Torrent Files Resolver IV

Maciej Michalski 311351

1 Instalacja i konfiguracja neo4j

W ramach montażu skorzystałem z pliku konfiguracyjnego `docker compose`:

```
name: neo4j
services:
  neo:
    image: neo4j:5.26.0-community-bullseye
    restart: always
    ports:
      - 7474:7474
      - 7687:7687
    volumes:
      - neo4j:/data

volumes:
  neo4j:
    name: "n4j"
```

`Neo4J` posiada własną aplikację webową zezwalającą na wykorzystywanie komend, wizualizację grafów etc.

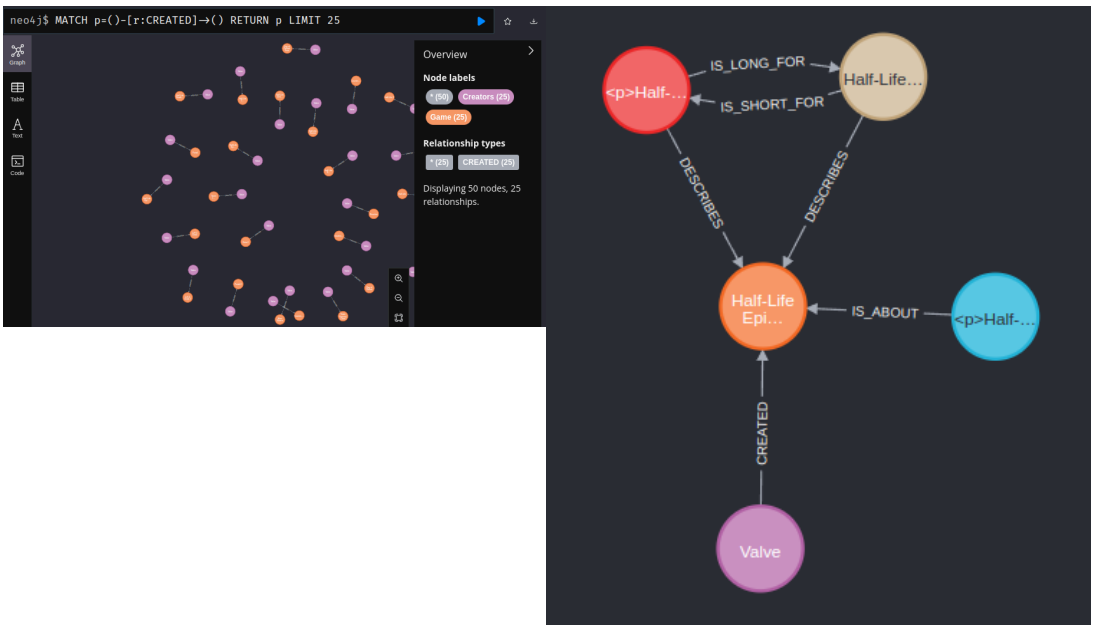
Wynik instalacji zweryfikowałem poniższą komendą:

```
SHOW DATABASES

Query executed in 30ms. Query type: SCHEMA_WRITE.
Got 2 rows. View results: as Table
....
"neo4j"
"system"
```

2 Dołączenie zbioru pierwszego

Po dokonaniu sanityzacji danych wejściowych (neo4j nie przyjmuje cudzysłowia wewnątrz cudzysłowia nawet przy wykorzystaniu escape sequence) dodałem dane, z tego samego źródła co przy etapie poprzednim:

```
LOAD CSV WITH HEADERS FROM 'file:///steam.csv' AS row
CREATE(d:Creators {developer: row.developer, publisher: row.publisher})
        -[r:CREATED]->(g:Game {steam_id: row.appid, name: row.name, date: row.release_date});

LOAD CSV WITH HEADERS FROM 'file:///desc_unsecaped.csv' AS row
MATCH (existing:Game) WHERE existing.steamid = row.steam_appid
CREATE (long:LongDescription {content: row.detailed_description})
CREATE (short:ShortDescription {content: row.short_description})
CREATE (about:AboutGame {content: row.about_the_game})
CREATE (long)-[:DESCRIBES]->(existing)
CREATE (short)-[:DESCRIBES]->(existing)
CREATE (about)-[:IS_ABOUT]->(existing)
CREATE (short)-[:IS_SHORT_FOR]->(long)
CREATE (long)-[:IS_LONG_FOR]->(short)
CREATE (about)-[:IS_SUPPLEMENT_FOR]->(short);
```



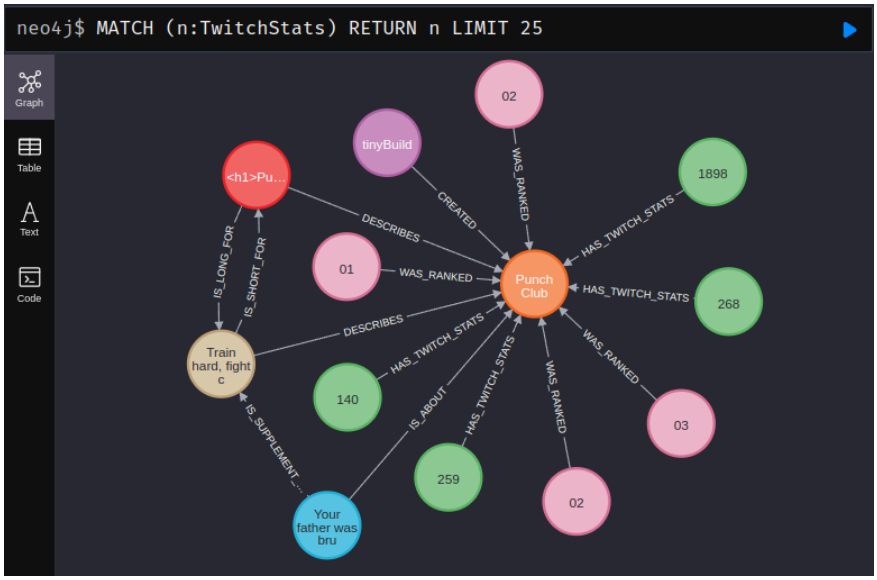
Rys. 1: Po lewej mamy liczniejszą grupę przykładów dla relacji CREATED. Po prawej wszystkie realcje dotyczące gry dla `Half-Life`.

