	"Norman Osborn came out the hero of Secret Invasion, and now the former Gre his own image? And what has become of the heroes?"
3	"Acts of Vengeance!"	93	"1989-12-10 00:00:00"	"2008-01-04 00:00:00"	"Loki sets about convincing the super-villains of Earth to attack heroes other tha
4	"Secret Invasion"	89	"2008-06-02 00:00:00"	"2009-01-25 00:00:00"	"The shape-shifting Skrulls have been infiltrating the Earth for years, replacing
5	"Civil War"	86	"2006-07-01 00:00:00"	"2007-01-29 00:00:00"	"After a horrific tragedy raises questions on whether or not super heroes should McNiven split the Marvel Universe in two as friend fights friend in one of the mc

Started streaming 5 records after 8 ms and completed after 78 ms.

در کووری که در صفحه بعدی مشاهده می‌کنید ابتدا در عبارت که در Match هست g از جنس گروه مشخص شده و سپس برای بازگرداندن نتایج به ازای هر گروه دو ویژگی را نوشته‌ایم. در ابتدا نام آن گروه که property گروه نیز می‌باشد مشخص شده است و در گام دوم نیز تعداد کاراکترهای که درون آن گروه هستند با استفاده از بررسی رابطه PART_OF_GROUP که جهت‌دار به سمت گروه می‌باشد مشخص شده است (یعنی گروه مذکور در چند رابطه با نام ذکر شده شرکت کرده است) و در آخر نیز تعداد این روابط با استفاده از size مشخص شده است و نام این تعداد نیز members گذاشته شده. نتایج مانند مثال‌های قبلی به صورت نزولی و از روی تعداد عضوها مرتب شده و صرفا ۵ تا نتیجه اول چاپ شده است.

```
MATCH (g:Group)
RETURN g.name as group,
       size((g)<-[:PART_OF_GROUP]-()) as members
ORDER BY members DESC LIMIT 5
```

	group	members
1	"X-Men"	41
2	"Avengers"	31
3	"Defenders"	26
4	"Next Avengers"	14
5	"Guardians of the Galaxy"	12

Started streaming 5 records after 2 ms and completed after 11 ms.

مثال بعدی مشخص می‌کند که آیا افرادی درون یک گروه هستند که باهم دشمن باشند یا خیر. در عبارت Match دو کاراکتر مشخص شده اند که هر دو در رابطه PART_OF_GROUP با یک گروه g باشند (یعنی در یک گروه باشند - فلش از کاراکتر به سمت گروه). سپس در عبارت Where بررسی شده است که این دو کاراکتر با هم در یک رابطه ENEMY باشند و id مشابهی نداشته باشند (همان id یکی کوچکتر باشد). در آخر نیز نام هر دو کاراکتر و گروهی که مشترکا در آن عضو هستند به عنوان خروجی داده شده است. مشاهده می‌شود که کلا ۵ مورد وجود دارد که دو کاراکتر هم‌گروه باهم رابطه دشمنی داشته باشند و همه این موارد هم در گروه X-Men رخ داده است.

```
MATCH (c1:Character)-[:PART_OF_GROUP]->(g:Group)
<-[:PART_OF_GROUP]-(c2:Character)
WHERE (c1)-[:ENEMY]-(c2) and id(c1) < id(c2)
RETURN c1.name as character1, c2.name as
character2, g.name as group
```

	character1	character2	group
1	"Logan"	"Sabretooth (House of M)"	"X-Men"
2	"Logan"	"Mystique (House of M)"	"X-Men"
3	"CAIN MARKO JUGGERNAUT"	"Logan"	"X-Men"
4	"CAIN MARKO JUGGERNAUT"	"Storm (Marvel Heroes)"	"X-Men"
5	"Rogue (X-Men: Battle of the Atom)"	"Warren Worthington III"	"X-Men"

Started streaming 5 records after 4 ms and completed after 492 ms.

