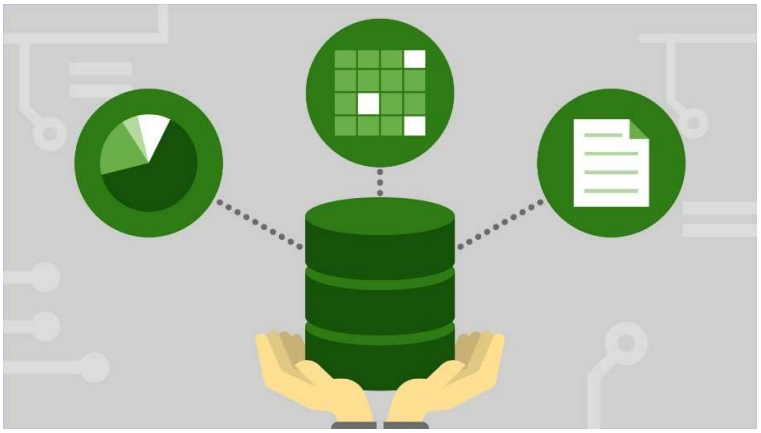
به نام خدا

دانشگاه تهران
پردیس دانشکده‌های فنی
دانشکده برق و کامپیوتر



آزمایشگاه پایگاه داده

دستور کار شماره ۳

پرنیان فاضل
۸۱۰۱۹۸۵۱۶

بهار ۱۴۰۲

مقدمه‌ای بر Neo4j

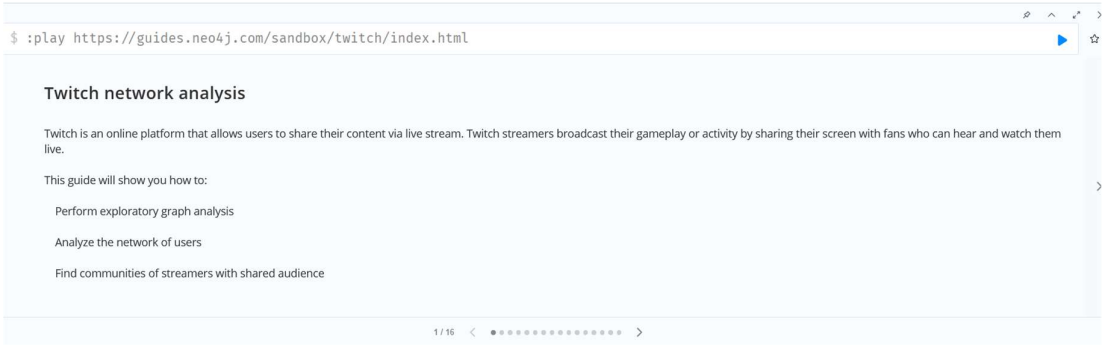
Neo4j یک سیستم مدیریت پایگاه داده گراف است که به ما امکان مدل سازی و ذخیره داده‌ها را در قالب نمودار می‌دهد. یکی از مزیت‌های Neo4j زبان پرس‌وجو قدرتمند آن، Cypher است که به ما امکان می‌دهد کوئری‌های گراف پیچیده را انجام دهیم.

بخش ۱: Twitch network analysis

ابتدا به سایت https://demo.neo4jlabs.com:7473/ مراجعه کرده و به دیتابیس کانکت شده و سپس دستور زیر را اجرا می‌کنیم:

:play https://guides.neo4j.com/sandbox/twitch/index.html

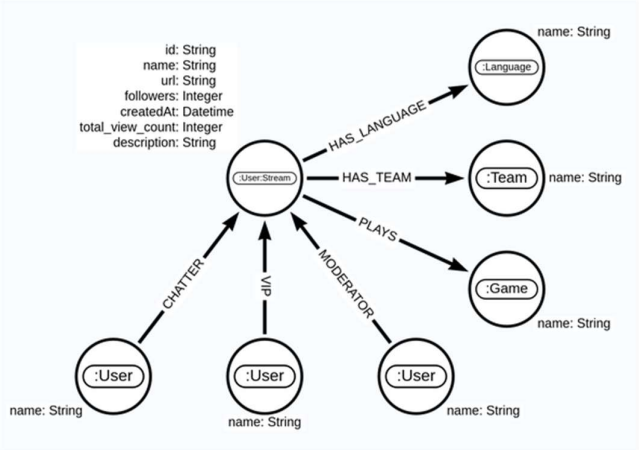
در بخش اول توضیحاتی مربوط به Twitch و راهنمای این دستور کار ذکر شده است:



Twitch: Twitch یک پلتفرم آنلاین است که به کاربران امکان می دهد محتوای خود را از طریق پخش زنده به اشتراک بگذارند. پخش‌کننده‌های توییچ، فعالیت خود را با اشتراک‌گذاری صفحه نمایش خود با طرفدارانی که می‌توانند آن‌ها را به صورت زنده بشنوند و تماشا کنند، پخش می‌کنند.