3 Dołączenie zbioru drugiego

Jako drugi zbiór dołączyłem ranking gier z kilku lat, pokazywanych na Twitchu. Każdy ranking wiąże się z konkretnym miesiącem z konkretnego roku. Kilka rankingów będzie istniało dla tej samej gry przy takim rozrachunku. Zbiór jest więc rozszerzony o następujące wartości:

```
LOAD CSV WITH HEADERS FROM 'file:///twitch.csv' AS row
MATCH (existing:Game)
  WHERE existing.name = row.Game
CREATE (t:Averages {viewers: row.Avg_viewers, channels: row.Avg_channels, viewer_ratio:
row.Avg_viewer_ratio})
CREATE (h:Viewings {watched: row.Hours_watched, streamed: row.Hours_streamed})
CREATE (p:Peaks {viewers: row.Peak_viewers, channels: row.Peak_channels})
CREATE (s:Streamers {streamers: row.Streamers})
```

```
CREATE (n:TwitchRank {rank: row.Rank, month: row.Month, year: row.Year})
CREATE (t)-[:HAS_AVERAGES_OF]->(existing)
CREATE (h)-[:HAS_VIEWINGS_OF]->(existing)
CREATE (p)-[:HAS_PEAKS_OF]->(existing)
CREATE (s)-[:IS_STREAMED_BY]->(existing)
CREATE (n)-[:WAS_RANKED]->(existing);
```



Rys. 2: Relacje po rozszerzeniu.

3.1 Informacja o rozszerzeniu i wykorzystaniu

3.2 5 zapytań (1 Union 1 Merdge)

Zapytanie wiążące prawdopodobne wystąpienie w rankingach z grami:

```
MATCH (g:Game)<-[:DESCRIBES]-(c:ShortDescription)
MATCH (g:Game)<-[:CREATED]-(d:Creators)
OPTIONAL MATCH (g:Game)<-[:HAS_TWITCH_STATS]-(t:TwitchStats)
RETURN g.name AS name, date(g.date) AS date, c.content AS short_description,
       collect(t) AS twitch_stats, d.developer AS developer;
```

Rezultat:

name	date	short_description	twitch_stats	developer
"Counter-Strike"	2000-11-01	"Play the worlds number 1 online action gam...	twitch_stats: list 0: node<145809> 1: node<146201> 2: node<146275> 3: node<146021> 4: node<146143> 5: node<146603> 6: node<146337> 7: node<145949> 8: node<145879>	"Valve"

Ograniczenia na dane.

```
CREATE CONSTRAINT creators_unique IF NOT EXISTS
FOR (c:Creators)
REQUIRE (c.developer, c.publisher) IS UNIQUE;
```

```
CREATE CONSTRAINT game_steam_id_unique IF NOT EXISTS
FOR (g:Game)
REQUIRE g.steam_id IS UNIQUE;

CREATE CONSTRAINT twitch_rank_unique IF NOT EXISTS
FOR (t:TwitchRank)
REQUIRE (t.rank, t.month, t.year) IS UNIQUE;
```

id	name	type	entityType	labelsOrTypes	properties
8	"creators_unique"	"UNIQUENESS"	"NODE"	["Creators"]	["developer", "publisher"]
4	"game_steam_id_unique"	"UNIQUENESS"	"NODE"	["Game"]	["steam_id"]
6	"twitch_rank_unique"	"UNIQUENESS"	"NODE"	["TwitchRank"]	["rank", "month", "year"]

Każde z nich to swoisty sanity check. Nie może istnieć dwóch takich samych duo (deweloper, wydawca), `steam_appid` musi być unikalne bo w innym wypadku nie można powiązać danych opisowych z konkretnymi tytułami oraz trójka (rank, month, year) musi być unikalna (nie może być dwóch gier na pierwszym miejscu w minimalnym kroku czasowym pomiarów). W każdym wypadku dopuszczam brak danych.

Ale niestety importując wcześniej `Creator` i `Games` nie sprawdziłem, że istnieją w danych duplikaty. Uniemożliwiło mi to utworzenie pierwszego ograniczenia. Musiałem więc zmienić import gier i twórców na:

```
LOAD CSV WITH HEADERS FROM 'file:///steam.csv' AS row
MERGE (d:Creators {developer: row.developer, publisher: row.publisher})
MERGE (g:Game {steam_id: row.appid, name: row.name, date: row.release_date})
ON CREATE SET g.name = row.name, g.date = row.release_date
MERGE (d)-[:CREATED]->(g);
```

Tak samo musiałem postąpić przy nie istniejącej grze:

```
LOAD CSV WITH HEADERS FROM 'file:///twitch.csv' AS row
MERGE (existing:Game {name: row.Game})
ON CREATE SET existing.date = row.Release_date
CREATE (t:Averages {viewers: row.Avg_viewers, channels: row.Avg_channels, viewer_ratio: viewer:
row.Avg_viewer_ratio})
CREATE (h:Viewings {watched: row.Hours_watched, streamed: row.Hours_streamed})
CREATE (p:Peaks {viewers: row.Peak_viewers, channels: row.Peak_channels})
CREATE (s:Streamers {streamers: row.Streamers})
CREATE (n:TwitchRank {rank: row.Rank, month: row.Month, year: row.Year})
CREATE (t)-[:HAS_AVERAGES_OF]->(existing)
CREATE (h)-[:HAS_VIEWINGS_OF]->(existing)
CREATE (p)-[:HAS_PEAKS_OF]->(existing)
CREATE (s)-[:IS_STREAMED_BY]->(existing)
CREATE (n)-[:WAS_RANKED]->(existing);
```

