[ZBD] Week 3 - PostgreSQL queries

Zacharczuk Jakub

jz418488

1 Zapytanie o pary ze wspólnym podmiotem [NEO4J]

1.1 Zapytanie z treści zadania

```
MATCH (a:Officer)-->(e:Entity)<--(b:Officer)
WITH a, b, COUNT(*) AS cnt
ORDER BY cnt DESC
RETURN a, b, cnt
LIMIT 10
```

Powyższe zapytanie działa bardzo długo (kilkadziesiąt sekund), po czym generuje błąd: "Neo.DatabaseError.General.UnknownError" z informacją "Java heap space". Z tego powodu chcemy ograniczyć powyższe zapytanie.

1.2 Nowe zapytanie

```
MATCH (a:Officer)-->(e:Entity)<--(b:Officer)
WHERE a.name STARTS WITH "A"
WITH a, b, COUNT(*) AS cnt
ORDER BY cnt DESC
RETURN a, b, cnt
LIMIT 10
```

1.3 Analiza wydajności nowego zapytania

Returned 10 rows in 830 ms.
Returned 10 rows in 485 ms.
Returned 10 rows in 428 ms.
Returned 10 rows in 426 ms.
Returned 10 rows in 422 ms.
Returned 10 rows in 446 ms.

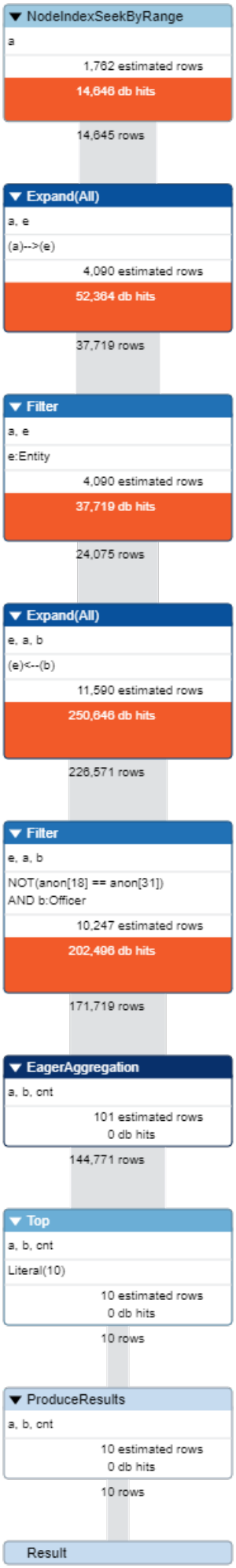
Wobec powyższych rezultatów można zauważyć, że pierwsze zapytanie wykonuje się znacznie dłużej niż kolejne. Najprawdopodobniej wynika to z faktu, że kolejne zapytania wykorzystują scachowany plan zapytania oraz pewne wiersze. Czasy po "rozgrzaniu maszynki" oscylują wokół 420ms.

1.4 Plan zapytania

Po wywołaniu poniższego zapytania:

```
PROFILE
MATCH (a:Officer)-->(e:Entity)<--(b:Officer)
WHERE a.name STARTS WITH "A"
WITH a, b, COUNT(*) AS cnt
ORDER BY cnt DESC
RETURN a, b, cnt
LIMIT 10
```

Otrzymujemy:


NodeIndexSeekByRange
a
1,762 estimated rows
14,646 db hits
14,645 rows
Expand(All)
a, e
(a)-->(e)
4,090 estimated rows
52,364 db hits
37,719 rows
Filter
a, e
e:Entity
4,090 estimated rows
37,719 db hits
24,075 rows
Expand(All)
e, a, b
(e)<--(b)
11,590 estimated rows
250,646 db hits
226,571 rows
Filter
e, a, b
NOT(anon[18] == anon[31])
AND b:Officer
10,247 estimated rows
202,496 db hits
171,719 rows
EagerAggregation
a, b, cnt
101 estimated rows
0 db hits
144,771 rows
Top
a, b, cnt
Literal(10)
10 estimated rows
0 db hits
10 rows
ProduceResults
a, b, cnt
10 estimated rows
0 db hits
10 rows
Result