بخش ۲: Graph model

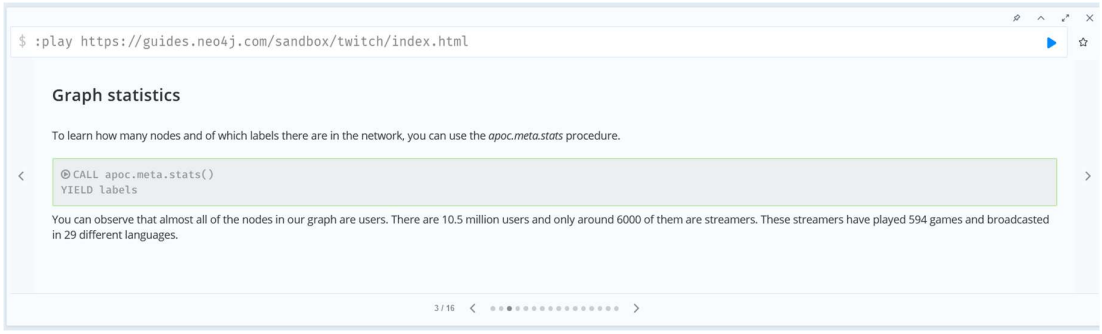
در این بخش توضیحاتی درباره مدل گراف Twitch داده شد که خلاصه آن به صورت زیر است:
پلتفرم اجتماعی Twitch از کاربران تشکیل شده است و تنها بخش کوچکی از این کاربران گیم پلی یا سایر فعالیت های خود را به صورت زنده پخش می کنند. در مدل گراف، آن دسته از کاربرانی که پخش stream می‌کنند با یک برچسب ثانویه «stream » شناسایی می‌شوند و پروفایل آنها حاوی اطلاعات اضافی مانند وابستگی‌های تیمی، بازی‌هایی که در stream خود انجام می‌دهند و زبانی که برای ارائه محتوای خود استفاده می‌کنند، است. این اطلاعات همچنین شامل تعداد فالوورهایی است که آنها دارند، تعداد بازدیدهای آنها و تاریخی که حساب کاربری خود را ایجاد کرده اند. برای تجزیه و تحلیل شبکه، مهم ترین اطلاعات شناسایی کاربرانی است که با چت streamer درگیر شده اند. روابط بین کاربران در چت را می توان به عنوان کاربران عادی (رابطه CHATTER)، ناظر stream (رابطه MODERATOR) یا VIP طبقه بندی کرد.
قابل توجه است که اطلاعات این پایگاه‌داده از تاریخ ۷ام تا ۱۰ می ۲۰۲۱ است.
توضیحات بالا در شکل زیر با رابطه‌ی گرافی نشان داده شده و entityهای آن بیشتر مشخص می‌شود:



بخش ۳: Graph statistics

apoc.meta.stats() در Neo4j توسط کتابخانه APOC ارائه شده است که عملکرد Neo4j را گسترش می دهد. روش apoc.meta.stats() این امکان را می‌دهد که اطلاعات مربوط به گره‌ها و روابط موجود در پایگاه داده گراف را بازیابی کنیم. مانند تعداد گره ها و روابط، تعداد گره های دارای یک برچسب خاص، و تعداد روابط با یک نوع خاص.

- **CALL** برای فراخوانی یک procedure استفاده می‌شود و apoc.meta.stats() آمار را برمی‌گرداند.
- **YIELD** برای استخراج داده‌های برگردانده شده و آمار برچسب‌ها استفاده می‌شود. آمار برچسب‌ها اطلاعاتی در مورد تعداد گره‌ها برای هر برچسب در پایگاه داده ارائه می‌دهد.



