Универзитет „Св. Кирил и Методиј“ во Скопје
ФАКУЛТЕТ ЗА ИНФОРМАТИЧКИ НАУКИ И КОМПЈУТЕРСКО ИНЖЕНЕРСТВО

Проектна задача

Предмет:

Неструктурирани бази на податоци

Тема:

Складирање на податоци во граф базирани бази на податоци и споредба

Тим:
Мартин Трајковски 195063
Вангел Трајковски 181261

Јуни 2023

Содржина

Апстракт

Се запознавме со два системи за складирање на податоци, при следење на документацијата, дознавме како се приспособуваат за работа, како се вметнуваат податоците и како да пристапиме до нив, со развивање на сценарија дојдовме до реален пример како би се искористило некое податочно множество за пристап до некои податоци во системот, со споредување на перформанси дознаваме кој од двата системи е побрз, поефикасен за користење и како би можеле да го оптимизираме.

1. Вовед

При изработка на апликации често се случува да се потребни податочни множества кои содржат ставки кои се меѓусебно поврзани, и е потребно да поминуваме низ овие релации, ова го овозможуваат граф базите на податоци, кои се состојат од точки кои содржат атрибути и ребра кои претставуваат релации, односно кои било две точки моќе да бидат поврзани со некоја релација, при што се губи таа класична структура на податоците, за цел на изработка на овој проект беше споредбата помеѓу два системи во кои се складираат податоци во вид на граф, при што се запознаваме со овој вид на системи и со користење на документација можеме да доведеме до некаков заклучок.

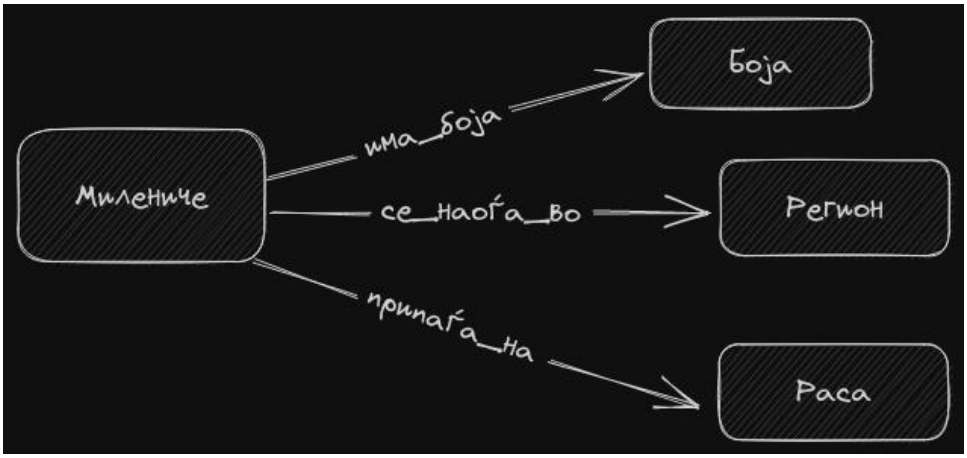
2. Методологија

2.1. Податоци

Податоците кои користени од наша страна се множество од миленичиња[1], ова множество е во формат „CSV“, и беше потребна преработка и читање од страна на користените системи за чување на податоци, во прилог е претставена слика, која ги отсликува податоците.

Како што можиме да видиме од следната слика, множеството од податоци се состои од еден главен тип на ставка која содржи податоци за миленичињата, и дополнително имаме уште 3 споредни типови кои го опишуваат миленичето, како што се бојата, која според податоците може да се содржи 3 пати, потоа ја имаме расата на миленичето, за која секое миленичe имам можност да поседува 2 референци доколку е мешано од повеќе раси, во расата се состои и типот на миленичe односно, дали миленичето е куче, маче итн, последниот податок е за каде е лоцирано, односно во кој регион од соодветната држава се наоѓа миленичето.

По претставување на овие податоци следува и нивно претставување во вид на граф, како што можи да се види од следната слика, каде се отсликани визуелно форматот на податоците при вметнување во базата на податоци, базата се состои од 3 релации, и во сите случаеви релациите се движат од ентитетот „миленичe“ до останатите 3 ентитети.



2.2. Агрегација

Во рамки на граф базите на податоци, податоците се агрегираат така што се почнува од една точка во графот и за неа се пребарува преку врските кои ги има, доколку базата се состои од елементи кои се меѓусебно поврзани со ребра, податоците се агрегираат во една комплетна низа од истите и се испраќаат до корисниците на системот.