2 Zapytanie o pary ze wspólnym podmiotem [PostgreSQL]

2.1 Założenia:

W rozwiązaniu zdecydowałem się dodać poniższe założenia

- nie interesują nas takie pary Officers,

  ```
  gdzie a->e<-a
  ```

- chcemy pozbyć się duplikatów tzn.

  ```
  a->e<-b to to samo co b->e<-a
  ```

- nie musimy się ograniczać do Officers a zaczynających się na 'A'

2.2 Zapytanie:

2.2.1 Wybieramy tylko krawędzie wskazujące na Entity

```
SELECT start_id, end_id, name FROM edge
INNER JOIN entity ON entity.node_id = end_id
LIMIT 10;
```

2.2.2 Zawężamy do wierszy gdzie a jest typu Officer

```
SELECT start_id, officer.name, end_id, entity.name FROM edge
INNER JOIN entity ON entity.node_id = end_id
INNER JOIN officer ON officer.node_id = start_id
LIMIT 10;
```

2.2.3 Tworzymy krotki typu (a:Officer)->(e:Entity)<-(b:Officer)

```
SELECT e1.start_id, of1.name, e1.end_id, entity.name, e2.start_id FROM edge e1
INNER JOIN entity ON entity.node_id = e1.end_id
INNER JOIN officer of1 ON of1.node_id = start_id
INNER JOIN edge e2 ON e1.end_id = e2.end_id
INNER JOIN officer of2 ON of2.node_id = e2.start_id
LIMIT 10;
```

2.2.4 Grupujemy krotki po parach a, b i zliczamy liczbę wspólnych e (a:Officer)->(e:Entity)<-(b:Officer)

```
SELECT e1.start_id AS a_id, e2.start_id AS b_id, COUNT(entity.node_id) AS common_entities
FROM edge e1
INNER JOIN entity ON entity.node_id = e1.end_id
INNER JOIN officer of1 ON of1.node_id = start_id
INNER JOIN edge e2 ON e1.end_id = e2.end_id
INNER JOIN officer of2 ON of2.node_id = e2.start_id
GROUP BY a_id, b_id
ORDER BY common_entities DESC
LIMIT 10;
```

2.2.5 Aby uniknąć krotek opisanych w założeniach łączymy tylko takie a i b, gdzie a.id < b.id

```
...
INNER JOIN officer of2 ON of2.node_id = e2.start_id AND e2.start_id > e1.start_id
...
```

2.3 Ostateczna wersja zapytania

```
 EXPLAIN ANALYZE
 SELECT e1.start_id AS a_id, e2.start_id AS b_id, COUNT(entity.node_id) AS common_entities
 FROM edge e1
    INNER JOIN entity ON entity.node_id = e1.end_id
    INNER JOIN officer of1 ON of1.node_id = start_id
    INNER JOIN edge e2 ON e1.end_id = e2.end_id
    INNER JOIN officer of2 ON of2.node_id = e2.start_id AND e2.start_id > e1.start_id
GROUP BY a_id, b_id
ORDER BY common_entities DESC
LIMIT 10;
```

2.4 Wynik zapytania:

```
    a_id    |   b_id   | common_entities
----------+----------+-----------------
 12215501 | 12215506 |             817
 12211989 | 12217554 |             540
 12205089 | 12205090 |             516
 12138721 | 12215506 |             473
 12132169 | 12175775 |             437
 12118873 | 12140444 |             432
 12130995 | 12197455 |             416
 12096558 | 12140444 |             396
 12204607 | 12204608 |             329
 12215503 | 12215506 |             305
```

2.5 Wydajność

Aby zmierzyć wydajność wpisałem timing ON. W wyniku otrzymałem poniższe czasy:
Time: 1185.487 ms (00:01.185)
Time: 1218.000 ms (00:01.218)
Time: 1158.426 ms (00:01.158)
Time: 1196.862 ms (00:01.197)
Time: 1166.028 ms (00:01.166)
Wobec powyższych danych nie można stwierdzić że wygenerowanie planu zapytania skróciło czas działania. Być może restart kontenera i restart połączenia z bazą nie wystarczył do wyczyszczenia cache'a.

