

Sprawozdanie – Etap IV

Wykorzystanie bazy grafowej Neo4j

Wybrany zbiór danych

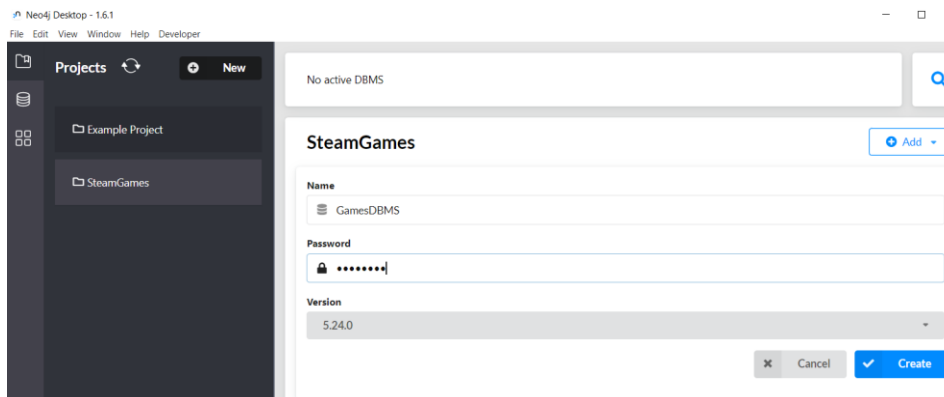
Zdecydowano się na zbiór danych dotyczący gier sprzedawanych na platformie Steam w latach 2006-2023: <https://www.kaggle.com/datasets/whigmawhim/steam-releases>.

Zbiór rozszerzony dotyczy nagród zdobytych przez gry w latach 2014-2019: <https://www.kaggle.com/datasets/unanimad/the-game-awards>.

Wybrano zbiór dotyczący gier na Steam, ponieważ był odpowiednich rozmiarów (posiada ponad 67 tysiące unikatowych gier), niemałą ilość pól (20 pól) i nie posiadał błędów ani nieprawidłowości. Ponadto, zbiór podstawowy wydawał się wszechstronnie rozszerzalny. Jako zbiór rozszerzony wybrano zbiór dotyczący nagród, ponieważ dobrze pasował do zbioru podstawowego i był odpowiednich rozmiarów (ponad 700 danych).

Instalacja Neo4j

Zainstalowano Neo4j Desktop w wersji 1.6.1 i bazę danych Neo4j w wersji 5.24 (najnowsze dostępne wersje). Stworzono projekt SteamGames, w którym stworzono bazę danych GamesDBMS.



Import danych i zbiór rozszerzony

Zdecydowano się na podział podstawowego zbioru danych na 6 node'ów:

- Game (gra) – właściwości (properties): nazwa, link (w Steam), data wydania.
- Developer – właściwości: nazwa (name)
- Publisher (wydawca) – właściwości: nazwa (name)
- Genre (gatunek) – właściwości: nazwa (name)

- PlayStats (dane dotyczące statystyk graczy) - właściwości: positive (ilość pozytywnych opinii), negative (ilość negatywnych opinii), total (sumaryczna ilość opinii), rating, review percentage (ocena procentowa z opinii)

- ReviewStats (dane dotyczące statystyk opinii) - właściwości: peak players (największa osiągnięta liczba graczy), players right now (ilość obecnych graczy), 24h peak (peak graczy w ciągu 24 godzin), all time peak (peak w całej historii gry), all time peak date (data wcześniej wymienionego peak'u).

Zbiór rozszerzony zawiera informację na temat nagród dla gier oraz informacje na temat nagród przyznawanym danym osobom w świecie gier (np. najlepszy host esportowy) oraz przyznawanych eventom. Dlatego też, zdecydowano się na następujący podział typów węzłów:

- Award – właściwości: category, votedBy oraz year

- EsportsTeam – właściwości: name

- Event – właściwości: name

- Person – właściwości: firstName, lastName, nickname i profession

Zaimportowano dane, najpierw z zbioru podstawowego a następnie z zbioru rozszerzonego, za pomocą funkcji LOAD CSV, poprzez konsolę Neo4j Browser (dostępną z Neo4j Desktop).

Wprowadzano dane po kolei, po jednym Node'dzie na raz.

Ponieważ pierwsze próby importu danych dotyczących gier zajmowały bardzo dużo czasu (w zbiorze jest ponad 67 tysięcy gier), zastopowano operację i stworzono najpierw UNIQUE CONSTRAINT na link'u do gry (ponieważ w przeciwieństwie do np. nazwy gry, jest on unikatowy). Po tej operacji, dane dotyczące gier zaimportowały się bardzo szybko (kilka minut). Podobną operację przeprowadzono dla danych dotyczących wydawców gier (stworzono UNIQUE CONSTRAINT na nazwie wydawcy).

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

LOAD CSV WITH HEADERS FROM 'file:///steam.csv' AS row
MERGE (g:Game {
  link: row.link
})
ON CREATE SET g.name = row.game,
              g.release = row.release,
              g.detectedTechnologies = row.detected_technologies;

CREATE CONSTRAINT publisher_name_unique IF NOT EXISTS
FOR (p:Publisher)
  REQUIRE p.name IS UNIQUE;

LOAD CSV WITH HEADERS FROM 'file:///steam.csv' AS row
MATCH (g:Game { link: row.link })
MERGE (p:Publisher { name: row.publisher })
MERGE (g)-[:PUBLISHED_BY]->(p);

LOAD CSV WITH HEADERS FROM 'file:///steam.csv' AS row
MATCH (g:Game { link: row.link })
MERGE (d:Developer { name: row.developer })
MERGE (g)-[:DEVELOPED_BY]->(d);
```

Jak widać na zamieszczonym wyżej obrazku, Game ma relację PUBLISHED_BY z Publisher i DEVELOPED_BY z developerem. Ponieważ import danych dotyczących statystyk graczy i opinii również zajmował dużo czasu, skorzystano z opcji IN TRANSACTIONS OF 1000 ROWS, które spowodowały, że po imporcie każdego 1000 sztuk danych następował commit.