در گام بعدی که در شکل زیر قابل مشاهده است می‌خواهیم کاراکترهایی که origin آنان Yogoslavia می‌باشد را مشخص کنیم. برای اینکار ابتدا در Match کاراکتر c از جنس کاراکتر را در نظر گرفته و در عبارت Where شرط میگذاریم که property مربوط به place_of_origin آن کاراکتر شامل عبارت Yogoslavia باشد. در آخر نیز برای چاپ کردن نتایج ابتدا نام کاراکتر سپس نام کامل place_of_origin و در آخر نیز aliases آن کاراکتر که همان Nickname های وی می‌باشد چاپ شده اند.

```
MATCH (c:Character)
WHERE c.place_of_origin contains "Yugoslavia"
RETURN c.name as character,
       c.place_of_origin as place_of_origin,
       c.aliases as aliases
```

Table | Text | Code

	character	place_of_origin	aliases
1	"Purple Man"	"Rijeka, Yugoslavia"	"Killgrave the Purple Man, Killy"
2	"Abomination (Ultimate)"	"Zagreb, Yugoslavia"	"Agent R-7, the Ravager of Worlds"

Started streaming 2 records after 2 ms and completed after 39 ms.

در کووری زیر مانند قبل کاراکتر c در Match در نظر گرفته شده و در عبارت Where شرط شده است که تحصیلات کاراکتر که property می‌باشد شامل مدرک دکتری یا همان Ph.D. باشد. در آخر نیز برای چاپ کردن نتایج ابتدا نام آن کاراکتر و سپس تحصیلات کامل ولی چاپ شده اند. لازم به ذکر است که نتایج طبق Limit که گذاشته شده است به ۱۰ مورد محدود شده‌اند.

```
MATCH (c:Character)
WHERE c.education contains "Ph.D"
RETURN c.name as character, c.education as education
LIMIT 10
```

	character	education
1	"UNKNOWN ACHEBE"	"Ph.D. in Law (Yale), degrees in Psychology, Political Science and Divinity"
2	"PROFESSOR MENDEL STROMM MENDEL STROMM"	"Ph.D. in robotics"
3	"FRANKLIN HALL GRAVITON"	"Ph.D. in physics"
4	"Morbius"	"Ph.D in Biochemistry"
5	"Tony Stark"	"Ph.Ds in physics and electrical engineering"
6	"Hulk-dok"	"Ph.D in nuclear physics and two other fields"
7	"Sasquatch (Walter Langkowski)"	"Ph.D. in physics from the Massachusetts Institute of Technology; Bachelors degree from Pennsylvania State University"
8	"Professor X (Ultimate)"	"Ph.Ds in genetics, biophysics, psychology, and anthropology, and an M.D. in psychiatry"
9	"Klaw"	"Ph.D. in physics, bachelor's degree in geology"
10	"High Evolutionary"	"Uncompleted Ph.D at Oxford University"

در کووری زیر مانند قبل کاراکتر c در Match در نظر گرفته شده و بعد برای بازگرداندن ابتدا نام آن کاراکتر که property می‌باشد و بعد تعداد allies های آن که با استفاده از بازگرداندن size آن روابطی که که c یک طرف ALLY بوده (در واقع فلش از سمت c به بیرون می‌باشد) و بعد از آن تعداد تعداد دشمن‌ها با استفاده از بازگرداندن size آن روابطی که که c یک طرف ENEMY بوده (در واقع فلش از سمت c به بیرون می‌باشد) و بعد هم تعداد آشنایان یا همان relative ها با استفاده از بازگرداندن size آن روابطی که که c یک طرف RELATIVE بوده (در واقع فلش از سمت c به بیرون می‌باشد) و بعد هم نتایج طبق جمع این سه فیلد (تعداد همه) به صورت نزولی مرتب شده و مانند قبل نیز تعداد نتایج با استفاده از LIMIT به تعداد ۵ تا محدود شده است.

```
neo4j$ MATCH (c:Character) RETURN c.name as name, size((c)-[…
```