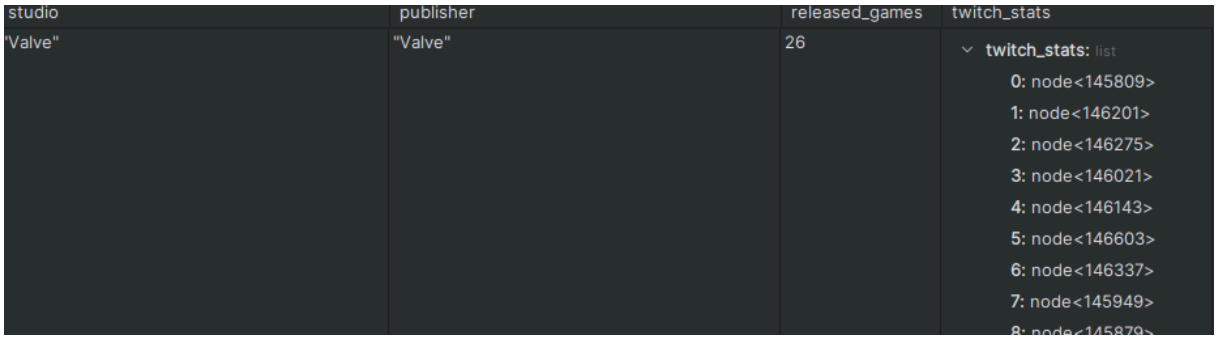
Poniżej zawarłem inne zapytani, które mogłyby być przydatne przy potencjalnej aplikacji biznesowej.

Zapytanie sprawdzające który z duo deweloper + wydawca generuje najwięcej rankingów i ich szczegóły:

```
MATCH (g:Game)<-[:CREATED]-(d:Creators)
OPTIONAL MATCH (g:Game)<-[:HAS_TWITCH_STATS]-(t:TwitchStats)
```

```
RETURN d.developer AS studio, d.publisher AS publisher,  COUNT(DISTINCT g) AS released_games,
       collect(t) AS twitch_stats
```

Rezultat:



Zapytanie sprawdzające istniejące rankingi, zliczające je, pokazujące datę wydania gry razem z ostatnią datą rankingu oraz to czy posiada słowo `shooter` w swoim długim opisie.

```
MATCH (g:Game)<-[:DESCRIBES]-(d:LongDescription)
OPTIONAL MATCH (g:Game)<-[h:WAS_RANKED]-(t:TwitchRank)
  WHERE t.rank > '30'
WITH
  g.name AS name,
  d.content AS long_description,
  count(DISTINCT t) AS observed_rankings,
  collect(t) AS ranks,
  g.date AS release_year,
  max(t.year) AS most_recent_ranking_year
  WHERE
  observed_rankings > 1
RETURN
  name,
  long_description,
  observed_rankings,
  ranks,
  release_year,
  most_recent_ranking_year,
  CASE
    WHEN long_description CONTAINS 'shooter' THEN true
    ELSE false
    END AS contains_shooter_in_description;
```

Rezultat:

Zapytanie robiące to co powyżej, ale dla ilości zaobserwowanych rankingów większej niż 5, zostają one rozbrojone na osobne wiersze. Wydaje mi się, że mogłoby to być dosyć przydatna forma dla cacheowanego rezultatu bowiem zachowuje on mniej więcej informację o wielkości zwrotu do potencjalnej aplikacji, jednocześnie zachowując formę agregacji posiadającą informację biznesową.

```
MATCH (g:Game)<-[:DESCRIBES]-(d:LongDescription)
OPTIONAL MATCH (g:Game)<-[h:WAS_RANKED]-(t:TwitchRank)
```

```
  WHERE t.rank > '30'
WITH
  g.name AS name,
  d.content AS long_description,
  count(DISTINCT t) AS observed_rankings,
  g.date AS release_year,
  collect(t) AS ranks,
  max(t.year) AS most_recent_ranking_year
  WHERE
  observed_rankings < 5 AND observed_rankings > 1
RETURN
  name,
  long_description,
  observed_rankings,
  ranks,
  release_year,
  most_recent_ranking_year,
  CASE
    WHEN long_description CONTAINS 'team' THEN true
    ELSE false
    END AS contains_team_in_description
UNION
MATCH (g:Game)<-[:DESCRIBES]-(d:LongDescription)
OPTIONAL MATCH (g:Game)<-[h:WAS_RANKED]-(t:TwitchRank)
  WHERE t.rank > '30'
WITH
  g.name AS name,
  d.content AS long_description,
  [t] AS ranks,
  g.date AS release_year,
  [t.year] AS most_recent_ranking_year,

  count(DISTINCT t) AS observed_rankings
  WHERE observed_rankings >= 5 OR observed_rankings = 1

RETURN
  name,
  long_description,
  null as observed_rankings,
  ranks,
  release_year,
  most_recent_ranking_year,
  CASE
    WHEN long_description CONTAINS 'team' THEN true
    ELSE false
    END AS contains_team_in_description;
```

4 Indeksy

4.1 Zasada działania

Indeksy w `neo4j` odgrywają podobną docelową rolę co indeksy w innych do tej pory referowanych bazach danych [1]. Mają one na celu szybsze wykonywanie zapytań. Dokumentacja `neo4j` oznajmia:

„*An index is a copy of specified primary data in a Neo4j database, such as nodes, relationships, or properties. The data stored in the index provides an access path to the data in the primary storage and allows users to evaluate query filters more efficiently (and, in some cases, semantically interpret query filters). In short, much like indexes in a book, their function in a Neo4j graph database is to make data retrieval more efficient*“. [2]

Wspierane są dwie kategorie indeksów [3]:

- `Search-performance indexes` - mające na celu przyspieszenie wydobycia danych na podstawie dokładnych dopasowań.
- `Semantic indexes` - sprawdza podobieństwo między podanym ciągiem znaków, a danymi. W tej kategorii zawierają się wyszukiwania tekstowe oraz indeksy wektorowe.