```
:auto LOAD CSV WITH HEADERS FROM 'file:///steam.csv' AS row
CALL {
  WITH row
  MATCH (g:Game { link: row.link })
  MERGE (r:ReviewStats {
    positiveReviews: coalesce(toInteger(row.positive_reviews), 0),
    negativeReviews: coalesce(toInteger(row.negative_reviews), 0),
    totalReviews: coalesce(toInteger(row.total_reviews), 0),
    rating: coalesce(toFloat(row.rating), 0.0),
    reviewPercentage: coalesce(toFloat(row.review_percentage), 0.0)
  })
  MERGE (g)-[:HAS_REVIEW]->(r)
} IN TRANSACTIONS OF 1000 ROWS;

:auto LOAD CSV WITH HEADERS FROM 'file:///steam.csv' AS row
CALL {
  WITH row
  MATCH (g:Game { link: row.link })
  MERGE (ps:PlayerStats {
    peakPlayers: coalesce(toInteger(row.peak_players), 0),
    playersRightNow: coalesce(toInteger(row.players_right_now), 0),
    '24HourPeak': coalesce(toInteger(row.'24_hour_peak'), 0),
    allTimePeak: coalesce(toInteger(row.all_time_peak), 0),
    allTimePeakDate: CASE WHEN row.all_time_peak_date IS NULL OR row.all_time_peak_date = '' THEN '' ELSE row.all_time_peak_date END
  })
  MERGE (g)-[:HAS_PLAYER_STATS]->(ps)
} IN TRANSACTIONS OF 1000 ROWS;
```

Game ma relację HAS_PLAYER_STATS z PlayerStats i HAS_REVIEW z ReviewStats.

W celu przyspieszenia importu danych, stworzono również indeks na nazwę gatunku gier przed importem danych.

```
CREATE INDEX genre_name_index IF NOT EXISTS
FOR (g:Genre)
ON (g.name);

:auto LOAD CSV WITH HEADERS FROM 'file:///steam.csv' AS row
CALL {
  WITH row
  MATCH (g:Game { link: row.link })
  MERGE (gen:Genre { name: coalesce(row.primary_genre, '') })
  MERGE (g)-[:HAS_PRIMARY_GENRE]->(gen)
} IN TRANSACTIONS OF 1000 ROWS;

:auto LOAD CSV WITH HEADERS FROM 'file:///steam.csv' AS row
CALL {
  WITH row
  MATCH (g:Game { link: row.link })
  WITH g, SPLIT(row.store_genres, ',') AS genres
  UNWIND genres AS genreName
  WITH g, TRIM(genreName) AS trimmedGenreName
  MERGE (gen:Genre { name: trimmedGenreName })
  MERGE (g)-[:HAS_STORE_GENRE]->(gen)
} IN TRANSACTIONS OF 1000 ROWS;
```

Game (gra) może mieć jeden gatunek główny i kilka gatunków sklepowych (przypisanych przez sklep Steam). Dlatego też, Game ma 2 rodzaje relacji z Genre: HAS_PRIMARY_GENRE i HAS_STORE_GENRE.

Następnie, wprowadzono dane z zbioru rozszerzającego:

```
CREATE INDEX genre_name_index IF NOT EXISTS
FOR (g:Genre)
ON (g.name);

:auto LOAD CSV WITH HEADERS FROM 'file:///steam.csv' AS row
CALL {
    WITH row
    MATCH (g:Game { link: row.link })
    MERGE (gen:Genre { name: coalesce(row.primary_genre, '') })
    MERGE (g)-[:HAS_PRIMARY_GENRE]->(gen)
} IN TRANSACTIONS OF 1000 ROWS;

:auto LOAD CSV WITH HEADERS FROM 'file:///steam.csv' AS row
CALL {
    WITH row
    MATCH (g:Game { link: row.link })
    WITH g, SPLIT(row.store_genres, ',') AS genres
    UNWIND genres AS genreName
    WITH g, TRIM(genreName) AS trimmedGenreName
    MERGE (gen:Genre { name: trimmedGenreName })
    MERGE (g)-[:HAS_STORE_GENRE]->(gen)
} IN TRANSACTIONS OF 1000 ROWS;
```

```
:auto LOAD CSV WITH HEADERS FROM 'file:///awards.csv' AS row
CALL {
    WITH row
    MATCH (g:Game { name: row.nominee })
    MERGE (a:Award {
        year: toInteger(row.year),
        category: row.category,
        votedBy: row.voted
    })
    MERGE (g)-[:NOMINATED_FOR]->(a)
    FOREACH (_ IN CASE WHEN row.winner = '1' THEN [1] ELSE [] END |
        MERGE (g)-[:WON]->(a)
    )
} IN TRANSACTIONS OF 1000 ROWS;

//////////
MERGE (a:Award {
    year: 2019,
    category: 'Global Gaming Citizens',
    votedBy: 'jury'
})

MERGE (p1:Person {
    firstName: 'Stephen',
    lastName: 'Machuga',
    profession: '',
    nickname: ''
})
MERGE (p1)-[:NOMINATED_FOR]->(a)
MERGE (p1)-[:WON]->(a);
```