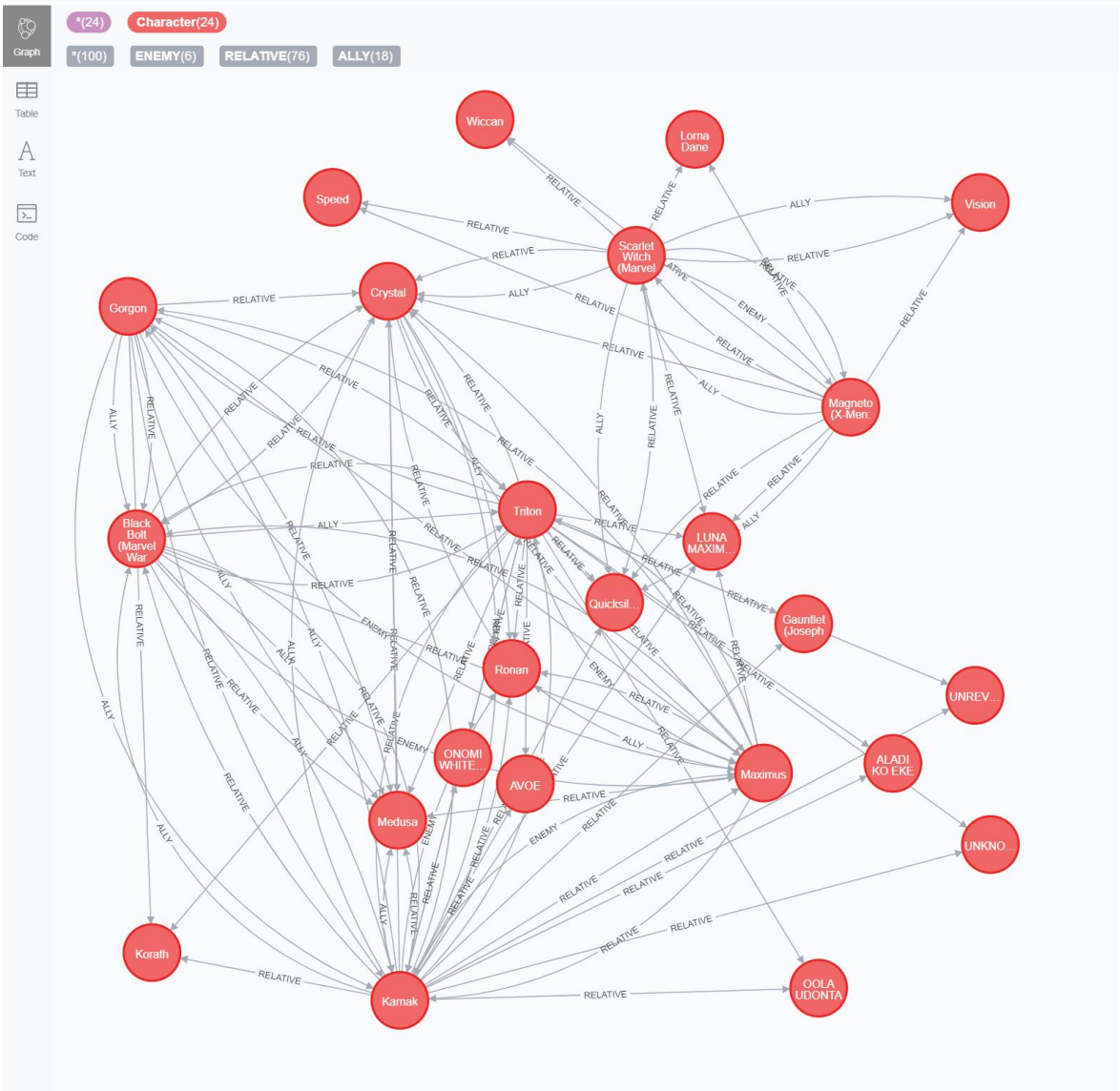
Table | Text | Code

	name	allies	enemies	relative
1	"Scarlet Witch (Marvel Heroes)"	16	14	8
2	"Thor (Marvel: Avengers Alliance)"	9	14	10
3	"Invisible Woman (Marvel: Avengers Alliance)"	13	10	7
4	"Logan"	14	10	5
5	"Karnak"	6	2	17

Started streaming 5 records after 1 ms and completed after 8 ms.