Same `search-performence indexes` dzielą się na kilka kategorii [3]:

- `range indexes` - podstawowy indeks,
- `text indexes` - zoptymalizowany pod zapytania mające na celu korzystanie z `CONTAINS` i `ENDS WITH`,
- `point indexes` - przyspiesza wyszukiwanie przestrzenne wartości typu `POINT` np. względem dystansu czy przynależności,
- `token lookup indexes` - zajmujący się wyłacznie labelami węzłów.

Dla każdego z wcześniej przedstawionych zapytań dokonałem pomiarów czasu i wglądu w plan zapytania. Dla każdego zapytania została wyliczona mediana z 10 prób wykoniania zapytania.

4.1.1 Zapytanie 1

Sprawdziłem poniższe zapytanie:

```
PROFILE MATCH (g:Game)<-[:DESCRIBES]-(c:ShortDescription)
MATCH (g:Game)<-[:CREATED]-(d:Creators)
OPTIONAL MATCH (g:Game)<-[:HAS_TWITCH_STATS]-(t:TwitchStats)
RETURN g.name AS name, date(g.date) AS date, c.content AS short_description,
       collect(t) AS twitch_stats, d.developer AS developer;
```

Z podanymi indeksami:

```
CREATE INDEX short_description_content_index IF NOT EXISTS FOR (c:ShortDescription) ON
(c.content);
CREATE INDEX creators_developer_index IF NOT EXISTS FOR (d:Creators) ON (d.developer);
CREATE INDEX t_rank_index IF NOT EXISTS FOR (t:TwitchRank) ON (t.rank, t.year);
CREATE INDEX t_month_index IF NOT EXISTS FOR (t:TwitchRank) ON (t.month);
```

Tab. 1: Wyniki z indeksami i bez indeksów.

\	Indexed time [ms]	Non indexed time [ms]
Med	20	45

4.1.2 Zapytanie 2

```
PROFILE MATCH (g:Game)<-[:CREATED]-(d:Creators)
OPTIONAL MATCH (g:Game)<-[:HAS_TWITCH_STATS]-(t:TwitchStats)
RETURN d.developer AS studio, d.publisher AS publisher, count(DISTINCT g) AS released_games,
       collect(t) AS twitch_stats
```

Z poniższymi indeksami:

```
CREATE INDEX short_description_content_index IF NOT EXISTS FOR (c:ShortDescription) ON
(c.content);
CREATE INDEX creators_developer_index IF NOT EXISTS FOR (d:Creators) ON (d.developer);
CREATE INDEX t_rank_index IF NOT EXISTS FOR (t:TwitchRank) ON (t.rank, t.year);
CREATE INDEX t_month_index IF NOT EXISTS FOR (t:TwitchRank) ON (t.month);
```

Tab. 2: Wyniki z indeksami i bez indeksów.

\	Indexed time [ms]	Non indexed time [ms]
Med	78	97

Jak widać przyrost nie jest tak duży jak w przypadku pierwszego zapytania.

4.1.3 Zapytanie 3

```
PROFILE MATCH (g:Game)<-[:DESCRIBES]-(d:LongDescription)
OPTIONAL MATCH (g:Game)<-[h:WAS_RANKED]-(t:TwitchRank)
  WHERE t.rank > '30'
WITH
  g.name AS name,
  d.content AS long_description,
  count(DISTINCT t) AS observed_rankings,
  collect(t) AS ranks,
  g.date AS release_year,
  max(t.year) AS most_recent_ranking_year
  WHERE
  observed_rankings > 1
RETURN
  name,
  long_description,
  observed_rankings,
  ranks,
  release_year,
  most_recent_ranking_year,
  CASE
    WHEN long_description CONTAINS 'shooter' THEN true
    ELSE false
    END AS contains_shooter_in_description;
```

Z poniższymi indeksami:

```
CREATE INDEX creators_developer_index IF NOT EXISTS FOR (d:Creators) ON (d.developer);
CREATE TEXT INDEX  long_description_fulltext_index FOR (d:LongDescription) ON (d.content);
CREATE TEXT INDEX  short_description_content_index FOR (d:ShortDescription) ON (d.content);
CREATE TEXT INDEX  about_description_content_index FOR (d:AboutGame) ON (d.content);
CREATE INDEX game_index IF NOT EXISTS FOR (g:Game) ON (g.name);
CREATE INDEX t_rank_index IF NOT EXISTS FOR (t:TwitchRank) ON (t.rank, t.year);
CREATE INDEX t_month_index IF NOT EXISTS FOR (t:TwitchRank) ON (t.month);
```

Tab. 3: Wyniki z indeksami i bez indeksów.