پس از اجرای این قطعه کد میبنیم که تقریباً همه گره‌های نمودار ما کاربر هستند. ۱۰/۵ میلیون کاربر وجود دارد و تنها حدود ۶۰۰۰ نفر از آنها streamer هستند. این streamerها ۵۹۴ بازی انجام داده‌اند و به ۲۹ زبان مختلف پخش شده‌اند:

بخش ۴: Exploring streamer's view and follower count

در این بخش می‌خواهیم اطلاعاتی درباره تعداد viewها و دنبال‌کننده‌های پخش‌کننده‌ها بدست آوریم.
تعداد کل بازدیدها و تعداد دنبال کنندگان برای هر streamer به عنوان properties گره ذخیره می شود. با استفاده از ORDER BY و LIMIT می‌توانیم پربیننده‌ترین و بیشترین دنبال‌کننده‌ها را پیدا کنیم.

- **پیدا کردن پخش‌کننده‌هایی با بیشترین تعداد بازدید در تمام زمان‌ها**

از دستور زیر استفاده می‌کنیم:

```
MATCH (u:Stream)
WHERE u.total_view_count IS NOT NULL
RETURN u.name as streamer,
       u.total_view_count as total_view_count
ORDER BY total_view_count DESC
LIMIT 10;
```

این پرس و جو در پایگاه داده برای همه گره هایی که دارای برچسب "Stream" هستند، جستجو می کند و فقط گره‌هایی را فیلتر می‌کند که ویژگی "total_view_count" آن‌ها null نیست. سپس نام streamer و تعداد کل بازدید آن‌ها را برای گره‌های منطبق برمی‌گرداند که به ترتیب نزولی بر اساس تعداد کل بازدید مرتب شده و به ۱۰ نتیجه برتر محدود می‌شود.
خروجی به صورت زیر است:

	streamer	total_view_count
1	"fextralife"	1451487256
2	"riotgames"	1272988816
3	"esl_csgo"	610467739
4	"shroud"	470283753
5	"beyondthesummit"	469790640
6	"summit1g"	449304545
7	"lirik"	387742294
8	"sodapoppin"	362553070
9	"xqcow"	341904167
10	"imaqtpie"	339812323

- **پیدا کردن پخش‌کننده‌هایی با بیشترین تعداد دنبال‌کننده**

از دستور زیر استفاده می‌کنیم:

```
MATCH (u:Stream)
WHERE u.followers IS NOT NULL
RETURN u.name as streamer,
       u.followers as followers
ORDER BY followers DESC
LIMIT 10;
```

این پرس و جو به دنبال تمام گره هایی با برچسب "Stream" در پایگاه داده است و فقط گره‌هایی را انتخاب می کند که دارای ویژگی "follower" غیر null هستند. سپس نام پخش‌کننده‌های منطبق و تعداد فالوورهای آنها را برمی‌گرداند که به ترتیب نزولی بر اساس تعداد دنبال‌کننده‌ها مرتب شده‌اند و به ۱۰ نتیجه برتر محدود می‌شوند. خروجی به صورت زیر است:

	streamer	followers
1	"tfue"	10243195
2	"shroud"	9073161
3	"rubius"	8975837
4	"auronplay"	8490422
6	"pokimane"	7698905
6	"thegrefg"	7269018
7	"myth"	7210332
8	"ibai"	5921582
9	"summit1g"	5898590
10	"nickmercs"	5675504

بخش ۵: Cypher implicit aggregations

در این قسمت توضیحی داده می‌شود که هنگام استفاده از عملگرهای aggregation در Cypher، باید توجه کرد که Cypher از aggregation ضمنی استفاده می کند. تمام متغیرهای non- aggregated که در RETURN یا WITH هستند به عنوان کلیدهای گروه بندی استفاده خواهند شد.

حال تعداد streamerهای جدید در هر سال را نشان می‌دهیم:

```
MATCH (u:Stream)
WHERE u.createdAt IS NOT NULL
RETURN u.createdAt.year as year,
       count(*) as countOfNewStreamers
ORDER BY year;
```

	year	countOfNewStreamers
1	2007	4
2	2008	12
3	2009	37
4	2010	67
5	2011	264
6	2012	442

این پرس و جو تمام گره هایی را که دارای برچسب "Stream" در پایگاه داده هستند جستجو می کند و فقط گره هایی را فیلتر می کند که ویژگی "createdAt" آن‌ها null نباشد. سپس بخش سال از ویژگی "createdAt" را برای هر گره منطبق به عنوان "year" به همراه تعداد کل گره های منطبق به عنوان "countOfNewStreamers" برمی گرداند. نتایج به ترتیب صعودی بر اساس مقدار "سال" مرتب شده اند.