```
LOAD CSV WITH HEADERS FROM 'file:///awards.csv' AS row
WITH row
WHERE row.category = 'Best Esports Host'
WITH row, SPLIT(row.nominee, ' ') AS nameParts
WITH row, nameParts,
    nameParts[0] AS firstName,
    nameParts[-1] AS lastName,
    CASE
        WHEN SIZE(nameParts) > 2 AND nameParts[1] STARTS WITH "'" AND nameParts[1] ENDS WITH "'" THEN SUBSTRING(nameParts[1], 1, SIZE(nameParts[1]) - 2)
        ELSE NULL
    END AS nickname
WITH row, firstName, lastName, coalesce(nickname, '') AS nickname
MERGE (a:Award {
    year: toInteger(row.year),
    category: row.category,
    votedBy: row.voted
})
MERGE (p:Person {
    firstName: firstName,
    lastName: lastName,
    profession: 'Event host',
    nickname: nickname
})
MERGE (p)-[:NOMINATED_FOR]->(a)
FOREACH (_ IN CASE WHEN row.winner = '1' THEN [1] ELSE [] END |
    MERGE (p)-[:WON]->(a)
);
```

(Ze względu na długość skryptu importującego dane, nie umieszczono go w pełni w sprawozdaniu. Całość znajduje się w pliku loading.txt, załączonym wraz z sprawozdaniem).

Person ma relację NOMINATED_FOR i, jeśli dana osoba wygrała nagrodę, relację WON z Award. Jeśli jest to coach zespołu Esportowego lub członek zespołu Esportowego, mają oni też relację z EsportTeam (odpowiednio COACH_OF i MEMBER_OF). Gry też mają relacje NOMINATED_FOR i WON z Award.

5 nowych węzłów

Ponieważ skorzystano z gotowego rozszerzonego zbioru danych, wprowadzono 5 nowych węzłów do bazy danych – 2 dotyczące gier, które połączono z odpowiednimi wydawcami, developerami, gatunkami i nagrodami. Stworzono też jeden węzeł developera związany z jedną z nowych gier. Wprowadzono też jedną nagrodę którą powiazano z nowymi grami, i jedną którą powiazano z istniejącą w bazie już grą.

The screenshot displays the Neo4j Cypher query interface with three separate query executions. Each execution shows the query text, a status message, and a list of results with green checkmarks indicating successful completion.

Query 1:

```
neo4j$ MERGE (g1:Game { link: '/app/2531310/' }) MERGE (g2:Game { link: '/app/2215430/' }) MERGE (a:Award { year: 2020, category: 'Gam...
```

Added 1 label, created 1 node, set 3 properties, created 3 relationships, completed after 20 ms.

Query 2:

```
$ MERGE (g2:Game { link: '/app/2215430/' }) ON CREATE SET g2.name = 'Ghost of Tsushima', g2.release = '2020-06-20', g2.detectedTechnol...
```

neo4j\$ MERGE (g2:Game { link: '/app/2215430/' }) ON CREATE SET g2.name = 'Ghost of Tsushima', g2.release = '2020-06-20', g2.det... ✓

neo4j\$ MERGE (p2:Publisher { name: 'PlayStation PC LLC' }) MERGE (g2)-[:PUBLISHED_BY]-(p2) MERGE (d2:Developer { name: 'Sucker... ✓

Query 3:

```
$ MERGE (g1:Game { link: '/app/2531310/' }) ON CREATE SET g1.name = 'The Last of Us Part II', g1.release = '2020-06-19', g1.detectedTe...
```

neo4j\$ MERGE (g1:Game { link: '/app/2531310/' }) ON CREATE SET g1.name = 'The Last of Us Part II', g1.release = '2020-06-19', g... ✓

neo4j\$ MERGE (p1:Publisher { name: 'PlayStation PC LLC' }) MERGE (g1)-[:PUBLISHED_BY]-(p1) MERGE (d1:Developer { name: 'Naught... ✓

Query 4:

```
neo4j$ MERGE (a2:Award { year: 2021, category: 'Game of the Year', votedBy: 'jury' }) MERGE (g3:Game { link: '/app/1426210/' }) MERGE ...
```

Added 1 label, created 1 node, set 3 properties, created 2 relationships, completed after 12 ms.

Zapytania

Zapytanie 1

```

1 MATCH (g:Game)-[:WON]→(a:Award)
2 WHERE a.year = 2017
3 RETURN g.name AS name, a.category AS category, a.year AS year
4 UNION
5 MATCH (p:Person)-[:WON]→(a:Award)
6 WHERE a.year = 2017
7 RETURN p.firstName + ' ' + p.lastName AS name, a.category AS category, a.year AS year;
8

```

	name	category	year
1	"What Remains of Edith Finch"	"Best Narrative"	2017
2	"Cuphead"	"Best Art Direction"	2017
3	"Hellblade: Senua's Sacrifice"	"Best Audio Design"	2017
4	"Hellblade: Senua's Sacrifice"	"Games for Impact"	2017
5	"Cuphead"	"Best Independent Game"	2017
6	"Monument Valley 2"	"Best Mobile Game"	2017

Zapytanie 1 dotyczy nagród wygranych przez gry i ludzi w 2017 roku. Wykorzystuje ono UNION, tym samym zwracając informacje na temat wygranych nagród zarówno przez gry jak i ludzi.

Zapytanie 2

```

1 MATCH (p:Person { firstName: 'Eefje', lastName: 'Depoortere' })-[:WON]→(a:Award)
2 RETURN a.category AS category, a.year AS year, a.votedBy AS votedBy;
3

```

	category	year	votedBy
1	"Best Esports Host"	2018	"jury"
2	"Best Esports Host"	2019	"jury"

Zapytanie 2 dotyczy Eefje Depoortere – wyświetlamy wygrane przez niego nagrody.

Zapytanie 3