\	Indexed time [ms]	Non indexed time [ms]
Med	165	180

4.1.4 Zapytanie 4

```
PROFILE MATCH (g:Game)<-[:DESCRIBES]-(d:LongDescription)
OPTIONAL MATCH (g:Game)<-[h:WAS_RANKED]-(t:TwitchRank)
  WHERE t.rank > '30'
WITH
  g.name AS name,
  d.content AS long_description,
  count(DISTINCT t) AS observed_rankings,
  g.date AS release_year,
  collect(t) AS ranks,
  max(t.year) AS most_recent_ranking_year
  WHERE
  observed_rankings < 5 AND observed_rankings > 1
RETURN
  name,
  long_description,
  observed_rankings,
  ranks,
  release_year,
```

```
  most_recent_ranking_year,
  CASE
    WHEN long_description CONTAINS 'team' THEN true
    ELSE false
    END AS contains_team_in_description
UNION
MATCH (g:Game)<-[:DESCRIBES]-(d:LongDescription)
OPTIONAL MATCH (g:Game)<-[h:WAS_RANKED]-(t:TwitchRank)
  WHERE t.rank > '30'
WITH
  g.name AS name,
  d.content AS long_description,
  [t] AS ranks,
  g.date AS release_year,
  [t.year] AS most_recent_ranking_year,

  count(DISTINCT t) AS observed_rankings
  WHERE observed_rankings >= 5 OR observed_rankings = 1

RETURN
  name,
  long_description,
  null as observed_rankings,
  ranks,
  release_year,
  most_recent_ranking_year,
  CASE
    WHEN long_description CONTAINS 'team' THEN true
    ELSE false
    END AS contains_team_in_description;
```

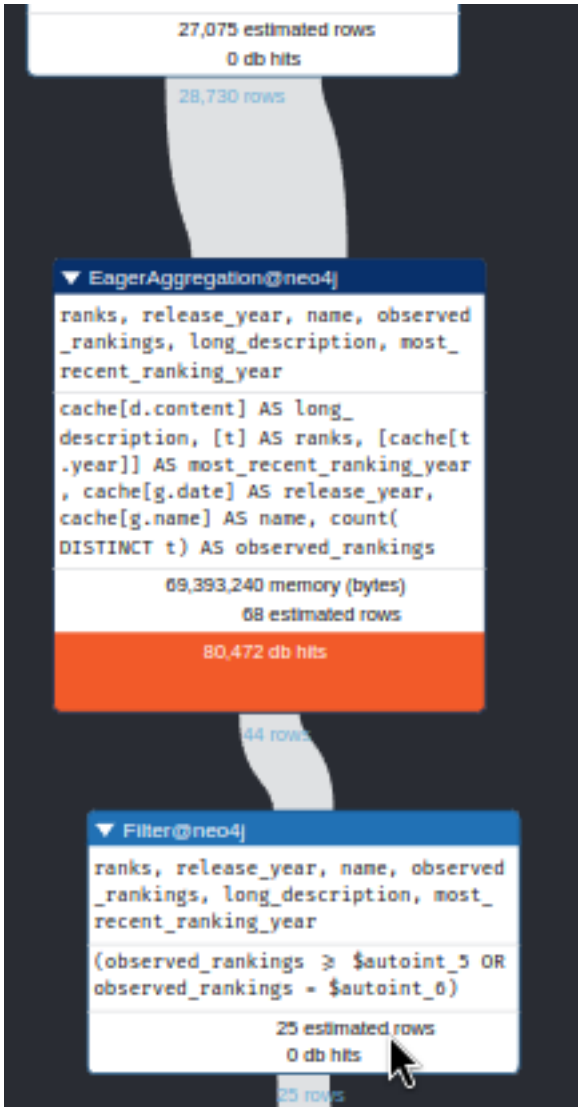
Z poniższymi indeksami:

```
CREATE INDEX creators_developer_index IF NOT EXISTS FOR (d:Creators) ON (d.developer);
CREATE TEXT INDEX  long_description_fulltext_index FOR (d:LongDescription) ON (d.content);
CREATE TEXT INDEX  short_description_content_index FOR (d:ShortDescription) ON (d.content);
CREATE TEXT INDEX  about_description_content_index FOR (d:AboutGame) ON (d.content);
CREATE INDEX game_index IF NOT EXISTS FOR (g:Game) ON (g.name);
CREATE INDEX t_rank_index IF NOT EXISTS FOR (t:TwitchRank) ON (t.rank, t.year);
CREATE INDEX t_month_index IF NOT EXISTS FOR (t:TwitchRank) ON (t.month);
```

Tab. 4: Wyniki z indeksami i bez indeksów.

\	Indexed time [ms]	Non indexed time [ms]
Med	210	334

Należy nadmienić, że dla wszystkich tych powyższych zapytań, nie była limitowany rozmiar zwrotu. Czy może mieć to znaczenie dla szybkości wykonania zapytania ? Sprawdźmy plan wykonania dla zapytania 4:



Na powyższym obrazku widać fragment planu wykonania dla zapytania 4. Widać spadek z 28 tysięcy wierzszy na 25 w ciągu dwóch kroków. I faktycznie szybkość wykonania zapytania dla była mniejsza niż przy pełnym zwrocie.

Tab. 5: Wyniki z indeksami i bez indeksów.

\	Indexed time [ms]	Non indexed time [ms]
Med	177	289

Wygląda więc na to, że **wczesne** filtrowanie wierszy w planie pozwala na szybsze jego wykonanie. Artykuł *Query tuning* ze strony `neo4j` potwierdza to spostrzeżenie.

„*Queries should aim to filter data as early as possible in order to reduce the amount of work that has to be done in the later stages of query execution.*“ [4]

Na te środki składają się między innymi `WHERE`, `CONTAINS`, `ENDS WITH` etc. na **zaindeksowanych** polach czyli to co w gruncie rzeczy chciałem osiągnąć przy zapytaniu 3 i 4. Dodatkowo `neo4j` posiada możliwość dostosowanie rozmiaru cache oraz jej współdzielenia [5]. W przypadku braku dodatkowej konfiguracji, każda baza ma własny cache, a jej dzielenie nie jest włączone.