بخش ۶: Counting the number of relationships per node

در Neo4j، هر رابطه بین گره ها دارای یک درجه است که نشان دهنده تعداد روابطی است که یک گره با گره های دیگر دارد. درجه یک گره می‌تواند یک معیار مهم برای تجزیه و تحلیل و درک روابط بین گره ها در یک نمودار باشد. برای بازیابی کارآمد درجه یک گره در Neo4j، پایگاه داده یک ذخیره شمارش نگه می دارد که درجه ارتباط را برای هر گره نگه می دارد. این ذخیره‌سازی تعداد به‌طور خودکار با ایجاد یا حذف روابط به‌روزرسانی می‌شود و امکان دسترسی سریع به درجه یک گره در زمان ثابت را فراهم می‌کند.
برای بازیابی درجه یک گره می‌توان از size در یک کوئری Cypher استفاده کرد تا تعداد روابط را برای یک گره یا مجموعه‌ای از گره ها به طور موثر بشماریم، بدون اینکه نیازی به پیمایش کل نمودار باشد.

حال بازی‌هایی را که دارای بیشترین تعداد streamerهایی هستند که آنها را بازی می کنند، را بررسی می‌کنیم:

```
MATCH (g:Game)
RETURN g.name as game,
       size( (g)<-[:PLAYS]-() ) as number_of_streamers
ORDER BY number_of_streamers DESC
LIMIT 10
```

	game	number_of_streamers
1	"Just Chatting"	1114
2	"Resident Evil Village"	546
3	"Grand Theft Auto V"	479
4	"League of Legends"	371
5	"Fortnite"	301
6	"VALORANT"	238
7	"Call of Duty: Warzone"	223
8	"Apex Legends"	216
9	"Counter-Strike: Global Offensive"	203
10	"Minecraft"	192

این پرس و جو تمام گره های با برچسب "Game" را در پایگاه داده را جستجو می کند و نام هر گره بازی و تعداد streamerهایی را که آن بازی را انجام می دهند، بازیابی می کند. تعداد streamerها با شمارش تعداد گره هایی که رابطه "PLAYS" ورودی از گره بازی دارند تعیین می شود. نتایج به ترتیب نزولی بر اساس تعداد پخش‌کننده‌هایی که هر بازی را انجام می‌دهند، مرتب شده‌اند و به ۱۰ بازی برتر محدود می‌شوند.

حال کاربران با بیشترین تعداد روابط VIP را پیدا می‌کنیم:

```
MATCH (u:User)
RETURN u.name as user,
       size( (u)-[:VIP]→() ) as number_of_vips
ORDER BY number_of_vips DESC LIMIT 10;
```

	user	number_of_vips
1	"nightbot"	18
2	"karuzo1g"	13
3	"kristoferyee"	12
4	"crazyslick"	9
5	"wolfabelle"	8
6	"streamelements"	7
7	"supibot"	7
8	"kephlivestream"	7
9	"letshugotv"	7
10	"saiiren"	6

این پرس و جو تمام گره های با برچسب "User" را در پایگاه داده جستجو می کند و نام هر گره کاربر و تعداد کاربران دیگری را که با آنها رابطه "VIP" دارند بازیابی می کند. تعداد روابط VIP برای یک کاربر با شمارش تعداد گره هایی که یک رابطه "VIP" خروجی با سایر کاربران دارند محاسبه می شود. نتایج به ترتیب نزولی بر اساس تعداد روابط VIP برای هر کاربر، و محدود به ۱۰ کاربر برتر است.

حال کاربران با بیشترین تعداد روابط MODERATOR را پیدا می‌کنیم:

```
MATCH (u:User)
RETURN u.name as user,
       size( (u)-[:MODERATOR]→() ) as number_of_mods
ORDER BY number_of_mods DESC LIMIT 10;
```

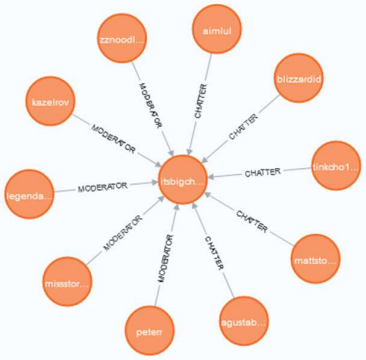
	user	number_of_mods
1	"nightbot"	3092
2	"streamelements"	2401
3	"moobot"	1369
4	"streamlabs"	688
5	"ssakdook"	248
6	"wizebot"	203
7	"fossabot"	182
8	"9kmmrbot"	53
9	"priestbot"	49
10	"soundalerts"	36

این پرس و جو تمام گره های با برچسب "User" را در پایگاه داده Twitch جستجو می کند و نام هر گره کاربر و تعداد کاربران دیگری را که با آنها رابطه "MODERATOR" دارند، بازیابی می کند. تعداد روابط تعدیل کننده برای یک کاربر با شمارش تعداد گره‌هایی که یک رابطه "MODERATOR" خروجی با سایر کاربران دارند محاسبه می شود. نتایج بر اساس تعداد روابط ناظر برای هر کاربر به ترتیب نزولی مرتب شده و محدود به ۱۰ کاربر برتر است.