```

1 MATCH (p:Person)-[:NOMINATED_FOR]→(a:Award { category: 'Best Esports Host', year: 2018 })
2 RETURN p.firstName AS firstName, p.lastName AS lastName, a.year AS year, a.votedBy AS votedBy;

```

	firstName	lastName	year	votedBy
1	"Eefje"	"Depoortere"	2018	"jury"
2	"Alex"	"Mendez"	2018	"jury"
3	"Alex"	"Richardson"	2018	"jury"
4	"Anders"	"Blume"	2018	"jury"
5	"Paul"	"Chaloner"	2018	"jury"

W zapytaniu 3, wyświetlamy wszystkie nominowane osoby do nagrody Best Esports Host w roku 2018.

Zapytanie 4

```

1 MATCH (g:Game)-[:WON]→(a:Award { category: 'Game of the Year' })
2 RETURN g.name AS gameName, a.year AS awardYear, a.votedBy AS votedBy
3 ORDER BY a.year;

```

	gameName	awardYear	votedBy
1	"The Witcher 3: Wild Hunt"	2015	"jury"
2	"God of War"	2018	"jury"
3	"Ghost of Tsushima"	2020	"jury"
4	"It Takes Two"	2021	"jury"

Zapytanie 4 dotyczy gier które wygrały nagrodę Game of the Year.

Zapytanie 5

```

1 MERGE (p:Person { firstName: 'Kim', lastName: 'Jeong-gyun' })
2 ON CREATE SET p.profession = 'Esports Coach', p.nickname = 'kkOma'
3 RETURN p.firstName AS firstName, p.lastName AS lastName, p.profession AS profession, p.nickname AS nickname;

```

	firstName	lastName	profession	nickname
1	"Kim"	"Jeong-gyun"	"Esports Coach"	"kkOma"

Zapytanie 5 dotyczy Kim'a Jeong-gyun'a. Znajdujemy go za pomocą MERGE, a następnie ustaiwamy mu odpowiedni zawód i nickname.

Wszystkie zapytania są w załączonym pliku queries.txt.

Indeksy

Podczas importu danych, w celu przyspieszenia procesu, stworzono kilka indeksów. Na czas testów w zadaniu związanym z indeksami, usunięto je aby nie wpływały na wyniki.

neo4j\$

neo4j\$ DROP INDEX developer_name_unique IF EXISTS

neo4j\$ DROP CONSTRAINT game_link_unique IF EXISTS

neo4j\$ DROP INDEX game_link_unique IF EXISTS

neo4j\$ DROP INDEX game_name_index IF EXISTS

neo4j\$ DROP CONSTRAINT genre_name_unique IF EXISTS

neo4j\$ DROP INDEX genre_name_unique IF EXISTS

neo4j\$ DROP CONSTRAINT publisher_name_unique IF EXISTS

neo4j\$ DROP INDEX publisher_name_unique IF EXISTS

neo4j\$ SHOW INDEXES

Table

Text

Code

	id	name	state	populationPercent	type	entityType	labelsOrTypes	properties	indexProvider	owningConstraint	lastRead
1	7	"developer_name_unique"	"ONLINE"	100.0	"RANGE"	"NODE"	["Developer"]	["name"]	"range-1.0"	"developer_name_unique"	"2025-01-07T19:18:56.41900"
2	5	"game_link_unique"	"ONLINE"	100.0	"RANGE"	"NODE"	["Game"]	["link"]	"range-1.0"	"game_link_unique"	"2025-01-07T19:22:53.41900"
3	4	"game_name_index"	"ONLINE"	100.0	"RANGE"	"NODE"	["Game"]	["name"]	"range-1.0"	null	"2025-01-07T17:09:30.65800"
4	9	"genre_name_unique"	"ONLINE"	100.0	"RANGE"	"NODE"	["Genre"]	["name"]	"range-1.0"	"genre_name_unique"	"2025-01-07T19:18:56.42100"
5	0	"index_343aff4e"	"ONLINE"	100.0	"LOOKUP"	"NODE"	null	null	"token-lookup-1.0"	null	"2025-01-07T19:46:40.88200"

Stworzono następujące indeksy w celu optymalizacji moich zapytań:

- indeks na nazwie gry
- indeks na link'u gry
- indeks na imieniu osoby
- indeks na zawodzie osoby
- indeks na kategorii nagrody gry
- indeks na roku nagrody
- indeks na rodzaju głosowania w nagrodzie

- indeks na nazwie eventu

