1. Opis zapytania

Wyszukaj top 5 ogłoszeń mających najwięcej aplikacji:

* Posiadających od 3 do 5 zainteresowań
* Od klienta będącego kiedyś poszukującym pracy
* Aplikacje między 2018 a 2019 rokiem
* Co najmniej dwóch kandydatów jest znajomymi, a co najmniej jeden został przyjęty

1. Zapytanie SQL:

select distinct top 5 op.idogloszenia, op.opis as 'Opis oferty', op.zarobki as 'Proponowane zarobki',

k.nazwafirmy as 'Zleceniodawca' , aplikacje.ilosc as 'Ilosc aplikacji'

from dbo.ogloszeniepracownicze op

join (select count(\*) as ilosc, ka.ogloszenie as ogloszenie from dbo.kandydat ka group by ka.ogloszenie) aplikacje on aplikacje.ogloszenie = op.idogloszenia

join dbo.klient k on op.zleceniodawca = k.idklienta

join dbo.kandydat przyjety\_kandydat on (przyjety\_kandydat.ogloszenie = op.idogloszenia and przyjety\_kandydat.zaakceptowany = 1)

join dbo.osoba przyjeta\_osoba on przyjeta\_osoba.id = przyjety\_kandydat.osoba

join dbo.znajomi znajomi\_kandydaci on znajomi\_kandydaci.znajomy1 = przyjeta\_osoba.id or znajomi\_kandydaci.znajomy2 = przyjeta\_osoba.id

join dbo.kandydat znajomy\_kandydat\_przyjetego on znajomy\_kandydat\_przyjetego.ogloszenie = op.idogloszenia and

((znajomy\_kandydat\_przyjetego.osoba = znajomi\_kandydaci.znajomy1 and znajomi\_kandydaci.znajomy1 != przyjeta\_osoba.id) or

(znajomy\_kandydat\_przyjetego.osoba = znajomi\_kandydaci.znajomy2 and znajomi\_kandydaci.znajomy2 != przyjeta\_osoba.id))

join dbo.osoba znajomy\_przyjetego on znajomy\_przyjetego.id = znajomy\_kandydat\_przyjetego.osoba

where (select count(\*) from dbo.ogloszeniezainteresowanie oz where oz.ogloszenie = op.idogloszenia) between 3 and 5

and aplikacje.ilosc = (select count(\*) from dbo.kandydat ka where ka.ogloszenie = op.idogloszenia and dataaplikacji between '2018/01/01' and '2019/12/31')

and k.dzialalnosc is not null

order by aplikacje.ilosc desc

1. Dodatkowa tabela cache SQL

create view top5\_ogloszen as

select distinct top 5 op.idogloszenia, op.opis as 'Opis oferty', op.zarobki as 'Proponowane zarobki',

k.nazwafirmy as 'Zleceniodawca', aplikacje.ilosc as 'Ilosc aplikacji'

from dbo.ogloszeniepracownicze op

join (select count(\*) as ilosc, ka.ogloszenie as ogloszenie from dbo.kandydat ka group by ka.ogloszenie) aplikacje on aplikacje.ogloszenie = op.idogloszenia

join dbo.klient k on op.zleceniodawca = k.idklienta

join dbo.kandydat przyjety\_kandydat on (przyjety\_kandydat.ogloszenie = op.idogloszenia and przyjety\_kandydat.zaakceptowany = 1)

join dbo.osoba przyjeta\_osoba on przyjeta\_osoba.id = przyjety\_kandydat.osoba

join dbo.znajomi znajomi\_kandydaci on znajomi\_kandydaci.znajomy1 = przyjeta\_osoba.id or znajomi\_kandydaci.znajomy2 = przyjeta\_osoba.id

join dbo.kandydat znajomy\_kandydat\_przyjetego on znajomy\_kandydat\_przyjetego.ogloszenie = op.idogloszenia and

((znajomy\_kandydat\_przyjetego.osoba = znajomi\_kandydaci.znajomy1 and znajomi\_kandydaci.znajomy1 != przyjeta\_osoba.id) or

(znajomy\_kandydat\_przyjetego.osoba = znajomi\_kandydaci.znajomy2 and znajomi\_kandydaci.znajomy2 != przyjeta\_osoba.id))

join dbo.osoba znajomy\_przyjetego on znajomy\_przyjetego.id = znajomy\_kandydat\_przyjetego.osoba

where (select count(\*) from dbo.ogloszeniezainteresowanie oz where oz.ogloszenie = op.idogloszenia) between 3 and 5

and aplikacje.ilosc = (select count(\*) from dbo.kandydat ka where ka.ogloszenie = op.idogloszenia and dataaplikacji between '2018/01/01' and '2019/12/31')

and k.dzialalnosc is not null

order by aplikacje.ilosc desc

1. Zapytanie NoSQL

db.ogloszenie.aggregate([

{

$match: {

$or: [

{ogloszenieZainteresowanieList: {$size: 3}},

{ogloszenieZainteresowanieList: {$size: 4}},

{ogloszenieZainteresowanieList: {$size: 5}}

],

"zleceniodawca.dzialalnosc": {

$exists: true,

$ne: null

},

"kandydatList": {

$not: {

$elemMatch: {

$or: [{

"dataaplikacji": {

$lte : new ISODate("2018-01-01T00:00:00Z")

