Porovnanie efektívnosti vybraných noSQL databáz

Architektúra softvérových systémov

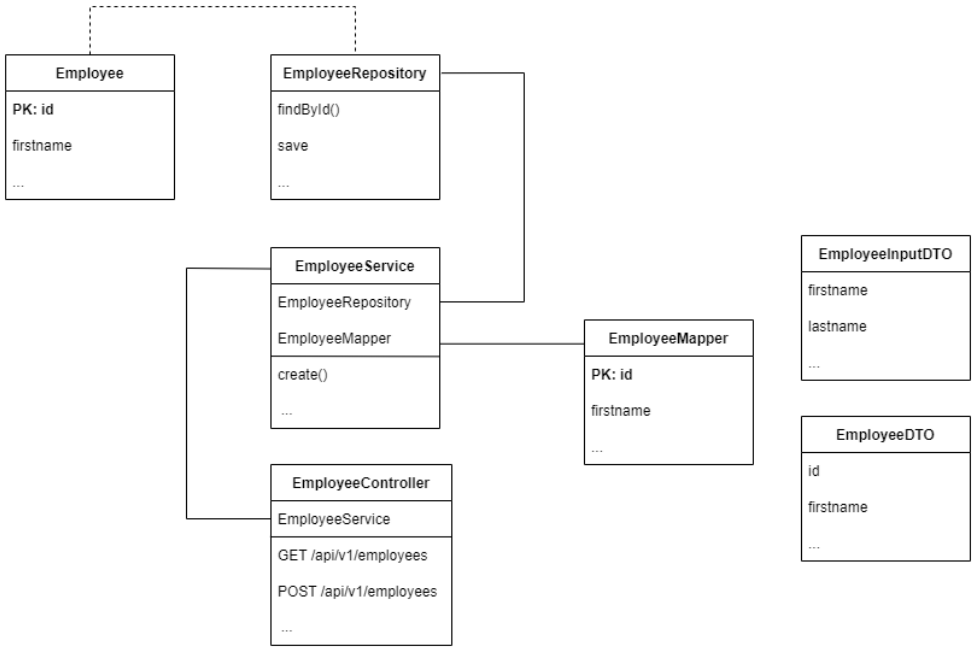
Zuzana Medzihradská, Peter Krajčí, Miroslav Kopecký,
Patrik Kupčulák, David Gellen

Aplikácia slúži na možnosť rezervácie jednotlivých sedadiel vo firme. Skladá sa z backendovej časti napísanej v Springu napojenej na lokálnu postgres databázu, redis databázu, neo4j databázu **sjnflidsbfv**

1. Návrh aplikácie

Keďže ide o backend REST aplikácie, princípom je vytvorenie základných endpointov na základe metodiku CRUD, teda GET, POST, PUT a DELETE.

Aplikácia je napísaná v Jave, konkrétne v Springu, a podlieha základnej Spring hierarchii pre prácu s databázou. Príklad môžeme vidieť pre Triedy pracujúce so zamestnancami.

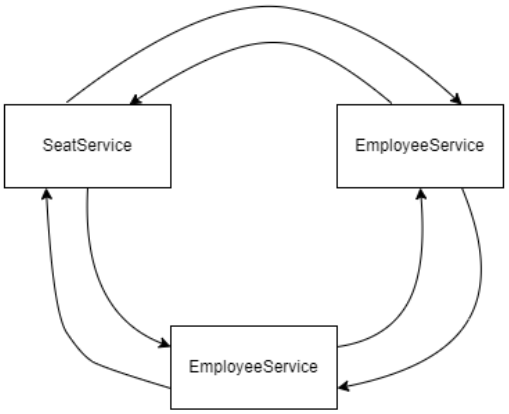


Spring má v ponuke množstvo knižníc pracujúcich s rôznymi typmi databáz ako napr. Spring Data JPA pre SQL databázy, Spring Data Mongo pre prácu s MongoDB či iné. Tieto knižnice ponúkajú svoje implementácie tzv. Repositories, teda interfaceov zabezpečujúcich kontakt s databázou, v ktorých vieme generovať samotné príkazy, ktoré sa nad databázou vykonajú. Tieto Repositories takisto vedia objekty vytiahnuté z databázy priamo mapovať na triedy v Jave (napr. Employee).