بخش ۷: Cypher subqueries

CALL امکان اجرای پرس و جوهای فرعی، یعنی پرس و جوها را در داخل پرس و جوهای دیگر می دهد. پرس و جوهای فرعی به ما این امکان را می‌دهند که پرس و جو بنویسیم که مخصوصاً هنگام کار با aggregation یا محدود کردن نتایج مفید است.
در مثال گفته شده، ابتدا یک streamer واحد را مطابقت می‌دهیم و بعد، از یک پرسش فرعی برای مطابقت با پنج streamer و به طور جداگانه پنج کاربر که در پخش streamer چت کرده اند، استفاده می‌کنیم.

```
MATCH (s:Stream)
WITH s LIMIT 1
CALL {
    WITH s
    MATCH p=(s)<--(:Stream)
    RETURN p
    LIMIT 5
    UNION
    WITH s
    MATCH p=(s)<--(:User)
    RETURN p
    LIMIT 5
}
```



همانطورکه مشاهده می‌شود، streamerها مانند کاربران عادی رفتار می کنند. آنها می‌توانند در برنامه‌های پخش کننده دیگر چت کنند، ناظر یا VIP باشند.
این پرس و جو یک گره منفرد با عنوان "Stream" را در پایگاه داده Twitch جستجو می کند و آن را به متغیر "s" اختصاص می دهد. سپس پرس و جوی فرعی را اجرا می کند، که پنج جریان یا کاربران اخیر را که با یک رابطه ورودی به گره جریان اصلی "s" متصل شده‌اند را پیدا می کند. پرس و جوی فرعی ابتدا روابط ورودی از سایر گره های «جریان» را جستجو می کند و سپس روابط ورودی از گره‌های «User » را جستجو می کند. کلمه کلیدی "UNION" برای ترکیب نتایج دو عبارت MATCH در جستار فرعی استفاده می شود.
در نهایت، پرس و جو اصلی متغیر "p" را برمی گرداند، که نشان دهنده پنج رابطه ورودی جدید به گره جریان اصلی "s" است. پرس و جو فقط یک نتیجه را برمی گرداند که جدیدترین مسیر در میان مواردی است که در جستار فرعی یافت می شود.

بخش ۸: Node degree distribution

درجه گره تعداد اتصالاتی است که هر گره در یک شبکه دارد. در این حالت شبکه جهت دهی می شود یعنی جهت رابطه معنادار است. در شبکه های هدایت شده، امکان تفکیک توزیع درجه گره به درون درجه (شمارش اتصالات ورودی) و خارج درجه (شمارش اتصالات خروجی) وجود دارد. تمرکز بر توزیع خارج از درجه شبکه چت کاربر است و برای تجزیه و تحلیل آن می توان از apoc.agg.statistics برای مشاهده مقادیر استفاده کرد.

```
MATCH (u:User)
WITH u, size( (u)-[:CHATTER|VIP|MODERATOR]→() ) as node_outdegree
RETURN apoc.agg.statistics(node_outdegree) as statistics
```

statistics


```
{
  "total": 10514229,
  "min": 0,
  "minNonZero": 1.0,
  "max": 5348,
  "mean": 1.8758665043342693,
  "0.5": 1,
  "0.99": 8,
  "0.75": 2,
  "0.9": 4,
  "0.95": 5,
  "stdev": 7.898665162178397
}
```


این پرس و جو در پایگاه داده Twitch برای همه گره های دارای برچسب "User" جستجو می کند. سپس درجه برتر هر گره کاربر را محاسبه می‌کند، که نشان‌دهنده تعداد روابطی است که از آن گره منشأ می‌گیرد، با انواع رابطه مشخص شده «CHATTER»، «VIP» و «MODERATOR». سپس نتیجه به تابع apoc.agg.statistics() منتقل می شود که اطلاعات آماری مربوط به درجه های برتر محاسبه شده را برمی گرداند. این اطلاعات شامل مقادیر حداقل و حداکثر، میانگین، میانه، انحراف معیار و مجموع همه مقادیر است. به طور خلاصه، این پرس و جو اطلاعات آماری مربوط به درجه برتر هر گره کاربر Twitch را بر اساس روابط "CHATTER"، "VIP" و "MODERATOR" محاسبه و برمی گرداند.