```
$ CREATE INDEX game_name_index IF NOT EXISTS FOR (g:Game) ON (g.name); CREATE INDEX game_link_index IF NOT EXISTS FOR (g:Game) ON (g.li...
```

neo4j\$ CREATE INDEX game_name_index IF NOT EXISTS FOR (g:Game) ON (g.name)	✓
neo4j\$ CREATE INDEX game_link_index IF NOT EXISTS FOR (g:Game) ON (g.link)	✓
neo4j\$ CREATE INDEX person_name_index IF NOT EXISTS FOR (p:Person) ON (p.firstName, p.lastName)	✓
neo4j\$ CREATE INDEX person_profession_index IF NOT EXISTS FOR (p:Person) ON (p.profession)	✓
neo4j\$ CREATE INDEX award_category_index IF NOT EXISTS FOR (a:Award) ON (a.category)	✓
neo4j\$ CREATE INDEX award_year_index IF NOT EXISTS FOR (a:Award) ON (a.year)	✓
neo4j\$ CREATE INDEX award_votedBy_index IF NOT EXISTS FOR (a:Award) ON (a.votedBy)	✓
neo4j\$ CREATE INDEX event_name_index IF NOT EXISTS FOR (e:Event) ON (e.name)	✓

Poniżej przedstawiono czasy wykonania zapytań uzyskane przed i po dodaniu indeksów.

Zapytanie	Czas – przed indeksami	Czas – po indeksach
1	122 ms	118 ms
2	60 ms	29 ms
3	64 ms	63 ms
4	67 ms	54 ms
5	64 ms	46 ms

Użyto komendy PROFILE przed zapytaniem, co też spowodowało ich wykonanie – lecz użyto tej komendy przed wszystkimi zapytaniem, więc wyniki przed i po powinny być wiarygodne.

Jak widać, czas wykonania dla zapytania nr. 2 skrócił się aż o połowę. W przypadku reszty zapytań, czasy wykonania również się skróciły, choć nie aż o tyle.

Ze względu na małą różnicę w czasie wykonania w przypadku niektórych zapytań, przeprowadzono test – usunięto indeksy na Person i przeprowadzono testy ponownie:

Bez indeksów na person:

Zapytanie	Czas – przed indeksami	Czas – po indeksach
1	122 ms	112 ms
2	60 ms	19 ms
3	64 ms	29 ms
4	67 ms	17 ms
5	64 ms	2 ms

Wyniki przed vs. po dodaniu indeksów po wykluczeniu indeksów Person wyglądają dużo lepiej niż z włączeniem indeksów Person. W szczególności czas wykonania zapytania 5 się bardzo skrócił. Czas wykonania zapytania 1 nie skrócił się zbyt (jedynie o 10 ms), lecz jest to i tak lepszy wynik niż w poprzednim teście. Może to wynikać z relatywnie małej ilości danych Person lub z wewnętrznym wyborów Neo4j związanych z wykonaniem zapytania. Indeks na votedBy zapewne był niepotrzebny.

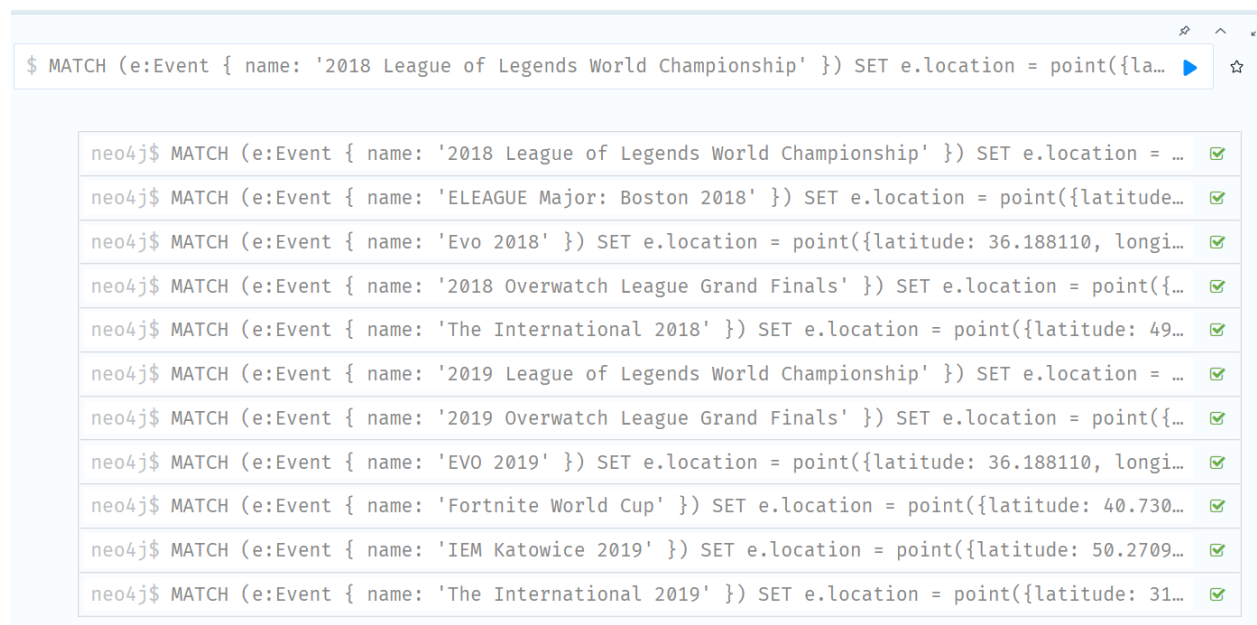
Indeksy są w pliku indexes.txt, załączonym wraz z sprawozdaniem.

Omów jakie jeszcze indeksy warto stworzyć w wykorzystywanym zbiorze danych i dlaczego.

Oprócz indeksów stworzonych do optymalizacji wymienionych wyżej zapytań, warto stworzyć indeksy na Genre, Publisher i Developer (na name w przypadku każdego). Może to być przydatne w przypadku np. operacji MERGE lub przy wyszukiwaniu gier po ich gatunkach, wydawcach, developerach.

Dane przestrzenne

Dodano dane przestrzenne do węzłów typu Event. Każde wydarzenie odbywało się w jakimś konkretnym mieście, a więc dołączono ich lokalizację do danych. Dodano właściwość o typie Point z wartościami latitude i longitude.



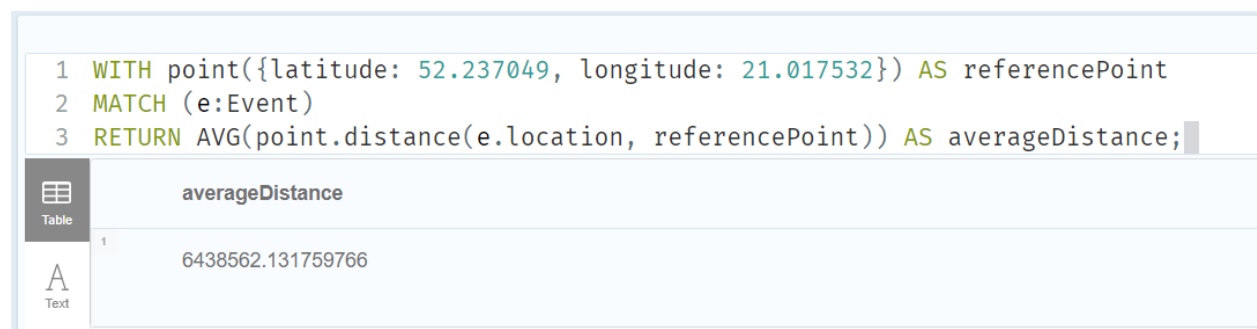
The screenshot shows a Neo4j Cypher query editor. The query is: `$ MATCH (e:Event { name: '2018 League of Legends World Championship' }) SET e.location = point({la...`. Below the query, there is a table of results. Each row shows the query being executed, the result, and a green checkmark indicating success.

Query	Result	Status
neo4j\$ MATCH (e:Event { name: '2018 League of Legends World Championship' }) SET e.location = ...		✓
neo4j\$ MATCH (e:Event { name: 'ELEAGUE Major: Boston 2018' }) SET e.location = point({latitude...		✓
neo4j\$ MATCH (e:Event { name: 'Evo 2018' }) SET e.location = point({latitude: 36.188110, longi...		✓
neo4j\$ MATCH (e:Event { name: '2018 Overwatch League Grand Finals' }) SET e.location = point({...		✓
neo4j\$ MATCH (e:Event { name: 'The International 2018' }) SET e.location = point({latitude: 49...		✓
neo4j\$ MATCH (e:Event { name: '2019 League of Legends World Championship' }) SET e.location = ...		✓
neo4j\$ MATCH (e:Event { name: '2019 Overwatch League Grand Finals' }) SET e.location = point({...		✓
neo4j\$ MATCH (e:Event { name: 'EVO 2019' }) SET e.location = point({latitude: 36.188110, longi...		✓
neo4j\$ MATCH (e:Event { name: 'Fortnite World Cup' }) SET e.location = point({latitude: 40.730...		✓
neo4j\$ MATCH (e:Event { name: 'IEM Katowice 2019' }) SET e.location = point({latitude: 50.2709...		✓
neo4j\$ MATCH (e:Event { name: 'The International 2019' }) SET e.location = point({latitude: 31...		✓

Zapytania:

Wykonano szereg zapytań przestrzennych:

Zapytanie 1:



The screenshot shows a Neo4j Cypher query editor. The query is: `1 WITH point({latitude: 52.237049, longitude: 21.017532}) AS referencePoint
2 MATCH (e:Event)
3 RETURN AVG(point.distance(e.location, referencePoint)) AS averageDistance;`. Below the query, there is a table of results. The table has one column named 'averageDistance' and one row with the value '6438562.131759766'.

averageDistance
6438562.131759766

W zapytaniu 1, zwracana jest średnia odległość eventów od Warszawy. Wynik został zwrócony w metrach. Zapytanie użyło funkcji agregującej AVG i funkcji odległościowej distance.

Zapytanie 2:

```
1 MATCH (e1:Event { name: '2018 League of Legends World Championship' }), (e2:Event { name: 'The International 2018' })
2 RETURN point.distance(e1.location, e2.location) AS distance;
```

	distance
1	8267664.105400379

Zapytanie 2 zwraca odległość między dwoma eventami. Skorzystano z funkcji odległościowej distance.

Zapytanie 3:

```
1 WITH point({latitude: 40.730610, longitude: -73.935242}) AS newYorkCity
2 MATCH (e:Event)
3 WHERE point.distance(e.location, newYorkCity) < 1000
4 RETURN COUNT(e) AS eventCount;
```

	eventCount
1	2

Zapytanie 3 odnosi się do eventów odbywających się w Nowym Jorku. Zapytanie korzysta z funkcji distance i funkcji agregującej COUNT.

Zapytanie 4:

```
1 WITH point({latitude: 52.237049, longitude: 21.017532}) AS warsaw
2 MATCH (e:Event)
3 WHERE point.distance(e.location, warsaw) < 1000000
4 RETURN e.name AS eventName, point.distance(e.location, warsaw) AS distance
5 ORDER BY distance;
```

	eventName	distance
1	"IEM Katowice 2019"	258602.30647730103
2	"2019 League of Legends World Championship"	518022.6301569513

Zapytanie 4 odnosi się do eventów odbywających się w odległości 100km od Warszawy.

Wady i zalety

Zalety:

- obsługa danych kartezjańskich lub typu lat/long, obsługa danych 2D i 3D

- możliwość wygodnego przechowywania informacji przestrzennych, dostęp do wbudowanych funkcji przestrzennych

- wydajność zapytań przestrzennych

Wady:

- modele zawierające dane przestrzenne mogą być zbyt złożone, nieczytelne

- w przypadku dużych ilości danych przestrzennych, wydajność zapytań może spaść

- ograniczona ilość funkcji przestrzennych

Wydajność

Zapytanie	Czas
1	20 ms
2	3 ms
3	24 ms
4	20 ms

Powyżej zamieszczono tabelę z czasem wykonania każdego z użytych zapytań przestrzennych. Jak widać, zapytania te są bardzo wydajne.

Procedury

Zaimplementowano 3 procedury w Javie (jak jest polecone w dokumentacji <https://neo4j.com/docs/java-reference/current/extending-neo4j/procedures/>).