Samotná logika sa nachádza v Service triedach, ktoré majú v sebe injectnuté Repositories. Metódy v Service-och volajú Repositories, prípadne sa starajú o dodatočnú logiku ak je potrebná).

Service majú takisto v sebe Mappere (použité z knižnice mapstruct), ktoré transformujú telá prichádzajúcich requestov (triedy končiace na InputDTO) na triedy mapované do databáz a takisto telá odchádzajúcich response-ov (triedy končiace na DTO).

Service sú navzájom prepojené, keďže v niektorých prípadoch je potrebné, aby jeden Service zasiahol do databázy iného mapovaného objektu.



Samotné endpointy sa nachádzajú v Controller triedach, ktoré v sebe majú injectnuté príslušný Service.

EmployeeController

GET *api/v1/employees* vráti zoznam všetkých zamestnancov

GET *api/v1/employees/{id}* vráti zamestnanca s id {id}

POST *api/v1/employees* vytvorí nového zamestnanca, požaduje sa telo requestu

PUT *api/v1/employees/{id}* upraví zamestnanca s id {id}, požaduje sa telo requestu

DELETE *api/v1/employees/{id}* vymaže zamestnanca s id {id}

SeatController

GET *api/v1/seats* vráti zoznam všetkých sedadiel

GET *api/v1/seats /{id}* vráti sedadlo s id {id}

POST *api/v1/seats* vytvorí nové sedadlo, požaduje sa telo requestu

PUT *api/v1/seats /{id}* upraví sedadlo s id {id}, požaduje sa telo requestu

DELETE *api/v1/seats /{id}* vymaže sedadlo s id {id}

ReservationController

GET *api/v1/reservations* vráti zoznam všetkých rezervácii

GET *api/v1/reservations /{id}* vráti rezerváciu s id {id}

POST *api/v1/reservations* vytvorí novú rezerváciu, požaduje sa telo requestu

PUT *api/v1/reservations /{id}* upraví rezerváciu s id {id}, požaduje sa telo requestu

DELETE *api/v1/reservations /{id}* vymaže rezerváciu s id {id}

2. SQL verzia

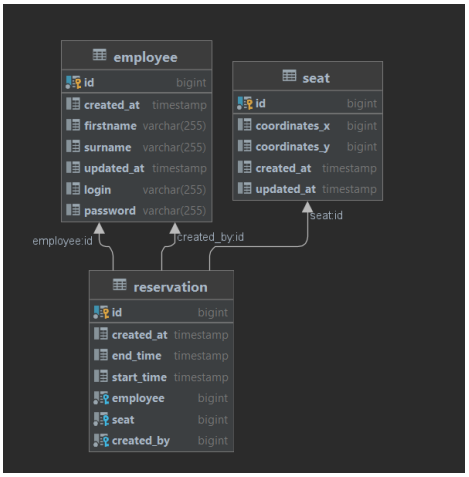
Teória

Celým menom **Structured Query Language** je počítačový jazyk na manipuláciu (výber, vkladanie, úpravu a mazanie) a definíciu údajov. V súčasnosti je to najpoužívanejší jazyk tohto druhu v relačných systémoch riadenia databáz. Existuje množstvo manažovacích systémov resp. distribúcii relačných konkr. SQL databáz ako napr. mySQL, PostgreSQL, MS SQL Server a iné.

Implementácia

Na implementáciu sme si zvolili PostgreSQL systém kvôli jednoduchému testovaniu. Databáza sa skladá z 3 tabuliek:

- **employee** – zamestnanec vytvárajúci rezervácie, má v sebe stĺpce
  - firstname
  - surname
  - login
  - password
  - created_at
  - updated_at
- **seat** – sedadlo, na ktoré sa robí samotná rezervácia, má v sebe stĺpce
  - coordinates_x
  - coordinates_y
  - created_at
  - updated_at
- **reservation** – samotná rezervácia viazaná na zamestnanca a sedadlo, má v sebe stĺpce
  - FK: employee
  - FK: seat
  - FK: created_by (tiež prepojená s employee)
  - start_time
  - end_time
  - created_at