Można je skonfigurować używając zmiennych :

```
server.memory.query_cache.shared_cache_num_entries
server.memory.query_cache.per_db_cache_num_entries
server.memory.query_cache.sharing_enabled
```

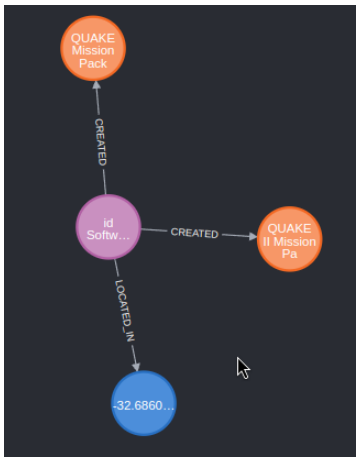
Wszystkie z tych poza środkową są dostępne jedynie w wersji enterprise, więc nie mogłem ich przetestować.

5 Przygotowanie danych przestrzennych

W celu wykonania punktów w związku z danymi przestrzennymi uznałem, że zgodnie z tematem będzie dodać do studiów gier `Creators` dane geograficzne.

```
MATCH (studio:Creators)
WITH studio,
     rand() * 180 - 90 AS latitude,
     rand() * 360 - 180 AS longitude
MERGE (loc:Location {latitude: latitude, longitude: longitude})
MERGE (studio)-[:LOCATED_IN]->(loc);
```

Rezultat:



5.1 Wykorzystanie funkcji przestrzennych

Dzięki temui możemy sprawdzić, które studia są w dystansie 1000 km od Warszawy.

```
WITH point({latitude: 52.2297, longitude: 21.0122}) AS warsaw_point, 1000 AS max_distance_km
MATCH (studio:Creators)-[:LOCATED_IN]->(loc:Location)
WITH studio, warsaw_point, point({latitude: loc.latitude, longitude: loc.longitude}) AS
studio_point, loc
  WHERE point.distance(warsaw_point, studio_point) <= max_distance_km * 1000
RETURN studio.developer AS studio_name, loc.latitude AS latitude, loc.longitude AS longitude,
       point.distance(warsaw_point, studio_point) / 1000 AS distance_in_km
  ORDER BY distance_in_km;
```

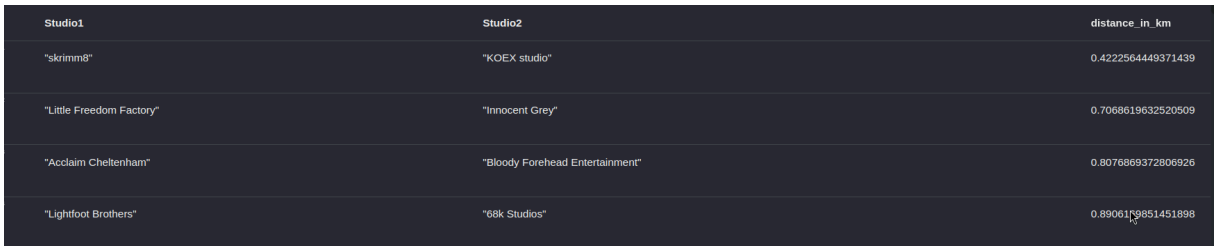
Rezultat:

studio_name	latitude	longitude	distance_in_km
"GiBar"	52.85554062982169	19.577205732609144	119.54542069931028
"KumaKumaManga"	52.68877287730638	22.880687669700023	136.6492531356526
"MERJ Media"	53.47501717069164	20.690255299356835	140.3068785873735
"Puzzling Dream"	52.82819901313343	19.063000280067143	147.85806130083714
"Greg Sergeant"	52.469358282728365	18.829578681829474	150.7880995930677
"Winter Night Games"	52.169114054280755	23.532731084447903	172.0979319760089
"Logic Games"	50.80136654950962	22.106162202230593	176.13288117299268
"Shiny Entertainment"	50.816125392475726	22.37634620062363	183.53819529079522

Można również dodać dodatkowe relacje będące bezpośrednio powiązane z lokalizacją w łatwy sposób. W poniższym przykładzie definiuję relację `NEAR` na odległość 200 km między studiami.

```
MATCH (studio1:Creators)-[:LOCATED_IN]->(loc1:Location)
WITH studio1, loc1
MATCH (studio2:Creators)-[:LOCATED_IN]->(loc2:Location)
  WHERE elementId(studio1) < elementId(studio2)
   AND point.distance(point({latitude: loc1.latitude, longitude: loc1.longitude}),
point({latitude:  loc2.latitude, longitude: loc2.
    longitude})) <= 200 * 1000
RETURN studio1.developer AS Studio1, studio2.developer AS Studio2,
       point.distance(point({latitude: loc1.latitude, longitude: loc1.longitude}),
point({latitude:  loc2.latitude, longitude: loc2.
         longitude})) / 1000 AS distance_in_km
  ORDER BY distance_in_km;
```

Można w ten sposób np. stworzyć nową relację `NEAR` pozwalającą na kojarzenie w płaszczyźnie fizycznych dystansów między studiami.



Studio1	Studio2	distance_in_km
"skrimm8"	"KOEX studio"	0.4222564449371439
"Little Freedom Factory"	"Innocent Grey"	0.7068619632520509
"Acclaim Cheltenham"	"Bloody Forehead Entertainment"	0.8076869372806926
"Lightfoot Brothers"	"68k Studios"	0.8906119851451898

Sprawdziłem wydajność pierwszego zapytania przestrzennego dla indeksu:

```
CREATE INDEX location_index IF NOT EXISTS FOR (l:Location) on (l.longitude,l.latitude);
```

Tab. 6: Wyniki z indeksami i bez indeksów.

\	Indexed time [ms]	Non indexed time [ms]
Med	12	37

Faktycznie skan indeksów przyspiesza wykonanie zapytania, jednak zapytania przestrzenne wymagają dużej ilości obliczeń na bierząco. Jest to w pewnym sensie dobrze bowiem obliczenia nie trwają tak długo jak wydobycie dużej ilości danych z dysku. Jednak w niektórych wypadkach, takich jak powtarzalna duża ilość obliczeń dla niewielkiej ilości danych zwrotnych, raczej zachęca do utwierdzenia wyniku obliczeń w formie nowej relacji (np. `NEAR`).