کووری که در صفحه بعدی مشاهده می‌کنید بررسی روی community از ارتباطاتی هستش که کاراکتر با نام Triton دارد. ابتدا در عبارت Match این کاراکتر با استفاده از property نام آن مشخص شده است (بدون استفاده از Where) و سپس از تابع path.subgraphAll که در apoc می‌باشد استفاده شده است. این تابع که hyperparameter های زیادی نیز دارد Node هایی که از یک Node مشخص شده (در اینجا کاراکتر با نام Triton) قابل رسیدن هستند (توسط یک سری رابطه که به طور مثال در اینجا رابطه RELATIVE بودن توسط پارامتر relationshipFilter مشخص شده است) به عنوان یک گراف کشیده می‌شوند. موارد دیگر مانند حداکثر عمق که از نود اصلی خواهیم داشت و موارد پیچیده‌تری را نیز می‌توان مشخص کرد. پس از صدا کردن این تابع با استفاده از عبارت YIELD نودهای این گراف و روابطی که باهم دارند (نوع روابط روی یال‌ها نوشته شده است) برگردانده شده و برای چاپ کردن این نتایج نیز روی خروجی دقیقا همین موارد را Return می‌کنیم. Documentation کامل مربوط به تابع استفاده شده در این قسمت به آدرس doc قابل دسترسی می‌باشد.

```
MATCH p=(c:Character{name:"Triton"})
CALL apoc.path.subgraphAll(id(c), {relationshipFilter:"RELATIVE"})
YIELD nodes, relationships
RETURN nodes, relationships
```



در کووری صفحه بعدی از کتابخانه gds استفاده شده است و از تابع wcc استفاده شده تا جزایر مختلف در گراف را پیدا کنیم. به این صورت که برای فیلد nodeProjection باید نوع Node هایی که میخواهیم بررسی کنیم وارد کنیم (کاراکتر) و بعد برای فیلد relationshipProjection که بررسی می‌کند برای در نظر گرفتن جزایر مختلف چه رابطه‌ای را در نظر بگیرد نام رابطه (هم‌رزم بودن) را وارد می‌کنیم. برای بازگرداندن خروجی این تابع که Call شده است مثل دفعه قبلی از Yield استفاده می‌کنیم که آیدی نودها و بخش‌ها (همان کامپوننت‌ها) را برمی‌گرداند. سپس آیدی هر کامپوننت و تعداد اعضای آن با استفاده از count(*) که همان members می‌باشد در نظر گرفته شده و شرط هم گذاشته‌ایم که صرفا کامپوننت‌هایی که بیشتر از یک عضو دارد باشند (حداقل یک رابطه در کامپوننت باشد). برای نمایش دادن هم با Return آیدی کامپوننت و تعداد اعضا را برگردانده و نتایج را طبق تعداد اعضا به شکل نزولی مرتب کرده و صرفا ۵ عضو اول را چاپ می‌نماییم. مشاهده می‌شود که بزرگترین جزیره شامل ۱۹۵ تا از کاراکترهای موجود در دیتابیس می‌باشد.

```
CALL gds.wcc.stream({
  nodeProjection:'Character',
  relationshipProjection:'ALLY'})
YIELD nodeId, componentId
WITH componentId, count(*) as members
WHERE members > 1
RETURN componentId, members
ORDER BY members DESC
LIMIT 5
```

Table | Text | Code

	componentId	members
1	0	195
2	26	4
3	245	3
4	6	2
5	2	2

بعد از کووری قبلی در مقاله ذکر شده چند قسمتی به کد پایتون اختصاص داده شده که با هماهنگی استاد این بخش‌ها را skip کرده و ادامه کار را از بخش بعدی کدهای دیتابیس اصلی آزمایشگاه ۳ انجام می‌دهیم.