2.6 Plan zapytania:

Plan zapytania wygenerowałem dodając prefix EXPLAIN ANALYZE.

```
                                    QUERY PLAN
-------------------------------------------------------------------------------------
 Limit  (cost=196869.56..196869.58 rows=10 width=16) (actual time=1525.732..1533.679
 rows=10 loops=1)
   ->  Sort  (cost=196869.56..198642.11 rows=709022 width=16) (actual
   time=1511.888..1519.833 rows=10 loops=1)
         Sort Key: (count(entity.node_id)) DESC
         Sort Method: top-N heapsort  Memory: 25kB
         ->  Finalize GroupAggregate  (cost=95918.74..181547.85 rows=709022 width=16)
         (actual time=639.999..1379.559 rows=1638334 loops=1)
               Group Key: e1.start_id, e2.start_id
               ->  Gather Merge  (cost=95918.74..170026.24 rows=590852 width=16)
               (actual time=639.989..1064.875 rows=1647477 loops=1)
                     Workers Planned: 2
                     Workers Launched: 2
                     ->  Partial GroupAggregate  (cost=94918.71..100827.23
                     rows=295426 width=16) (actual time=606.875..805.361 rows=549159
                     loops=3)
```

```
Group Key: e1.start_id, e2.start_id
->  Sort  (cost=94918.71..95657.28 rows=295426 width=12)
(actual time=606.858..661.858 rows=661267 loops=3)
      Sort Key: e1.start_id, e2.start_id
      Sort Method: external merge  Disk: 15208kB
      Worker 0:  Sort Method: external merge  Disk:
      12456kB
      Worker 1:  Sort Method: external merge  Disk:
      15208kB
      ->  Parallel Hash Join  (cost=38756.77..63025.16
      rows=295426 width=12) (actual time=254.795..409.997
      rows=661267 loops=3)
            Hash Cond: (e1.end_id = entity.node_id)
            Join Filter: (e2.start_id > e1.start_id)
            Rows Removed by Join Filter: 785207
            ->  Parallel Hash Join
            (cost=5721.02..20752.09 rows=280876 width=8)
            (actual time=29.584..73.810 rows=151719
            loops=3)
                  Hash Cond: (e1.start_id = of1.node_id)
                  ->  Parallel Seq Scan on edge e1
                  (cost=0.00..14293.76 rows=280876
                  width=8) (actual time=0.014..17.942
                  rows=224701 loops=3)
                  ->  Parallel Hash
                  (cost=4479.34..4479.34 rows=99334
                  width=4) (actual time=18.150..18.150
                  rows=79467 loops=3)
                        Buckets: 262144  Batches: 1
                        Memory Usage: 11424kB
                        ->  Parallel Seq Scan on officer
                        of1  (cost=0.00..4479.34
                        rows=99334 width=4) (actual
                        time=0.010..7.072 rows=79467
                        loops=3)
            ->  Parallel Hash  (cost=28152.80..28152.80
            rows=280876 width=12) (actual
            time=161.610..161.612 rows=103119 loops=3)
                  Buckets: 131072  Batches: 16  Memory
                  Usage: 2016kB
                  ->  Parallel Hash Join
                  (cost=12384.42..28152.80 rows=280876
                  width=12) (actual time=69.748..145.226
                  rows=103119 loops=3)
                        Hash Cond: (e2.start_id =
                        of2.node_id)
                        ->  Parallel Hash Join
                        (cost=6663.41..21694.48
                        rows=280876 width=12) (actual
                        time=16.391..90.740 rows=174330
                        loops=3)
                              Hash Cond: (e2.end_id =
                              entity.node_id)
                              ->  Parallel Seq Scan on
                              edge e2
                              (cost=0.00..14293.76
                              rows=280876 width=8) (actual
                              time=0.031..20.594
                              rows=224701 loops=3)
                              ->  Parallel Hash
                              (cost=5550.73..5550.73
```

```
                                                rows=89014 width=4) (actual
                                                time=15.313..15.313
                                                rows=71211 loops=3)
                                                      Buckets: 262144
                                                      Batches: 1  Memory
                                                      Usage: 10464kB
                                                      ->  Parallel Index
                                                      Only Scan using
                                                      entity_pkey on entity
                                                      (cost=0.42..5550.73
                                                      rows=89014 width=4)
                                                      (actual
                                                      time=0.023..5.524
                                                      rows=71211 loops=3)
                                                            Heap Fetches: 0
                                          ->  Parallel Hash
                                          (cost=4479.34..4479.34 rows=99334
                                          width=4) (actual
                                          time=25.412..25.413 rows=79467
                                          loops=3)
                                                Buckets: 262144  Batches: 1
                                                Memory Usage: 11424kB

                                                ->  Parallel Seq Scan on
                                                officer of2
                                                (cost=0.00..4479.34
                                                rows=99334 width=4) (actual
                                                time=8.549..15.224
                                                rows=79467 loops=3)
 Planning Time: 0.837 ms
 JIT:
   Functions: 121
   Options: Inlining false, Optimization false, Expressions true, Deforming true
   Timing: Generation 6.159 ms, Inlining 0.000 ms, Optimization 1.556 ms, Emission
   37.315 ms, Total 45.030 ms
 Execution Time: 1537.354 ms
```