Zapytania przestrzenne mogłyby znaleźć szerokie zastosowanie we wszelkich rozwiązaniach IOT np. przy komunikacji miejskiej, gdzie wiele różnych pojazdów na wielu różnych trasach wymagało wykorzystania kilku różnych konwencjonalnych baz danych (np. przy połączeniach tramwaj plus autobus). W ramach tego przykładu identyfikatory pojazdów oraz lokalizacje przystanków mogłyby być indeksowane. Potencjał takiego rozwiązania jest duży.

5.2 Stworzenie procedury

Aby utworzyć nową procedurę użytkownika należy stworzyć projekt wtyczki [6]. Dokumentacja `neo4j` zastosowała przykład z projektem w `maven`, tak też ja postąpiłem. Poniżej zamieściłem `pom.xml` projektu.

```
    <project
    xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.
0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
```

```
  <groupId>org.neo4j.example</groupId>
  <artifactId>procedure-template</artifactId>
  <version>1.0.0-SNAPSHOT</version>

  <packaging>jar</packaging>
  <name>Neo4j Procedure Template</name>
  <description>A template project for building a Neo4j Procedure</description>

  <properties>
    <java.version>17</java.version>
    <maven.compiler.release>${java.version}</maven.compiler.release>

    <neo4j.version>5.26.0</neo4j.version>
  </properties>
```

Postanowiłem, że problem który będzie rozwiązywać procedura będzie praktyczny. Każda gra na steamie korzysta ze `steam_id` w swoim urlu na stronie sklepu steam: `https://store.steampowered.com/app/{steam_id}/`. Procedura będzie pingować dla każdego `Game` czy taki url faktycznie się resolvuje i usuwaał te które się nie resolvują wraz z jej relacjami **poza** `Creator`.

```
// importy
public class SteamGameValidator {

    @Context
    public GraphDatabaseService db;

    @Procedure(name = "checkAndDeleteInvalidGames", mode = Mode.WRITE)
    @Description("Checks if each game in the database is available on Steam and deletes it
if not.")
    public void checkAndDeleteInvalidGames() {
        try (Transaction tx = db.beginTx()) {
            var games = db.findNodes(Label.label("Game"));

            while (games.hasNext()) {
                var game = games.next();
                var steamId = (String) game.getProperty("steam_id", null);

                if (steamId != null && !isSteamAppAvailable(steamId)) {
                    deleteGameAndRelatedEntities(game);
                }
            }
            tx.commit();
        }
    }
    private boolean isSteamAppAvailable(String steamId) {
        var url = "https://store.steampowered.com/app/" + steamId + "/";
        try {
            var response = sendHttpRequest(url);
            return response.getStatusLine().getStatusCode() == 200;
        } catch (Exception e) {
            return false;
        }
    }
    private HttpResponse sendHttpRequest(String url) throws IOException {
        var request = new HttpGet(url);
        var response = HttpClientBuilder.create().build().execute(request);
        return response;
    }
    private void deleteGameAndRelatedEntities(Node game) {
```

```
                    var   relationships   =   game.getRelationships(Direction.INCOMING,
RelationshipType.withName("CREATED"));
        for (Relationship relationship : relationships) {
            var creator = relationship.getStartNode();

            var allRelationships = game.getRelationships();
            allRelationships.forEach(r -> r.delete());
            game.delete();

            var relatedNodes = game.getNodesFromRelationships(Direction.INCOMING);
            relatedNodes.forEach(n ->
            {
              if (!relatedNode.equals(creator)) {
                    relatedNode.delete();
                })
              });
        }
    }
}
```

Następnie można wywołać ją przy pomocy:

```
CALL checkAndDeleteInvalidGames();
```

Niestety dockerowa wersja nie pozwalała na uruchomić procedury, na wirtualnej maszynie również. Jakkolwiek rozumiem, że wersja dockerowa może mieć jakieś dziwne blokady niezmienialności (ale importy csv zadziałały, więc szalenie dziwna sytuacja), to wersja na wirtualnej maszynie powinna raczej działać. Jako źródło naczelne wykorzystałem dokumentację oraz blog deweloperski [7]. Wszystkie kroki wykonałem zgodnie z zaleceniami, nawet downgradeowałem w dockerowych wersjach bazy z wersji 5 na niższe ale nic nie dało rezultatów. Instalować drugi raz bazy na vmce uznałem, że nie ma sensu skoro inne wersje dockerowych obrazów dają taki sam wynik bez zmian chociażby błędów.

Różnice w procedurach między `neo4j`, a tradycyjnymi relacyjnymi bazami danych wiąże się z kilkoma jakościowymi cechami. Najważniejszą rzeczą z punktu widzenia dewelopera jest to, że można napisać je w powszechnie używanym języku programowania, którym jest Java, można też to zrobić w Kotlinie [8], oraz Pythonie, JSie czy C#. Pozwala to zachować zasady biznesowe w obrębie jednej bazy kodu co jest szalenie przydatną rzeczą w warunkach biznesowych. Postgres miał możliwość napisania procedury w C, ale powiedzmy sobie szczerze - raczej niewiele osób w warunkach biznesowych byłoby skłonnych do pisania procedury w tym języku.

Możliwości przy pisaniu procedur są zorientowane wokół operacji na grafach co za tym idzie posiada on możliwości wynajdywania powiązań, ścieżek oraz sąsiedztw między węzłami. Jest to ogólnie przydatne w zasadzie do każdego przypadku użycia w nowoczesnych aplikacjach, w których występują gęste powiązania dziedzinowe na wielu płaszczyznach.