```
public class MyProcedures {  
  
    @Context  
    public Transaction tx;  
  
    @Procedure(name = "com.example.aggregateAwardsByYear", mode = Mode.READ)  
    @Description("Aggregate awards by year")  
    public Stream<YearAwardCount> aggregateAwardsByYear() {  
        String query = "MATCH (a:Award) RETURN a.year AS year, COUNT(a) AS awardCount ORDER BY year";  
        Result result = tx.execute(query);  
        return result.stream().map(row -> new YearAwardCount((long) row.get("year"), (long) row.get("awardCount")));  
    }  
  
    @Procedure(name = "com.example.listAwardsForGame", mode = Mode.READ)  
    @Description("List all awards for a specific game")  
    public Stream<AwardResult> listAwardsForGame(@Name("gameName") String gameName) {  
        String query = "MATCH (g:Game { name: $gameName })-[:NOMINATED_FOR|WON]->(a:Award) " +  
            "RETURN a.category AS category, a.year AS year, a.votedBy AS votedBy ORDER BY year";  
        Result result = tx.execute(query, Map.of("gameName", gameName));  
        return result.stream().map(row -> new AwardResult((String) row.get("category"), (long) row.get("year"), (String) row.get("votedBy")));  
    }  
  
    @Procedure(name = "com.example.aggregateAwardsByCategory", mode = Mode.READ)  
    @Description("Aggregate awards by category")  
    public Stream<CategoryAwardCount> aggregateAwardsByCategory() {  
        String query = "MATCH (a:Award) RETURN a.category AS category, COUNT(a) AS awardCount ORDER BY awardCount DESC";  
        Result result = tx.execute(query);  
        return result.stream().map(row -> new CategoryAwardCount((String) row.get("category"), (long) row.get("awardCount")));  
    }  
}
```

```
public static class YearAwardCount {
    public long year;
    public long awardCount;

    public YearAwardCount(long year, long awardCount) {
        this.year = year;
        this.awardCount = awardCount;
    }
}

public static class AwardResult {
    public String category;
    public long year;
    public String votedBy;

    public AwardResult(String category, long year, String votedBy) {
        this.category = category;
        this.year = year;
        this.votedBy = votedBy;
    }
}

public static class CategoryAwardCount {
    public String category;
    public long awardCount;

    public CategoryAwardCount(String category, long awardCount) {
        this.category = category;
        this.awardCount = awardCount;
    }
}
```

Zbudowano projekt i umieszczono plik .jar w folderze /plugins Neo4j.

Użyto komendy SHOW PROCEDURES do sprawdzenia czy się załączyły:

neo4j\$ show procedures

	name	description
3	"cdc.query"	"Query changes happened from the provided change identifier."
4	"com.example.aggregateAwardsByCategory"	"Aggregate awards by category"
5	"com.example.aggregateAwardsByYear"	"Aggregate awards by year"
6	"com.example.listAwardsForGame"	"List all awards for a specific game"

Procedura 1:

Pierwsza procedura, `aggregateAwardsByCategory`, ma na celu policzenie i zwrócenie ilości nagród per kategoria.

```
neo4j$ call com.example.aggregateAwardsByCategory() yield category, awardCount return category, awardCount;
```

	category	awardCount
1	"Best Independent Game"	6
2	"Game of the Year"	6
3	"Best Narrative"	6
4	"Games for Impact"	5
5	"Best Fighting Game"	5
6	"Best Sports/Racing Game"	5

Procedura 2:

```
1 call com.example.aggregateAwardsByYear()  
2 yield year, awardCount  
3 return year, awardCount
```

	year	awardCount
3	2016	13
4	2017	22
5	2018	25
6	2019	27
7	2020	1
8	2021	1

Procedura 2, `aggregateAwardsByYear`, zwraca ilość nagród per rok.

Procedura 3:

Procedura 3 pobiera argument – nazwę gry – i zwraca listę nagród które ona zdobyła, wraz z rokiem ich zdobycia.

```
neo4j$ call com.example.listAwardsForGame('The Witcher 3: Wild Hunt')
```

	category	year	votedBy
3	"Best Art Direction"	2015	"jury"
4	"Best Role Playing Game"	2015	"jury"
5	"Best Score/Soundtrack"	2015	"jury"
6	"Game of the Year"	2015	"jury"
7	"Best Narrative"	2015	"jury"
8	"Best Role Playing Game"	2015	"jury"

Started streaming 9 records after 4 ms and completed after 201 ms.

Omów w jaki sposób procedury w Neo4j różnią się w stosunku do relacyjnych baz danych oraz czego możemy je zastosować.

Procedury w Neo4j są implementowane w języku Java. Zbudowany plik JAR jest wrzucany do katalogu plugins. W bazach relacyjnych, możliwe jest pisanie procedur w np. PL/SQL a następnie zapisanie ich poprzez deklarację, najczęściej poprzez narzędzie wykorzystywane też do wszystkich innych operacji (np. pgAdmin), bez potrzeby ich budowania i przenoszenia.

Procedury w Neo4j są wykorzystywane do np. rozszerzania funkcjonalności o te niedostępne w języku Cypher, czy np. do przetwarzania danych w trybie wsadowym lub integracji z zewnętrznymi systemami.

Analiza końcowego zbioru danych

Mosty to krawędzie (relacje) których usunięcie spowodowałoby rozdzielenie grafu na kilka części. Węzły przegubowe to node'y / węzły których usunięcie podzieliłoby graf na kilka części. Po przeprowadzeniu analizy końcowego zbioru danych, nie zauważyłam obecności mostów. Istnieje za to węzeł przegubowy – Award. Award posiada krawędzie z Game (który posiada relacje z Developer, Game, Genre, PlayerStats i ReviewStats – węzły te mają relacje tylko z Game) oraz posiada relacje z Event, Person i EsportsTeam, które mają relacje tylko między sobą.

Zatem, węzeł Award dzieli graf na 2 podgrafy. Jeden z podgrafów skupia się wokół węzła Game, a drugi składa się z Event, Person i EsportsTeam. Jedyną częścią wspólną tych dwóch podgrafów jest węzeł Award – ponieważ zarówno wydarzenia, osoby, zespoły, jak i gry mogą być nominowane lub wygrać nagrodę.

Aby umieścić zbiór danych na kilku maszynach, można np. rozdzielić graf na dwa podgrafy opisane powyżej. W przypadku bardzo dużej zbioru danych, zwiększyłoby to wydajność zapytań i operacji na danych.

Biblioteka APOC

Biblioteka APOC oferuje szereg procedur i funkcji rozszerzających funkcjonalność języka Cypher w Neo4j. Poniżej zawarto kilka przykładów ciekawych możliwości biblioteki APOC:

Apoc.any.properties

Funkcja `apoc.any.properties` umożliwia podejrzanie właściwości i ich wartości dla danego węzła. Składnia jest zwięzła i wygodna w użyciu.