در کووری صفحه بعدی ابتدا c که از جنس کاراکتر می‌باشد مشخص شده و سپس با استفاده از رابطه HAS_STATS نودهایی از جنس Stats که با کاراکتر در ارتباط هستند نیز در نظر گرفته شده. بعد برای دادن خروجی ابتدا نام آن کاراکتر که property می‌باشد تحت عنوان "character" چاپ شده و روبروی آن نیز با استفاده از تابع avg که مربوط به کتابخانه apoc می‌باشد میانگین stat های موجود در نود STAT مربوط به آن کاراکتر محاسبه شده است. یعنی در خود نود STAT چند area مانند قدرت یا سرعت یا مثلا انرژی داریم که هر یک از این موارد یک امتیازی بین ۰ تا ۷ دارند. (به طور مثال مانند یک dictionary در پایتون) و این تابع با در نظر گرفتن مقداریا همان value و کلید یا همان key اقدام به محاسبه میانگین این مقادیر که در کل STATS موجود هستند می‌کند و نام این قسمت نیز "average_stats" در نظر گرفته شده. سپس ردیف‌ها طبق همان میانگین به صورت نزولی مرتب شده و در آخر نیز نتایج محدود به ۱۰ نتیجه طبق Limit شده اند. مشاهده می‌شود که به طور مثال چند نتیجه اول میانگین ۷ از ۷ دارند یعنی تمامی فیلدهای STAT آنان روی بالاترین حالت خود (همان ۷) می‌باشد.

```
MATCH (c:Character)-[:HAS_STATS]→(stats)
RETURN c.name as character,
       apoc.coll.avg(apoc.map.values(stats,
keys(stats))) as average_stats
ORDER BY average_stats DESC
LIMIT 10
```

	character	average_stats
1	"Sasquatch (Walter Langkowski)"	7.0
2	"Squirrel Girl"	7.0
3	"Galactus"	7.0
4	"Deathstrike (Ultimate)"	7.0
5	"GRAYDON CREED"	7.0
6		

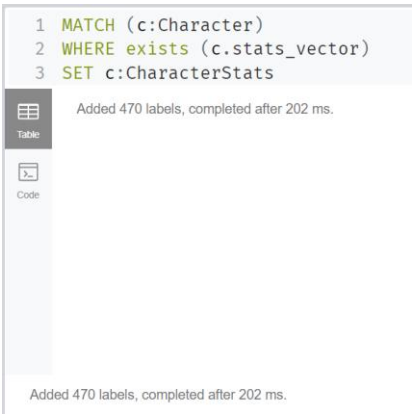
در سلسله تکه کدهای زیر کووری خاصی انجام نمی‌شود اما قصد داریم تا نوعی از تابع Knn را پیاده‌سازی کنیم تا در ادامه با کمک گرفتن از آن در کنار تابع Louvain Modularity نوعی clustering از کاراکترهای موجود در دیتابیس داشته باشیم. در گام اول که در شکل زیر قابل مشاهده می‌باشد می‌خواهیم که Stats مربوط به هر کاراکتر را به عنوان یک stats_vector خروجی بدهیم. البته لازم به ذکر است که برای نرمال کردن و مشابه‌سازی قدرت‌های مختلف به قابلی flight که تا الان به صورت ۱ تا ۰ بوده اعداد ۰ و ۷ را نسبت می‌دهیم تا scale آن مانند قدرت‌های دیگر شود. در این کد بعد از مشخص کردن کاراکتر c و s که از جنس Stats می‌باشد یک آرایه از فیلدهای مختلف Stats ساخته و آنرا به عنوان یک property جدید در همان کاراکتر c ذخیره می‌کنیم (با استفاده از Set)

```
MATCH (c:Character)-[:HAS_STATS]→(s)
WITH c, [s.durability, s.energy,
s.fighting_skills,
        s.intelligence, s.speed, s.strength,
        CASE WHEN c.flight = 'true' THEN 7 ELSE 0
END] as stats_vector
SET c.stats_vector = stats_vector
```

Set 470 properties, completed after 971 ms.