6 Analiza zbioru danych oraz rozmieszczenie danych

W przypadku mojego zbioru danych, moim zdaniem **nie ma sensu** na podstawie samych danych rozmieszczać ich w kilku regionach na fizycznych maszynach przy rozwiązaniu produkcyjnym. W gruncie rzeczy to dane z bazy reprezentują centralne repozytorium wiedzy. Jednak możliwe jest, że przy okazji analizy **ruchu** danych mogłoby wyjść na jaw kilka zależności, np. to, że niektóre gry są streamowane częściej w Europie niż w Azji (np. gry arcade, drużynowe z wysokim progiem rywalizacji), tak samo z konkretnymi grami.

Nie wiąże się to jednak z wydzieleniem żadnych konkretnych podgrafów, chyba, że tych, które są powiązane z konkretną grą w ramach wcześniejszych rozważań. Wtedy, przykładowo, koreański Counter-Strike, który jest dystrybuowany wyłącznie na rynek koreański poprzez wykluczenie powinien mieć dane o sobie zamieszczone bliżej Korei itd.

Neo4j posiada szerokie możliwości klasteryzacji swoich instancji np. w ramach replikacji [9]. Dysponuje on również czymś takim jak Casual Cluster, który jest specjalnie zaprojektowanych rozwiązaniem pod kątem regionalizacji danych. Ma on również możliwości ustawiania węzłów replikacji, centralnych itd.

Niestety nie zdołałem tych możliwości przetestować bo takie możliwości są wyłącznie dostępne w wersji enterprise.

7 Bibilioteka APOC

APOC jest biblioteką narzędzi dla neo4j, która dodaje wiele przydatnych narzędzi, procedur i funkcji rozszerzające język Cypher. Poniżej zamieściłem przegląd najciekawszych lub najprzydatniejszych funkcji z APOC.

7.0.1 Manipulacje stringami

Do nich należą funkcje takie jak apoc.text.join, apoc.text.split, apoc.test.replace. Przykładowo:

```
RETURN apoc.text.join(["con", "ca", "ted"], "-") AS message
```

Równie ciekawy jest apoc.text.clean usuwający znaki spacji, apostrofów i innych rzeczy typowo podchodzących pod sanityzację wejścia. Można z niego skorzystać też w innych zapytaniach przy prezentacji danych.

7.0.2 Wczytywanie CSV i JSON-ów

Wcześniej użyłem LOAD CSV, ale APOC posiada także własną wersję, która pozwala na ściąganie danych bezpośrednio poprzez http bez kopiowania do folderu /import.

```
CALL apoc.load.csv("https://www.kaggle.com/datasets/rankirsh/evolution-of-top-games-on-twitch?
select=Twitch_game_data.csv") YIELD map AS row
// i tutaj jak przy wcześniejszym imporcie
```

Podobnie można zrobić z plikami json:

```
CALL apoc.load.json("https://www.kaggle.com/datasets/rankirsh/evolution-of-top-games-on-
twitch?select=Twitch_game_data.json") YIELD map AS row
// i tutaj jak przy wcześniejszym imporcie
```

7.0.3 Strukturyzacja danych

Jak wiadomo, json jest hierarchicznym typem danych. APOC pozwala na interpretowanie json-owych hierarchii w formie drzewa:

```
CALL apoc.convert.toTree($json) YIELD value
RETURN value
```

Ma on także opcje kontroli głębokości przechodzenia przez hierarchię, kierunek oraz filtrowanie.

7.0.4 Manipulacja i edycja schematów i etykiet

APOC posiada możliwość zmiany etykiety poprzez apoc.refactor.rename.label:

```
CALL apoc.refactor.rename.label('Location', 'StudioLocation')
```

Także dostępny jest podgląd modelu schematu poprzez apoc.meta.schema()

```
CALL apoc.meta.schema() YIELD value
RETURN keys(value) AS types
```

Źródła

[1] „The impact of indexes on query performance - Cypher Manual“. Zugegriffen: 30. Dezember 2024. [Online]. Verfügbar unter: https://neo4j.com/docs/cypher-manual/5/indexes/search-performance-indexes/using-indexes/

[2] „Indexes - Cypher Manual“. Zugegriffen: 30. Dezember 2024. [Online]. Verfügbar unter: https://neo4j.com/docs/cypher-manual/5/indexes/

[3] „Search-performance indexes - Cypher Manual“. Zugegriffen: 30. Dezember 2024. [Online]. Verfügbar unter: https://neo4j.com/docs/cypher-manual/5/indexes/search-performance-indexes/overview/

[4] „Query tuning - Cypher Manual“. Zugegriffen: 30. Dezember 2024. [Online]. Verfügbar unter: https://neo4j.com/docs/cypher-manual/5/planning-and-tuning/query-tuning/

[5] „Query caches - Cypher Manual“. Zugegriffen: 30. Dezember 2024. [Online]. Verfügbar unter: https://neo4j.com/docs/cypher-manual/5/query-caches/

[6] „Setting up a plugin project - Java Reference“. Zugegriffen: 31. Dezember 2024. [Online]. Verfügbar unter: https://neo4j.com/docs/java-reference/5/extending-neo4j/project-setup/

[7] „Let’s Write a Stored Procedure in Neo4j – Part I“. Zugegriffen: 31. Dezember 2024. [Online]. Verfügbar unter: https://neo4j.com/developer-blog/lets-write-a-stored-procedure-in-neo4j-part-i/

[8] M. Falcier, „mfalcier/neo4j-kotlin-procedure-example“. Zugegriffen: 31. Dezember 2024. [Online]. Verfügbar unter: https://github.com/mfalcier/neo4j-kotlin-procedure-example

[9] „Introduction - Operations Manual“. Zugegriffen: 31. Dezember 2024. [Online]. Verfügbar unter: https://neo4j.com/docs/operations-manual/5/clustering/introduction/