حال توزیع درون درجه ای شبکه چت کاربر را بررسی خواهیم کرد. توجه شود که فقط پخش‌کننده‌ها روابط ورودی دارند، بنابراین می‌توانیم از بازرسی کاربران عادی صرف نظر کنیم، زیرا از قبل می‌دانیم که درجه آنها صفر است.

```
MATCH (u:Stream)
WITH u, size( (u)←[:CHATTER|VIP|MODERATOR]-() ) as node_indegree
RETURN apoc.agg.statistics(node_indegree) as statistics
```

statistics


```
{
  "total": 5787,
  "min": 1,
  "minNonZero": 1.0,
  "max": 316735,
  "mean": 3408.391048902713,
  "0.5": 1052,
  "0.99": 39295,
  "0.75": 2595,
  "0.9": 6791,
  "0.95": 12071,
  "stdev": 11067.095171834044
}
```


این پرس و جو برای همه گره های دارای برچسب "Stream" جستجو می کند. برای هر یک از این گره‌ها، تعداد روابط ورودی آنها از نوع ""CHATTER"، "VIP"یا "MODERATOR" را محاسبه می کند. سپس آمار مقادیر node_indegree حاصل را با استفاده از تابع apoc.agg.statistics()برمی گرداند.

بخش ۹: Graph Data Science library

کتابخانه Graph Data Science (GDS) در Neo4j مجموعه‌ای از ابزارها است که به کاربران اجازه می‌دهد الگوریتم‌های مختلف ریاضی را روی نمودارها اعمال کنند. این کتابخانه بیش از ۵۰ الگوریتم را برای تجزیه و تحلیل گراف فراهم می کند، مانند تعیین اهمیت گره های خاص در یک گراف، تشخیص جوامع گره ها با ویژگی های مشابه، و جاسازی گره ها در فضاهای با ابعاد پایین تر. کتابخانه GDS طوری طراحی شده است که بر روی یک فرمت گراف تخصصی در حافظه کار کند که به بهبود عملکرد و مقیاس پذیری کمک می کند. این کار انجام وظایف تحلیل گراف پیچیده را روی مجموعه داده های بزرگ ذخیره شده در پایگاه داده Neo4j برای کاربران آسان تر می کند.

حال تمام گره های User و Stream و روابط احتمالی بین آنها، که CHATTER، MODERATOR و VIP هستند، را نمایش می‌دهیم. خروجی به صورت زیر است:

```
CALL gds.graph.project('TwitchGraph',
  ['User', 'Stream'],
  ['CHATTER', 'VIP', 'MODERATOR'])
```

nodeProjection	relationshipProjection	graphName	nodeCount	relationshipCount	projectMillis
{ "User": { "label": "User", "properties": { } }, "Stream": { "label": "Stream", "properties": { } } }	{ "CHATTER": { "orientation": "NATURAL", "indexInverse": false, "aggregation": "DEFAULT", "type": "CHATTER", "properties": { } }, "VIP": { "orientation": "NATURAL", "indexInverse": false, "aggregation": "DEFAULT", "type": "VIP", "properties": { } }, "MODERATOR": { "orientation": "NATURAL", "indexInverse": false, "aggregation": "DEFAULT", "type": "MODERATOR", "properties": { } } }	"TwitchGraph"	10514229	19723263	4234

این کوئری تابع gds.graph.project() را فراخوانی می‌کند تا یک نمودار جدید ارائه کند که فقط شامل گره‌هایی با برچسب «User» یا «Stream » و روابط با برچسب «CHATTER»، «VIP» یا «MODERATOR» می‌شود. نمودار جدید پیش بینی شده فقط شامل این گره ها و روابط خواهد بود و هر گره یا رابطه دیگری که به صراحت در طرح ریزی گنجانده نشده باشد از نمودار حذف خواهد شد.