Vo všetkých triedach sú obojstranné referencie, čiže napr. trieda Seat má v sebe odkaz na zoznam rezervácii, ktorých je súčasťou

```
@OneToMany(fetch = FetchType.LAZY, mappedBy = "seat",
        cascade = CascadeType.ALL)
private List<Reservation> reservations = new ArrayList<>();
```

a trieda Reservation má v sebe odkaz na Seat.

```
@ManyToOne()
@JoinColumn(name = "seat", referencedColumnName = "id", nullable = false)
private Seat seat;
```

Toto znamená, že napr. pri mazaní sedadla je potrebné najprv vymazať príslušné rezervácie a až následne môžeme mazať samotné sedadlo. Tento čas nebol bratý do úvahy pri meraní časovej náročnosti operácii nad databázou, keďže bez spätnej referencie by sa dodatočné premazania/uloženie entity cez JPA nekonalo.

Kontakt s databázou zabezpečuje knižnica Jpa s rozhraním JpaRepository

```
public interface ReservationRepository extends JpaRepository<Reservation, Long>
```

3. DOKUMENT

Teória

Dokumentová databáza (známa aj ako databáza orientovaná na dokumenty alebo úložisko dokumentov) je databáza, ktorá ukladá informácie v dokumentoch. Dokumentové databázy ponúkajú množstvo výhod: Intuitívny dátový model, s ktorým sa vývojárom pracuje rýchlo a jednoducho. Flexibilná schéma, ktorá

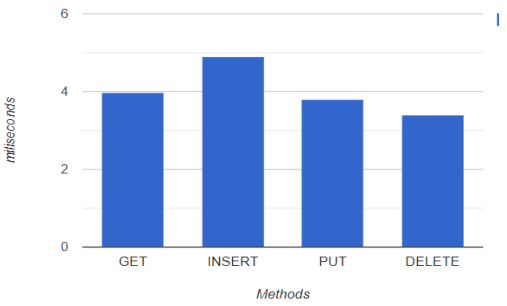
umožňuje, aby sa dátový model vyvíjal podľa potreby aplikácie a je možné ich horizontálne škálovať. Vďaka tomu sú dokumentové databázy univerzálne. Databázy dokumentov sa považujú za nerelačné (alebo NoSQL) databázy. Tieto databázy sú najobľúbenejšou alternatívou k tabuľkovým, relačným databázam. Samotný dokument môže byť reprezentovaný v JSON, BSON alebo XML.

Implementácia

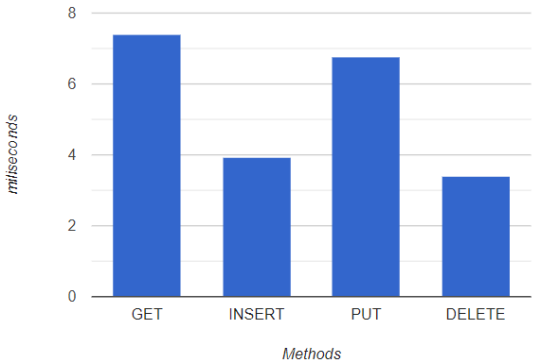
Ako implementáciu dokumentovej databázy sme využili populárnu MongoDB. Pomocou dostupných nástrojov sa nám podarilo vygenerované dáta z SQL databázy exportovať ako JSON a priamo importovať do dokumentovej databázy cez GUI MongoDB compass. Našou úlohou bolo porovnať relačnú a nerelačnú databázu, čo sme dosiahli pomocou experimentov s jednotlivými CRUD metódami pre SQL aj NoSQL dataset. Pred a po každej interakcii s databázou sme si pomocou funkcie System.currentTimeMillis() zistili aktuálny čas a rozdielom týchto dvoch premenných sme získali hodnotu v milisekundách potrebnú na vykonanie konkrétnej operácie. Pre každú metódu sme vykonali 50 operácii. Oba typy databáz mali dve veľkostné varianty (1000 dokumentov a 10000 dokumentov). Výsledky boli nasledovné:

SQL -1000 dokumentov

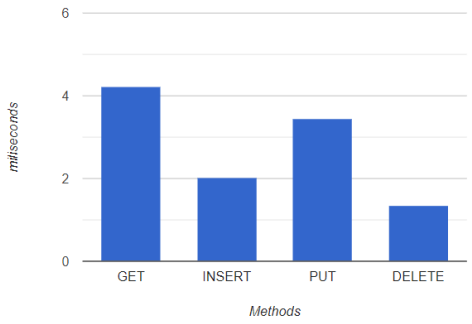
NoSQL -1000 dokumentov



SQL -10000 dokumentov



NoSQL -10000 dokumentov



4. SLOVNÍK (KEY - VALUE)

Teória

Key-value databáza, známa aj ako key-value úložisko, je typ NoSQL databázy. Narozdiel od relačných databáz, ktoré uchovávajú dáta v tabuľkách s definovanými stĺpcami a ich dátovými typmi, key-value databázy majú jednotlivé hodnoty uložené v unikátnych kľúčoch alebo kombinácii kľúčov.

Key-value páry sú podobné rôznym implementáciám hashovacích tabuliek, ako napríklad Dictionaries v Pythone alebo objekty v JavaScripte. Obsahujú in-memory dátovú štruktúru, ktorá je namapovaná na dáta uložené na disku.

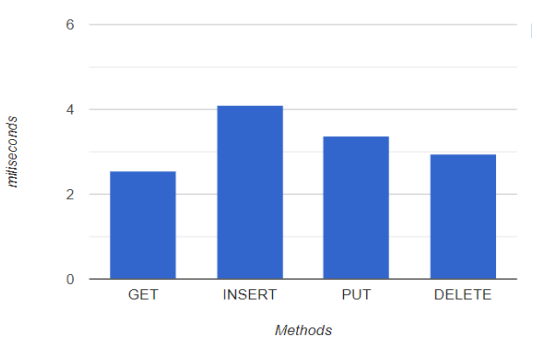
Sú považované za “najjednoduchší” typ NoSQL databáz. Ich prednosťami sú škálovateľnosť a rýchlosť pri čítaní a zapisovaní dát.

Implementácia

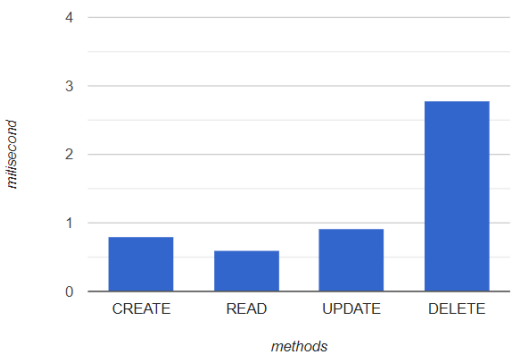
Ako slovníkovú databázu sme si vybrali databázu Redis. V aplikácii sme s Redis serverom komunikovali pomocou RedisClienta, ktorý poskytuje Java knižnica LETTUCE. Nakoľko sa nám nepodarilo implementovať plne funkčné mapovanie na entity, toto mapovanie sme implementovali ručne. RedisClient má v balíku všetky metódy, ktoré nahrádzajú príkazy v redis termináli. Pracovali sme teda s jednoduchými dátovými typmi ako String či Long, a so zložitejšími Set<String> a Map<String, String>.

Naším cieľom bolo porovnať relačnú databázu a key-value databázu pomocou CRUD operácií. Merali sme čas, ktorý zabrala interakcia kódu s databázou. Pre každú z CRUD operácií sme vykonali 50 opakovaní. Oba typy databáz mali dve veľkostné varianty (1000 dokumentov a 10000 dokumentov). Výsledky boli nasledovné:

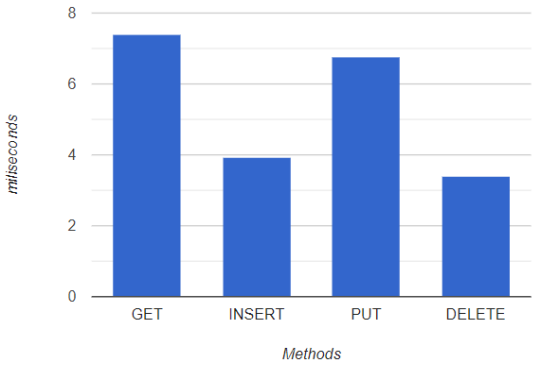
SQL -1000 dokumentov



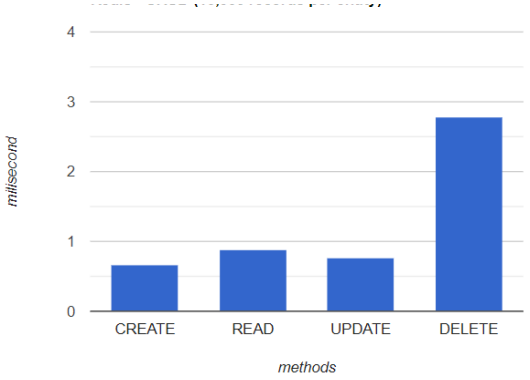
NoSQL -1000 dokumentov

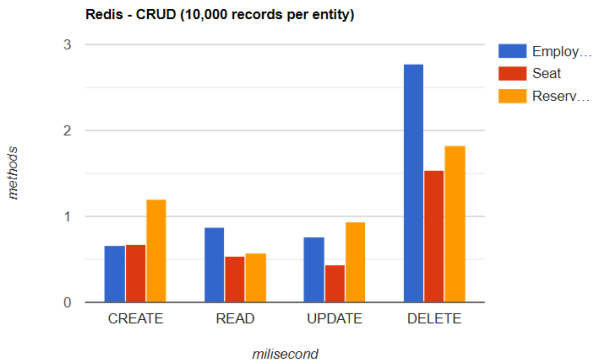


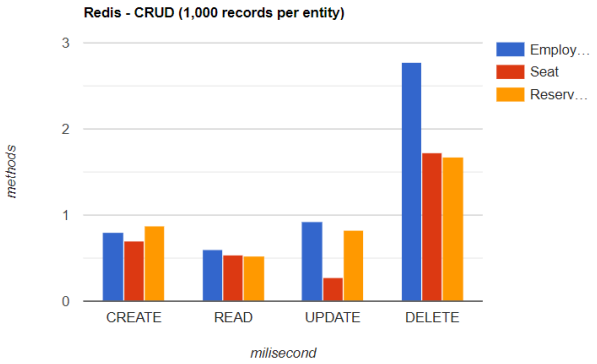
SQL -10000 dokumentov



NoSQL -10000 dokumentov







5. GRAF

Teória

Grafová databáza je využívaná najmä, keď je potrebné vyjadriť v dátovom modeli určitý vzťah medzi 2 entitami. Preto je vo veľkej miere vhodná a aj využívaná spoločnosťami, ktoré vyvíjajú sociálne siete ako napr. Facebook, Twitter, LinkedIn. Vďaka jedinečnej grafovej štruktúre je možné ju použiť aj v rámci odporúčacieho algoritmu, ktorý majú všetky sociálne siete implementovaný, a na základe určitých vzťahov vie systém odporučiť rôzny obsah ako aj reklamy.

V dnešnej dobe je generovaný oveľa väčší počet neštruktúrovaných dát ako v minulosti a to najmä v oblasti sociálnych sietí. Preto je vhodné, na takéto prípady, použiť grafovú noSQL databázu. Ďalšou veľkou výhodou oproti relačným databázam, je oveľa rýchlosť či už čítania z databázy, alebo zapisovania do nej.

V grafovej databáze neexistujú tabuľky ale dáta sú reprezentované vrcholmi a hranami = vzťahmi medzi nimi. Každý vrchol môže mať nejaký názov a každá hrana nejaký typ, pričom vrchol aj hrana môžu mať nejaké properties.