Податоците се приспособени за да се вклопат во рамки на системот, при што беа искористени 2 системи, има 2 различни начини на приспособување:

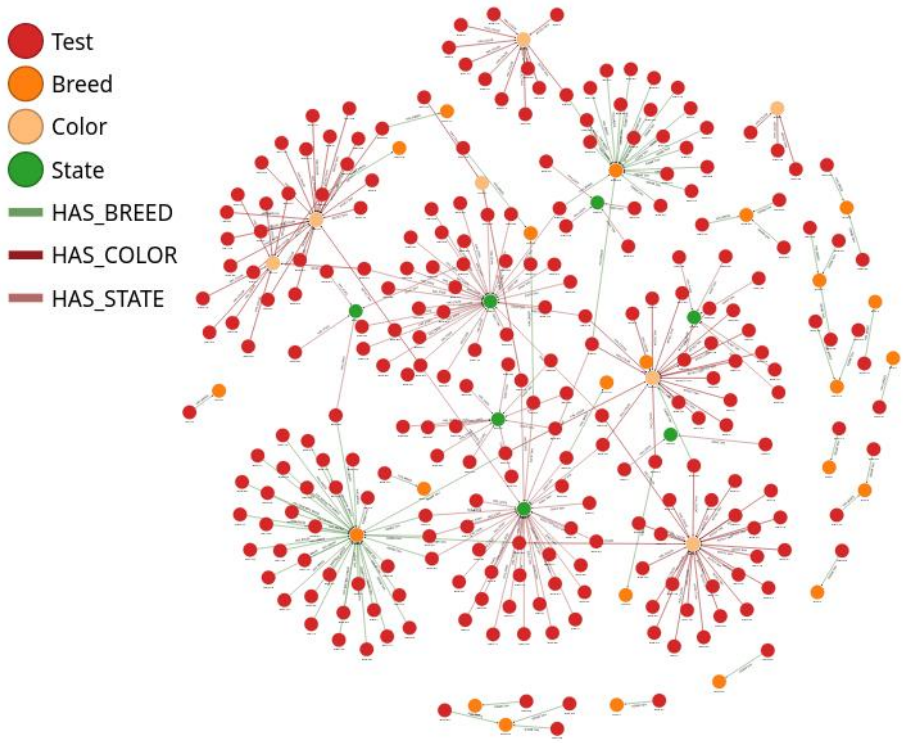
Neo4j:

За да се приспособат податоците и да се вметнат во овој систем, искористени се прашалници преку кои се читаат податоците од CSV формат, кои подоцна се креираат точките и ребрата на графот. Еeден од проблемите на овој систем беа бавното извршување на прашалниците за вметнување на податоците.

OrientDB:

Во вториот систем кој се одлучивме да го користиме, нема поддршка преку прашалници да се прочита директно од CSV датотека, но постои можност за вметнување на податоци со една алатка така наречена ETL (Extractor, Transformer, Loader), оваа алатка работи со конфигурација во JSON формат, преку која е потребно да се кажи на системот како да ги извади податоците (Extract), на кој начин да ги промени и адаптира на системот (Transformer), и како да ги впиши.

Агрегацијата е извршена така што од податоците дадени во повеќе табели се претставени во граф, тоа можи да се види на претходната слика, во која податоците немаат веќе колони, туку имаат атрибути и ребра кои ги поврзуваат ентитетите, тоа можи да се види на следната слика.



2.3. Сценарија

Според даденото податочно множество, издвоени се следните сценарија:

- Нов корисник, кој за прв пат го користи системот, навигира низ системот и стигнува до страница која му нуди можност за пребарување на миленичиња, и на корисникот му се понудени многу различни критериуми за пребарување и при селекција на еден критериум на за тој корисник се ажурираат останатите критериуми за пребарување со можните комбинации, и ги пребарува добиените миленичиња според одбраниот критериум.

- Аналитичар, сака да добие информации од системот преку кои сака да објасни своја теза, при што прво ги групира тие податоци по одредени критериуми и добива агрегирани податочни множества преку различни критериуми (регион, боја, итн.)

2.4. Прашалници

- Едноставни:
  - Пребарување на сите миленичиња

    OrientDB:

    Neo4j:

- Пребарување на миленичиња со пагинација

  OrientDB:

```
select from Test skip 100 limit 50;
```

  Neo4j:

```
match (test: Test) return test skip 100 limit 50;
```

- Соединување меѓу ентитетите за миленичиња, раса и боја

  OrientDB:

```
select *, outE('HAS_BREED').in.Name as Breed, outE('HAS_COLOR').in.Name as Color from Test;
```

  Neo4j

```
match (color: Color)<-[r_color:COLOR]-(test: Test)
with color, test, r_color
match (test)-[r_breed: BREED]-(breed: Breed)
return breed, test, color;
```

- Сложени:
  - Пронаоѓање на сите можни раси врз основа на одбраните критериуми

    OrientDB:

    Neo4j:

  - Пронаоѓање на сите миленичиња кои ги задоволуваат сите избрани критериуми со пагинација

    OrientDB:

    Neo4j:

- Агрегирани:
  - Пронаоѓање на сите миленичиња со одредена боја и колкав е нивниот број

    OrientDB:

```
match { class: Test, as: pet }-HAS_BREED→{class: Breed, as: breed, where: (Name = 'Husky')},
      { class: Color, as: color }←HAS_COLOR-{ class: Test, as: pet }
return distinct color.ID as ColorID, color.Name as ColorName, count(pet) as Huskies
group by color.Name, Color.ID
order by Huskies desc
```

    Neo4j:

```
match (pet: Test)-[r_breed:BREED]→(breed: Breed),
      (color: Color)←[r_color:COLOR]-(pet)
where breed.name = 'Husky'
return color.id as ColorID, color.name as colorName, count(pet) as huskies
order by huskies desc;
```

  - Пронаоѓање на бројот на сите миленичиња од одредена раса во регионите

    OrientDB:

```
match { class: Test, as: pet }-HAS_BREED→{class: Breed, as: breed, where: (Name = 'Husky')},
      { class: State, as: state }←HAS_STATE-{ class: Test, as: pet }
return state.Name as stateName, state.ID as stateId, count(pet) as petCount
group by state.Name, state.ID
order by petCount desc
```

    Neo4j:

```
match (pet: Test)-[r_breed: BREED]→(breed: Breed),
      (state: State)←[r_state: STATE]-(pet)
where pet.name = 'Husky'
return distinct state.name as name, state.id as id, count(pet) as petCount
order by petCount desc;
```

3. Добиени резултати

3.1. Читање и запишување на постоечки податоци

При читање и запишување на податоците се соочивме со проблемот каде што беше потребно 904148ms за neo4j да го прочита, агрегира и запиши цело множество на податоци, доколку од другата страна беше потребно 1200ms, но за orientDB беше многу потешко да се прочитаат и запишат податоците, при што е потребно да се научи за ETL, и да се напиши конфигурација за тоа како да се прочитаат, променат и запишат, додека neo4j користеше позната синтакса, односно истиот јазик што се користи за прашалници се користи и за читање, агрегација и запишување.

3.2. Извршување на прашалници

Постои слична синтакса за пишување на прашалници во двата системи, се разликува во јазикот, при што orientDB користи класичен SQL начин за пишување на прашалници со додадени начини за пишување на прашалници кои може да се претпостави дека се директно инспирирани од јазикот кој го користи neo4j, додека orientDB поседува по стар јазик и се познава дека било потребно да се прават некои заобиколувања за јазикот наликува како тој на neo4j, при што се користи комбинација од SQL со json и стрелки во вид на „-->“ и „<--“, кои исто така се користат во neo4j.

За извршување на едноставни прашалници во orientDB без поврзување на повеќе релации при користење на select прашалник потребно беше 875ms, 7944 ставки, додека во neo4j исто така без релации и истиот број на ставки во neo4j потребни се 8ms, но доколку користиме поразлична синтакса за orientDB, односно со користење на match прашалник, добиваме споредливи резултати од 82ms.

При споредување на прашалници со додавање на релации добиваме слични разлики како и претходно, односно за orientDB се добива 261ms za 7944 ставки со користење на match прашалници, додека со користење на select прашалници е добиено време на извршување од 862ms, додека во neo4j со истите релации и истиот број на ставки времето на извршување е 14ms.

При аналитички прашалници добиваме слични перформанси, во orientDB истиот прашалник се извршува за 265ms, додека во neo4j доволни се само 7ms.

4. Заклучок

Со прикажаните резултати доаѓаме до заклучок дека neo4j базата на податоци која од почетокот била градена идеата за складирање на податоци во вид на граф, и била приспособена за да се справи со тие релации, од друга страна во orientDB е воведен граф моделот во систем кој бил направен за чување на json документи.

Од наша страна е прочитано податочното множество, со што се соочивме со различен тип на проблеми, при што во neo4j го користи истиот начин за читање и запиување што се користи во класично користење, додека во orientDB, користи специјален начин за вметнување на податоци, додека е побрзо, исто така беше комлицирано за пишување.

Креирани беа сценарија врз основа на кои се пишуваа прашалници, при што откривме кои се начините за пристап на податоците од двата системи, се запознавме со cypher јазикот кој го користи neo4j, релациите се претставени со интуитивен начин, додека orientDB продолжува да следи сличен начин на пишување на прашалници иако како јазик за пристап до податоците го користи sql.

Направена беше анализата на двата системи, при што дојдовме до податоци за перформансите на двата системи, во која анализа дојдовме до заклучок дека neo4j базата беше доста побрза за пристап на податоците.

5. Користена литература

1. https://www.kaggle.com/c/petfinder-adoption-prediction
2. https://neo4j.com/
3. https://orientdb.org/
4. https://github.com/legendmt25/nbnp