3 Najkrótsza ścieżka [NEO4J]

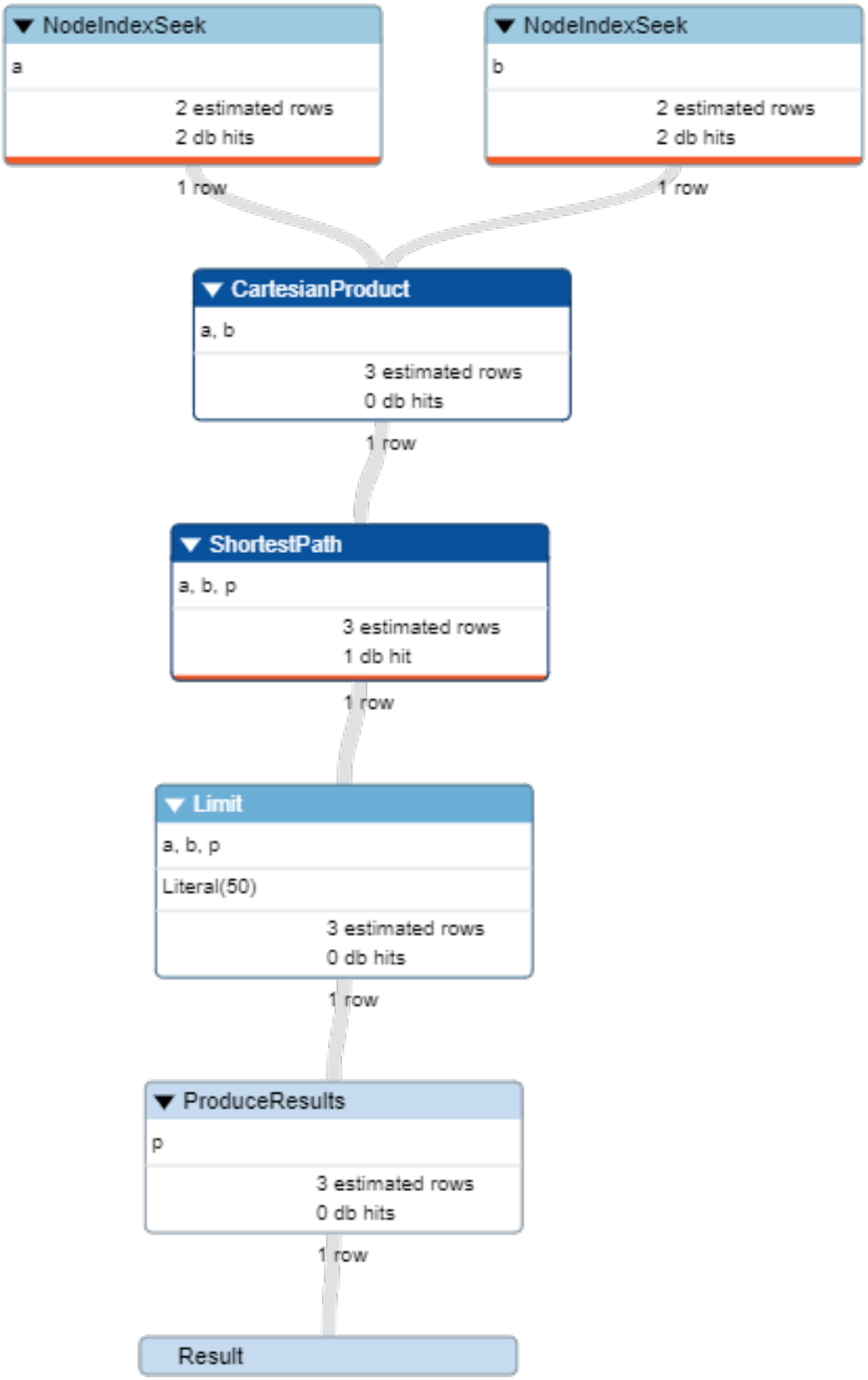
3.1 Zapytanie

```
MATCH (a:Officer {name: "Emma Watson"})
MATCH (b:Officer {name: "APPLETON INVEST CAPITAL LIMITED"})
MATCH p=shortestPath((a)-[*]-(b))
RETURN p
LIMIT 10
```