Set 470 properties, completed after 971 ms.

بعد از ست کردن فیلد stats_vector یک لیبل به آن کاراکترهایی که دارای این property می‌باشند اضافه می‌کنیم و نام این فیلد جدید را نیز ChatacterStats ذخیره می‌کنیم.



حال در کد شکل زیر یک گراف تازه می‌سازیم. این گراف از روی Node های کاراکتر ساخته شده با این تفاوت که Projection روی property های stats_vector و ChatacterStats انجام شده است. لازم به ذکر است که گراف جدید Named می‌باشد.

به عنوان گام آخر این قسمت نیز الگوریتم Knn را فراخوانی می‌کنیم.از مدل mutate این تابع استفاده شده که نتیجه را روی خود گراف ذخیره می‌کند. ۴ پارامتر اصلی این تابع شامل:

* پارامتر topK که تعداد همسایه‌هایی که برای هر Node در نظر گرفته می‌شود و K تا نزدیک‌ترین Node بازگردانده شود. این مقدار در تکه کد زیر عدد ۱۵ در نظر گرفته شده.

* پارامتر sampleRate که نرخ سمپلینگ را برای محدود کردن تعداد مقایسه‌ها به ازای هر Node را ست می‌کند. این مقدار در کد زیر عدد ۰.۸ در نظر گرفته شده.

* deltaThreshold که به صورت درصد بوده و که چه زمانی عمل early stopping انجام شود. این مقدار همان مقدار پیش‌فرض در نظر گرفته شده.

* randomJoins که در هر پیمایش چه تعداد اقدام برای اتصال Node های همسایه و جدید بر اساس انتخاب تصادفی صورت گیرد که . این مقدار همان مقدار پیش‌فرض در نظر گرفته شده

کد زیر صدا کردن تابع Louvain می‌باشد. در کد زیر مشابه بودن (برای یافتن community ها) از روی property هایی که مشخص است محاسبه شده و با استفاده از relationshipWeightProperty مشخص می‌کنیم که می‌بایست وزن رابطه نیز موقع محاسبه ساختار community شبکه در نظر گرفته شود.همچین برای ذخیره کردن نتیجه خروجی روی خود گراف از writeProperty استفاده می‌کنیم.

به عنوان آخرین قسمت نیز community هایی که توسط الگوریتم بالا طبقه‌بندی شده اند را نشان می‌دهیم. در کووری زیر ابتدا کاراکتر و Stats آن توسط رابطه HAS_STATS مشخص شده و برای خروجی دادن ابتدا community که کاراکتر به آن تعلق دارد طبق تابع Louvain محاسبه شده و در مرحله بعد این تعداد اعضای این community با استفاده از count(*) محاسبه می‌شود. میانگین قدرت اعضای موجود در community نیز با استفاده از میانگین property های مختلف stats محاسبه می‌شود. مانند دفعات قبلی قابلیت پرواز کردن که به صورت Boolean ذخیره شده است به صورت ۰ یا ۷ در نظر گرفته خواهد شد.

```
MATCH (c:Character)-[:HAS_STATS]→(stats)
RETURN c.louvain as community, count(*) as members,
       avg(stats.fighting_skills) as fighting_skills,
       avg(stats.durability) as durability,
       avg(stats.energy) as energy,
       avg(stats.intelligence) as intelligence,
       avg(stats.speed) as speed,
       avg(stats.strength) as strength,
       avg(CASE WHEN c.flight = 'true' THEN 7.0 ELSE 0.0 END) as flight
```