بخش ۱۰: Weakly Connected Components

الگوریتم Weakly Connected Components (WCC) ابزاری در کتابخانه Neo4j Graph Data Science است که برای شناسایی جزایر متمایز یا خوشه‌های گره در یک گراف استفاده می‌شود. به طور خاص، این الگوریتم برای شناسایی اجزایی که در آن حداقل یک مسیر بین هر دو گره وجود دارد، بدون توجه به جهت روابط استفاده می شود. اساساً، الگوریتم WCC می‌تواند گروه‌هایی از گره‌ها را شناسایی کند که به نوعی به یکدیگر متصل هستند، اما مستقیماً به گروه‌های دیگر گره‌ها در همان گراف متصل نیستند. این می تواند در برنامه های مختلف مانند تجزیه و تحلیل شبکه های اجتماعی، شناسایی شبکه های فرعی در یک شبکه بزرگتر، یا شناسایی خوشه هایی از گره ها که دارای ویژگی های مشابه هستند، مفید باشد.

حال از کوئری Cypher زیر برای اجرای الگوریتم Weakly-Connected Components در شبکه کاربر Twitch استفاده می‌کنیم(روش stats زمانی استفاده می شود که ما فقط به آمار سطح بالا از نتایج الگوریتم نیاز داریم).

```
CALL gds.wcc.stats('twitch')
YIELD componentCount, componentDistribution
```

componentCount	componentDistribution
1	{"p99": 10514239, "min": 10514176, "max": 10514239, "mean": 10514208.0, "p90": 10514239, "p50": 10514239, "p999": 10514239, "p95": 10514239, "p75": 10514239}

با in-memory graph projection، می‌توانیم گره‌ها یا روابط را در زمان اجرای الگوریتم فیلتر کنیم. در اینجا، الگوریتم WCC فقط گره های Stream و اتصالات بین آنها را در نظر می گیریم:

```
CALL gds.wcc.stats('twitch', {nodeLabels:['Stream']})
YIELD componentCount, componentDistribution
```

componentCount	componentDistribution
1902	{"p99": 3, "min": 1, "max": 3692, "mean": 3.0425867507886437, "p90": 1, "p50": 1, "p999": 17, "p95": 2, "p75": 1}

بخش ۱۱: PageRank

PageRank یک الگوریتم گراف پرکاربرد است که اهمیت نسبی گره ها را در یک گراف اندازه گیری می کند. این الگوریتم به هر گره بر اساس ارتباط آن با گره های دیگر در نمودار امتیازی اختصاص می دهد. ایده اصلی PageRank این است که گره‌هایی با تعداد زیادی لینک ورودی از سایر گره‌های مهم، خود مهم‌تر از گره‌هایی هستند که لینک‌های ورودی کم یا بدون آن هستند. این الگوریتم در ابتدا توسط گوگل برای رتبه‌بندی اهمیت webpage توسعه داده شد، اما از آن زمان در برنامه‌های کاربردی مختلفی مانند تجزیه و تحلیل شبکه‌های اجتماعی، سیستم‌های توصیه و بازیابی اطلاعات استفاده شده است.

۱۰حال کاربر مهم Twitch را بر اساس تعداد و اهمیت سایر کاربرانی که با آنها تعامل دارند، پیدا می کنیم:

```
CALL gds.pageRank.stream('twitch')
YIELD nodeId, score
WITH nodeId, score
ORDER BY score
DESC LIMIT 10
RETURN gds.util.asNode(nodeId).name as user, score
```

	user	score
1	"yassuo"	100479.0200620129
2	"trainwreckstv"	81438.90516599993
3	"lolttyler1"	51774.70070114764
4	"riotgames"	42755.56330485601
5	"xqcow"	40166.719242040526
6	"csgomc_ru"	30045.479648472534
7	"benjyfishy"	29315.003128816497
8	"alexby11"	26306.094834403193
9	"nickmercs"	25194.171606722313
10	"esl_csgo"	24080.77147632009

این کوئری از کتابخانه GDS (Graph Data Science) برای محاسبه الگوریتم PageRank بر روی گره های گراف "twitch" استفاده می کند. الگوریتم PageRank روشی برای اندازه‌گیری اهمیت هر گره در نمودار بر اساس تعداد لینک‌های ورودی و اهمیت گره‌هایی است که به آن پیوند می‌دهند. این پرس و جو امتیاز PageRank را برای هر گره در نمودار محاسبه می کند و ۱۰ گره برتر با بالاترین امتیاز را به همراه نام آنها (بازیابی شده با استفاده از ویژگی "name") و امتیاز PageRank آنها برمی گرداند.