3.2 Wydajność:

Returned 2 rows in 75 ms.
Returned 2 rows in 20 ms.
Returned 2 rows in 22 ms.
Returned 2 rows in 30 ms.
Returned 2 rows in 24 ms.
Ponownie widać, że kolejne wywołania tego samego zapytania działają znacznie szybciej.

3.3 Plan zapytania:



4 Najkrótsza ścieżka [PostgreSQL]

4.1 Zapytanie:

id: 12126782 - Emma Watson
id: 12132897 - APPLETON INVEST CAPITAL LIMITED

```
    WITH RECURSIVE generate_path (id, depth, visited) AS (
        SELECT start_id AS id,
                0 AS depth,
                ARRAY[start_id] AS visited
        FROM edge
        WHERE start_id = 12126782
      UNION ALL (
          (WITH generate_path(id, depth, visited) AS (TABLE generate_path)
        SELECT e.end_id, depth + 1, visited || e.end_id
        FROM generate_path gp, edge e
        WHERE gp.id = e.start_id AND NOT e.end_id = ANY(visited)
            AND depth < 7
        UNION ALL
          SELECT e.start_id, depth + 1, visited || e.start_id
        FROM generate_path gp, edge e
        WHERE gp.id = e.end_id AND NOT e.start_id = ANY(visited)
            AND depth < 7
      ))
)
SELECT visited
FROM generate_path
WHERE id = 12132897
LIMIT 1;
```

4.2 Wynik

```
                    visited
-----------------------------------------------
 {12126782,10152535,11011539,10155950,12132897}
 Time: 456.378 ms
```