}

}, {

"dataaplikacji": {

$gte : new ISODate("2019-12-31T23:59:99Z")

}

}]

}

},

$elemMatch: {

"zaakceptowany": "true",

"kandydujacyznajomiIds": {

$exists: true,

$ne: []

}

}

}

}

},

{

$addFields: { kandydat\_count: {$size: { "$ifNull": [ "$kandydatList", [] ] } } }

},

{

$sort: {"kandydat\_count":1}

},

{

$limit: 5

}

]);

1. Test 100 tyś. rekordów

Zapytanie SQL Server:

CPU time = 169464 ms, elapsed time = 17966 ms.

Widok SQL Server:

CPU time = 170929 ms, elapsed time = 17936 ms.

Zapytanie MongoDB:

"executionTimeMillis" : 13,

1. Test 250 tyś. rekordów

Zapytanie SQL Server:

CPU time = 654024 ms, elapsed time = 65153 ms

Widok SQL Server:

CPU time = 663204 ms, elapsed time = 67834 ms.

Zapytanie MongoDB:

"executionTimeMillis" : 403

1. Test 500 tyś. rekordów

Zapytanie SQL Server:

CPU time = 2568138 ms, elapsed time = 244360 ms.

Widok SQL Server:

CPU time = 2563467 ms, elapsed time = 244102 ms.

Zapytanie MongoDB:

"executionTimeMillis" : 1028

1. MongoDB - import

Na początku musiałem wygenerować pliki JSON przy pomocy generatora. Następnie korzystając z komend:

mongoimport --jsonArray --db wat --collection ogloszenie --file C:\sbd4\ogloszenie.json

Zaimportowałem dokumenty z wygenerowanych plików.

1. Napotkane problemy i pomysły

Jednym z problemów było wygenerowanie odpowiedniej ilości rekordów w bazie danych. Napisałem w tym celu program w języku Java generujący zapytania insertujące dla bazy MSSQL i MongoDB. Załączam kod programu w repozytorium. W celu zmiany ilości generowanych rekordów należy zmienić wartość parametru count w klasie Main. Dla testu 100 tysięcy rekordów parametr count ma wartość 12 500(8 encji x 12 500 = 100 000). Zauważyłem, że generowane dane testowe dają bardzo małą szansę na zwrócenie odpowiedniej wartości z powodu warunku:

* Co najmniej dwóch kandydatów jest znajomymi, a co najmniej jeden został przyjęty

W celu rozwiązania tego problemu dodałem parametry:

count\_znajomi = 6.75 x count,

count\_kandydat = 4 x count,

count\_osoba = count / 4.

Dla 100 tyś. rekordów count = 6 500 ( 5 encji \* 6 500 + 6.75 \* 6 500 + 4 \* 6500 + 0.25 \* 6 500 = 100 000).

Plusem generatora jest to, że dane dla bazy relacyjnej i noSQL są generowanie na podstawie tych samych obiektów – testy są bardziej wiarygodne.

W bazie mongoDB nie trzeba było tworzyć schematu – schemat tworzony jest wraz z dodawaniem danych. W celu uzyskania danych odnośnie czasu wykonywania zapytania musiałem wykorzystać komendę:

db.ogloszenie.explain('executionStats').aggregate([]);

1. Ocena środowisk

Porównując oba środowiska ze względu na szybkość działania baza MongoDB wypadła znacznie lepiej od bazy relacyjnej MSSQL. Dotychczas korzystałem głównie z relacyjnych baz danych, więc przestawienie się na bazy noSQL było trudne, tym bardziej pisząc tak skomplikowane zapytanie. Jednak sądzę, że używając dłużej tego narzędzia przyzwyczaiłbym się do tego i pisanie zapytań nie byłoby trudniejsze niż w relacyjnej bazie danych. Ciekawą rzeczą w bazie noSQL jest to, że schemat bazy danych tworzony jest na podstawie zapytań wstawiających. Nie jestem fanem takiego rozwiązania, ponieważ prowadzi to do błędów typów danych(ciekawym rozwiązaniem jest użycie opcji validator podczas tworzenia kolekcji). Jednakże bazy danych noSQL są używane do analizy i przechowywania dużych ilości danych, a w tym zadaniu ważniejsza jest wydajność.

To co mi się podobało na korzyść bazy noSQL to format wprowadzania danych: JSON. Jest to bardzo wygodny i intuicyjny format danych, dodatkowo doskonale widać w nim strukturę dodawanych dokumentów.

Zawartość repozytorium:

sbd4\_generator – program w języku java generujący inserty.

diagram.png – diagram bazy danych

skrypt.sql – schemat bazy danych

zapytanie\_mongodb.json – zapytanie mongodb

zapytanie\_mssql.sql – zapytanie sql

YouTube: <https://youtu.be/FruF_KioMmI>