همچنین به طور مشابه با الگوریتم WCC، می‌توانیم انتخاب کنیم که PageRank در زیرگراف streamer اجرا شود:

```
CALL gds.pageRank.stream('twitch', {nodeLabels:['Stream']})
YIELD nodeId, score
WITH nodeId, score
ORDER BY score
DESC LIMIT 10
WITH gds.util.asNode(nodeId) as node,score
RETURN node.name as streamer,
    score,
    size( (node)<--(:Stream) ) as relationships_from_streamers,
    size( (node)<--(:User) ) as relationships_from_users
```

	streamer	score	relationships_from_streamers	relationships_from_users
1	"yassuo"	38.377973188919285	16	22632
2	"trainwreckstv"	32.3300706103876	101	83934
3	"lolttyler1"	18.88854473858766	9	44564
4	"xqcow"	12.268081806320227	105	280443
5	"riotgames"	12.202151751055517	128	316735
6	"benjyfishy"	8.824295872510227	57	65097
7	"asmongold"	7.818024131908357	61	102450
8	"roshtein"	7.233139396623394	38	56238
9	"nickmercs"	6.011014091869845	18	85486
10	"csgomc_ru"	5.805311072662871	39	221012

پرس و جو ابتدا gds.pageRank.stream() را فراخوانی می کند و نام نمودار را ارسال می کند، که در این مورد "twitch" است، و یک نقشه پیکربندی که شامل برچسب های گره برای در نظر گرفتن است، که در اینجا فقط "Stream" است. سپس نتیجه محاسبه

PageRank به صورت جریانی از گره‌ها و امتیازهای مربوط به آنها برگردانده می شود. سپس پرس و جو نتایج را به‌گونه‌ای فیلتر می‌کند که فقط شامل ۱۰ پخش کننده برتر با بالاترین امتیاز باشد و نام، امتیازات و تعداد روابط دریافتی از پخش کننده ها و کاربران را برمی‌گرداند. به عبارت ساده تر، این پرس و جو به یافتن ۱۰ streamer برتر با بالاترین امتیاز PageRank و نشان دادن نام، امتیازات و تعداد روابط ورودی از دیگر streamerها و کاربران می‌پردازد.

بخش ۱۲: Community detection

Community detection مجموعه‌ای از الگوریتم‌های گراف است که برای شناسایی گروه‌ها یا خوشه‌هایی از گره‌ها در یک شبکه استفاده می‌شود که برخی از ویژگی‌ها یا ویژگی‌های مشترک را به اشتراک می‌گذارند. جوامع به عنوان زیرمجموعه‌هایی از گره‌ها تعریف می‌شوند که نسبت به گره‌های خارج از جامعه، به طور متراکم‌تری به یکدیگر متصل هستند و می‌توانند واحدهای عملکردی یا ساختاری را در شبکه نشان دهند. این الگوریتم‌ها به‌طور گسترده در زمینه‌های مختلف مورد استفاده قرار می‌گیرند و بینشی در مورد ساختار و پویایی شبکه‌های پیچیده ارائه می‌دهند و تجسم و تحلیل شبکه را تسهیل می‌کنند.

حال ۱۰ انجمن بزرگ را پیدا می‌کنیم:

```
CALL gds.louvain.stream('twitch', {nodeLabels:['Stream']})
YIELD nodeId, communityId
RETURN communityId, count(*) as communitySize
ORDER BY communitySize DESC LIMIT 10
```

	communityId	communitySize
1	1154	510
2	3464	445
3	389	423
4	3846	323
6	4702	259
8	2694	245
7	2590	235
5	2943	158
9	4048	151
10	2059	137

این پرس و جو از الگوریتم Community detection Louvain از کتابخانه Graph Data Science برای شناسایی ۱۰ انجمن برتر در پایگاه داده Twitch گره هایی با برچسب "Stream" استفاده می کند. سپس شناسه جامعه و تعداد گره‌ها در هر جامعه را برمی‌گرداند که به ترتیب نزولی بر اساس اندازه جامعه مرتب شده‌اند. الگوریتم Louvain استریمرهایی را که به طور نزدیک به هم مرتبط هستند را گروه بندی می کند و پرس و جو شناسه ها و اندازه های بزرگترین جوامع را برمی گرداند.

مشکلات و توضیحات تکمیلی

تنها مشکل، وجود ارور در کدهای داده شده به دلیل استفاده از {} count بود که با تغییر آن به ()size برطرف شد.

آنچه آموختم / پیشنهادات

در این دستور کار با مدیریت دیتابیس های گرافی توسط Neo4j و زبان Cypher آشنا شدم و توانستم روابط بین داده‌ها را نه تنها توسط گراف‌ها نمایش دهم.