**Multi-dimensional index: response time for read requests to table with vs without geo-spatial index**

**Benchmark report**

**Opis zadatka**

Zadatak je bio testirati te usporediti performanse čitanja iz baze za geo-lokacijske podakte. Odabrana baza podataka je MongoDB, prvenstveno iz dva razloga – prvo, riječ je o bazi podataka koja je lako dostupna i ima svoj službeni 'image' na Docker Hub-u, te drugo, ova baza se vrlo često koristi kod dizajniranja podatkovno intenzivnih aplikacija.

Uz MongoDB, korišteni alati uključuju Docker – alat za kontejnerizaciju i razvoj aplikacija, te Visual Studio Code – alat za pisanje programskog koda.

**Arhitektura sustava**

Sustav koji smo ovom prilikom testirali ima dvije verzije, svaka sa po dva testna slučaja. Prva verzija sustava sastoji se od baze podataka, python skripte za popunjavanje baze nasumičnim podacima, te programa koji čita podatke iz baze. Rang ili raspon toga koji podaci se dohvaćaju svakim čitanjem je također nasumičan.

A diagram of a computer

Description automatically generated Kod druge verzije sustava, u bazu se pomoću dodatne python skripte dodaje indeks tipa '2dsphere', koji bi trebao ubrzati dohvaćanje podataka iz baze kada je riječ o podacima koji predstavljaju točke sa geografskim koordinatama.

Testni slučajevi u obje verzije sustava odnose se na to koliko je podataka pri fazi unošenja unijeto u bazu, odnosno koliko dokumenata imamo u bazi prilikom pretraživanja. Kod prvog testnog slučaja, za obje verzije sustava, taj broj je 10.000, dok kod drugog imamo 100.000 dokumenata u bazi.

**Pomoćne skripte**

Osim same baze podataka, koja je realizirana jednostavnim povlačenjem MongoDB slike sa Docker Hub-a i pokretanjem njene instance u kontejneru, imamo tri pomoćne skripte pisane u programskom jeziku Python.

Prva skripta, nazvana „read.py“ sadrži programski kod koji čita podatke iz baze, te mjeri koliko je vremena potrebno za čitanje i ispis podataka na ekranu.

A screen shot of a computer program

Description automatically generated

Skripta se najprije povezuje na instancu MongoDB kontejnera kojem pristupa putem port-a 27017, te određuje varijablu 'collection', iz koje čita podatke. Najprije pomoću funkcije 'randint' iz biblioteke 'random' određujemo gornju i donju granicu geografske širine i dužine, te potom slažemo upit koji pronalazi sve točke u bazi čije koordinate ulaze u raspon definiran gornjim i donjim granicama.

Za mjerenje vremena potrebnog za dohvaćanje podataka, koristimo modul 'time' i njegovu funkciju 'time'. Najprije odredimo jedan trenutak (t1) nakon kojeg krećemo dohvaćati podatke iz baze, a nakon što smo završili sa dohvaćanjem podataka, te nakon što smo podatke prikazali i izbrojali broj dokumenata koji su prikazani pomoću inkrementa brojača 'num', određujemo drugi trenutak (t2). Razliku trenutaka t2 i t1 prikazujemo kao vrijeme potrebno za odrađivanje upita.

A screen shot of a computer program

Description automatically generated

Što se tiče unosa podataka u našu bazu, python skripta koja je za to zadužena koristi isti način pristupa bazi kao i prethodna skripta. Koristimo funkciju 'MongoClient' iz biblioteke 'pymongo', te pristupamo bazi putem port-a 27017.

Prilikom izvođenja, skripta postavlja pitanje koliko dokumenata želimo unijeti u bazu, te pomoću for-petlje unosi nasumične dokumente u bazu toliko puta koliko je definirano. U našem slučaju, prvi put je uneseno 10.000 dokumenata, a drugi put 100.000 dokumenata.

A screen shot of a computer

Description automatically generated

Skipta za postavljanje indeksa je analogna prethodnoj, samo što umijesto dodavanja podataka dodajemo samo geo-lokacijski indeks tipa '2dsphere'.

**Testiranje**

Testiranje sustava provođeno je u koracima:

1. Postavljanje kontejnera sa instancom MongoDB i kontejnerizacija skripte „read.py“
2. Unos 10.000 dokumenata putem skripte „insert\_fake.py“
3. Uzastopno čitanje pomoću kontejnerizirane skripte „read.py“
4. Zapis rezultata
5. Unos indeksa pomoću skripte „insert\_fake\_index.py“
6. Uzastopno čitanje pomoću kontejnerizirane skripte „read.py“
7. Zapis rezultata
8. Zaustavljanje, brisanje, te ponovno postavljanje kontejnera sa instancom MongoDB
9. Unos 100.000 dokumenata putem skripte „insert\_fake.py“
10. Uzastopno čitanje pomoću kontejnerizirane skripte „read.py“
11. Zapis rezultata
12. Unos indeksa pomoću skripte „insert\_fake\_index.py“
13. Uzastopno čitanje pomoću kontejnerizirane skripte „read.py“
14. Zapis rezultata
15. Zaustavljanje i brisanje kontejnera sa intancom MongoDB

**Prikaz rezultata**

Broj dokumenata – broj dokumenata dohvaćenih prilikom svakog pokretanja skripte „read.py“

Vrijeme čitanja – vijeme potrebno za izvođenje upita i prikaza podataka u bazi, prikazano u sekundama (s)

Brzina čitanja – (vrijeme čitanja / broj dokumenata) \* 1000, prosječno vrijeme potrebno za čitanje svakog dokumenta, prikazano u milisekundama (ms)

**A table of numbers and numbers

Description automatically generated**

Prilikom testiranja dobili smo rezultate koji nisu ono čemu bismo se nadali. Naime, kod korištenja 2dsphere indeksa u bazi koja sadrži 10.000 dokumenata primjećujemo razliku u prosječnom vremenu potrebnom za dohvaćanje svakog dokumenta, koje je prilikom upotrebe indeksa smanjeno za 0.04 ms, dok u slučaju sa 100.000 dokumenata u bazi vidimo uvećanje prosječnog vremena potrebnog za dohvaćanje i ispis dokumenata za 0.01 ms.

Drugim riječima, kod oba slučaja, neovisno o veličini baze podataka, nemamo značajnu razliku prilikom čitanja iz baze bez indeksa i uz pomoć indeksa. U drugom slučaju imamo čak i povećanje potrebnog vremena za čitanje iz baze kada koristimo indeks u odnosu na vrijeme kada ga ne koristimo.