	community	members	fighting_skills	durability	energy	intelligence	speed	strength	flig
1	262	94	4.617021276595745	5.72340425531915	5.22340425531915	4.404255319148937	5.25531914893617	4.946808510638298	1.6
2	334	51	2.509803921568627	2.509803921568628	0.803921568627451	2.8235294117647047	1.941176470588235	2.0588235294117645	0.1
3	287	96	3.5625	2.7395833333333344	2.260416666666666	2.9583333333333317	2.3333333333333335	2.4791666666666683	0.5
4	175	36	4.25	5.027777777777778	4.138888888888889	4.166666666666667	3.6111111111111116	4.777777777777778	0.1
5	144	99	3.808080808080809	4.03030303030303	2.7878787878787867	3.242424242424241	3.1111111111111107	3.8282828282828283	0.8

دو تکه کد زیر نیز دقیقا مشابه کدهای قبلی بوده و توضیحات آنان یکی است با این تفاوت که به جای تابع Louvain از تابع labelPropagation استفاده شده است. تعداد community ها و اعضای آن تفاوت‌هایی با مثال قبل خواهد داشت.

```
CALL gds.labelPropagation.write('marvel',
  {relationshipTypes:['SIMILAR'],
   relationshipWeightProperty:'score',
   writeProperty:'labelPropagation'})
```

	writeMillis	nodePropertiesWritten	ranIterations	didConverge	communityCount	communityDistribution	postProcessingMillis	createMillis	compu
1	232	470	10	false	12	{ "p99": 144, "min": 3, "max": 144, "mean": 39.166666666666664, "p90": 70, "p50": 22, "p999": 144, "p95": 70, "p75": 46 }	32	2	1842

```
MATCH (c:Character)-[:HAS_STATS]→(stats)
RETURN c.labelPropagation as community, count(*) as members,
       avg(stats.fighting_skills) as fighting_skills,
       avg(stats.durability) as durability,
       avg(stats.energy) as energy,
       avg(stats.intelligence) as intelligence,
       avg(stats.speed) as speed,
       avg(stats.strength) as strength,
       avg(CASE WHEN c.flight = 'true' THEN 7.0 ELSE 0.0 END) as flight
```

Table Text Code

	community	members	fighting_skills	durability	energy	intelligence	speed	strength	fl
1	218	35	4.457142857142857	4.971428571428571	4.542857142857141	4.428571428571429	4.7714285714285705	4.428571428571429	0.
2	343	144	3.118055555555556	2.8055555555555554	1.9236111111111116	2.965277777777777	2.3263888888888897	2.361111111111111	0.
3	270	18	4.722222222222222	5.333333333333334	4.5	4.499999999999999	4.333333333333332	4.944444444444445	0.
4	118	38	4.473684210526314	3.578947368421052	2.973684210526316	3.0526315789473686	3.3684210526315783	3.5	1.
5	215	15	4.466666666666668	3.4666666666666663	2.066666666666667	3.1999999999999997	3.1333333333333333	4.133333333333333	0.
6									

مشکلات و توضیحات تکمیلی

در بحث کار کردن با مقاله بعضا قسمت‌هایی از کد بود که نیاز به import کردن یک کتابخانه مانند apoc و یا gds بود. متاسفانه خود مقاله نیز توضیح مناسبی روی نحوه import کردن این کتابخانه‌ها نداده و بنده پس از ۳۰ دقیقه گشتن متوجه شدم که ابتدا باید فایل jar این کتابخانه‌ها دانلود شده و سپس تغییراتی در فایل config خود دیتابیس داده شود تا بتوان از این کتابخانه‌هایی که در قسمت plugin کپی و پیست کرده‌ایم استفاده کرد. پیشنهاد می‌شود که در صورت امکان یک لینک کمکی در داک آزمایشگاه برای توضیح نحوه نصب کتابخانه‌های اضافی ضمیمه شود.

آنچه آموختم

آشنایی خوبی با دیتابیس مذکور حاصل شده و پس از انجام این آزمایش می‌توان گفت که تسلط نسبی و در حد مباحث Basic و Midlevel این زبان به‌وجود آمده است.