```
1 MATCH (p:Person { firstName: 'Eefje' })
2 return apoc.any.properties(p)
```

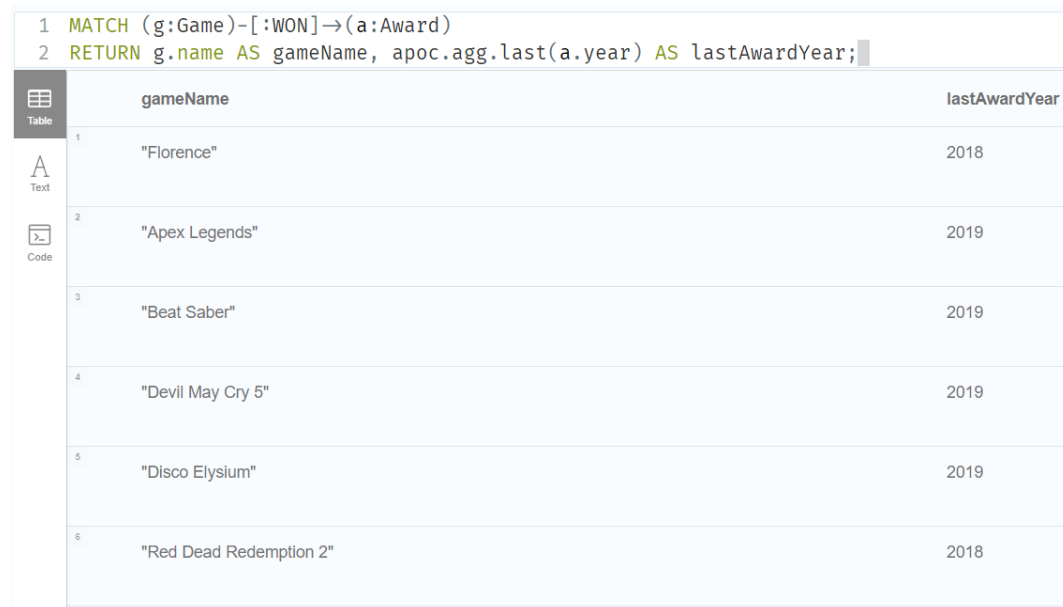


apoc.any.properties(p)	
1	{ "lastName": "Depoortere", "nickname": "Sjokz", "profession": "Event host", "firstName": "Eefje" }

Apoc.agg.last

Procedura `apoc.agg.last` umożliwia podejrzanie ostatniej wartości z danej kolekcji. Jest to wygodne np. przy sprawdzaniu ostatniej wstawionej wartości, lub np. przy szukaniu wydarzenia najbliższego czasowo.

```
1 MATCH (g:Game)-[:WON]-(a:Award)
2 RETURN g.name AS gameName, apoc.agg.last(a.year) AS lastAwardYear;
```



	gameName	lastAwardYear
1	"Florence"	2018
2	"Apex Legends"	2019
3	"Beat Saber"	2019
4	"Devil May Cry 5"	2019
5	"Disco Elysium"	2019
6	"Red Dead Redemption 2"	2018

Apoc.date.convertFormat

Jest to procedura umożliwiająca konwersję daty z jednego formatu (w formie String) na drugi. Jest to bardzo wygodne narzędzie do konwersji dużej ilości dat.

```
1 MATCH (g:Game)
2 WITH g, apoc.date.convertFormat(g.release, 'yyyy-MM-dd', 'dd-MM-yyyy') AS formattedDate
3 RETURN g.name AS gameName, formattedDate;
```

	gameName	formattedDate
1	"Panda City"	"08-09-2021"
2	"Neeron: The Blade of Nature"	"22-09-2021"
3	"Hero Park"	"28-05-2021"
4	"Astria Ascending"	"30-09-2021"
5	"We're All Going To Die"	"19-02-2021"
6	"Ragnarok Chess"	"04-05-2021"

Apoc.export.cypher.all

Procedura apoc.export.cypher.all umożliwia na eksport danych z bazy danych do pliku z skryptem w języku Cypher. Jest to bardzo przydatne jako zabezpieczenie przed utratą danych lub jako narzędzie do przenoszenia danych na inną maszynę.

```
1 CALL apoc.export.cypher.all('all.cypher', {})
2 YIELD file, nodes, relationships, properties, time
3 RETURN file, nodes, relationships, properties, time;
```

	file	nodes	relationships	properties	time
1	"all.cypher"	215654	531926	650361	26233

Apoc.neighbors.athop

Procedura apoc.neighbors.athop zwraca wszystkie node'y połączone przez daną relację. Jest to bardzo zwięzłe i wygodne narzędzie do sprawdzenia / podejrzenia danych.

```
1 MATCH (g:Game { link: '/app/1593180/' })
2 CALL apoc.neighbors.athop(g, 'DEVELOPED_BY|PUBLISHED_BY', 1) YIELD node
3 RETURN node;
```



Graph



Table



Text



Code

node

1

```
{
  "identity": 54198,
  "labels": [
    "Publisher"
  ],
  "properties": {
    "name": "Minimol Games"
  },
  "elementId": "4:0097473b-7354-4bcb-9ce6-7cc0e87c70e8:54198"
}
```

2

```
{
  "identity": 89910,
  "labels": [
    "Developer"
  ],
```