4.3 Plan zapytania (wygenerowany jak wcześniej)

```
                                QUERY PLAN
-------------------------------------------------------------------------------
 Limit  (cost=431484.07..431488.56 rows=1 width=32) (actual time=663.135..663.193
 rows=1 loops=1)
   CTE generate_path
     ->  Recursive Union  (cost=1000.00..431484.07 rows=5380 width=40) (actual
     time=32.934..644.860 rows=5095 loops=1)
           ->  Gather  (cost=1000.00..15996.95 rows=10 width=40) (actual
           time=32.932..33.090 rows=2 loops=1)
                 Workers Planned: 2
                 Workers Launched: 2
                 ->  Parallel Seq Scan on edge  (cost=0.00..14995.95 rows=4 width=40) (actual
                 time=15.933..16.021 rows=1 loops=3)
                       Filter: (start_id = 12126782)
                       Rows Removed by Filter: 224700
           ->  Append  (cost=4.66..41537.95 rows=537 width=40) (actual time=79.179..152.762
           rows=1273
           loops=4)
                 CTE generate_path
                   ->  WorkTable Scan on generate_path generate_path_1  (cost=0.00..2.00
```

```
                  rows=100 width=40) (actual time=0.000..0.049 rows=1272 loops=4)
                ->  Hash Join  (cost=2.66..20766.99 rows=379 width=40) (actual
                time=70.864..84.878 rows=1270 loops=4)
                      Hash Cond: (e.start_id = gp.id)
                      Join Filter: (e.end_id <> ALL (gp.visited))
                      Rows Removed by Join Filter: 2
                      ->  Seq Scan on edge e  (cost=0.00..18226.02 rows=674102 width=8)
                      (actual time=0.002..41.239 rows=674102 loops=4)
                      ->  Hash  (cost=2.25..2.25 rows=33 width=40) (actual time=0.377..0.377
                      rows=1272 loops=4)
                            Buckets: 8192 (originally 1024)  Batches: 1 (originally 1)
                            Memory Usage: 461kB
                            ->  CTE Scan on generate_path gp  (cost=0.00..2.25 rows=33
                            width=40) (actual time=0.003..0.246 rows=1272 loops=4)
                                  Filter: (depth < 7)
                ->  Hash Join  (cost=2.66..20760.91 rows=158 width=40) (actual
                time=53.464..67.817 rows=3 loops=4)
                      Hash Cond: (e_1.end_id = gp_1.id)
                      Join Filter: (e_1.start_id <> ALL (gp_1.visited))
                      Rows Removed by Join Filter: 1
                      ->  Seq Scan on edge e_1  (cost=0.00..18226.02 rows=674102 width=8)
                      (actual time=0.002..31.782 rows=505622 loops=4)
                      ->  Hash  (cost=2.25..2.25 rows=33 width=40) (actual time=0.279..0.279
                      rows=1272 loops=4)
                            Buckets: 8192 (originally 1024)  Batches: 1 (originally 1)
                            Memory Usage: 461kB
                            ->  CTE Scan on generate_path gp_1  (cost=0.00..2.25 rows=33
                            width=40) (actual time=0.002..0.149 rows=1272 loops=4)
                                  Filter: (depth < 7)
   ->  CTE Scan on generate_path  (cost=0.00..121.05 rows=27 width=32) (actual
   time=645.934..645.934 rows=1 loops=1)
         Filter: (id = 12132897)
         Rows Removed by Filter: 5094
 Planning Time: 0.184 ms
 JIT:
   Functions: 50
   Options: Inlining false, Optimization false, Expressions true, Deforming true
   Timing: Generation 3.228 ms, Inlining 0.000 ms, Optimization 1.124 ms, Emission 20.760 ms,
   Total 25.112 ms
 Execution Time: 666.105 ms
(39 rows)

Time: 667.072 ms
```

4.4 Analiza

Najprawdopodobniej z powodu takiego jak w punkcie 2, kolejne wywołania nie zmieniają czasu działania zapytania. Dodatkowo w zapytaniu zastosowałem sztywny limit na depth < 7, gdyż zwiększenie tej wartości powoduje znaczne wydłużenie działania zapytania (tzn. ponad minutę).

5 Wnioski

Bazę danych należy wybrać pod kątem wykonywanych na niej operacji. W operacjach typu 1 znacznie lepiej spisał się Postgres, natomiast w zapytaniach grafowych znaczną przewagę ma dedykowany takim zapytaniom Neo4j.