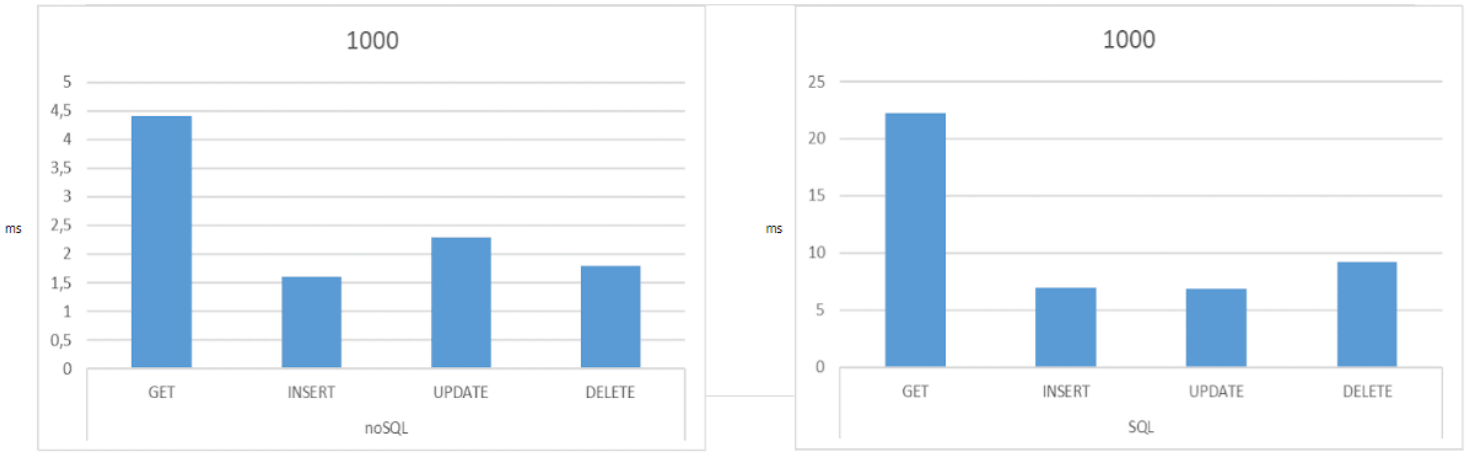
Implementácia

Implementácia grafovej databázy je vykonaná asi v najznámejšej grafovej databáze Neo4j. Vďaka tomu, že je veľmi rozšírená, bolo spustiť ju nenáročné. Konkrétne my sme ju implementovali na Windows zariadení. Na jej spustenie bolo potrebné stiahnuť jeden .zip súbor, rozbaliť a následne pomocou pár príkazov v cmd spustiť. Na porovnanie tejto databázy s SQL databázou bolo potrebné získať rovnaké dáta na porovnanie. Vďaka našej SQL DB bolo jednoduché takéto dáta vygenerovať. Vygenerované dáta z SQL databázy sme exportovali ako json, preformátovali pomocou online nástroja do csv a importovali do Neo4j. Následne prebiehalo meranie CRUD metód na 2 rôzne veľkých datasetoch, pričom štruktúra z tabuliek sa zmenila do vrcholov a hrán.

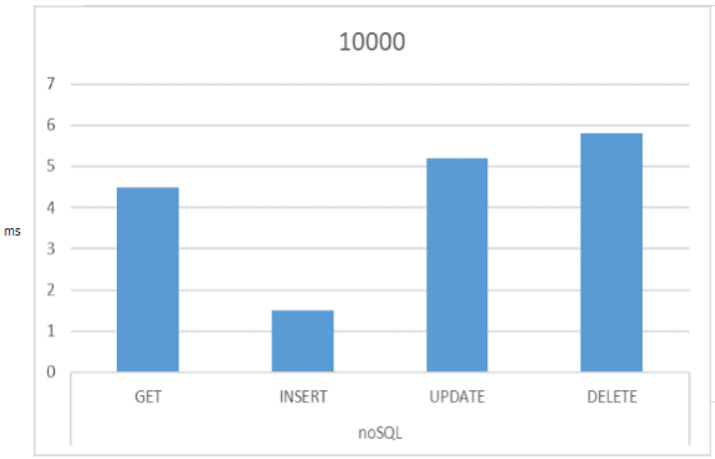
1. dataset:
   - 1000 záznamov z Employee tabuľky
   - 1000 záznamov z Seat tabuľky
   - 1000 záznamov z Reservation tabuľky
2. dataset:
   - 10000 záznamov z Employee tabuľky
   - 10000 záznamov z Seat tabuľky
   - 10000 záznamov z Reservation tabuľky

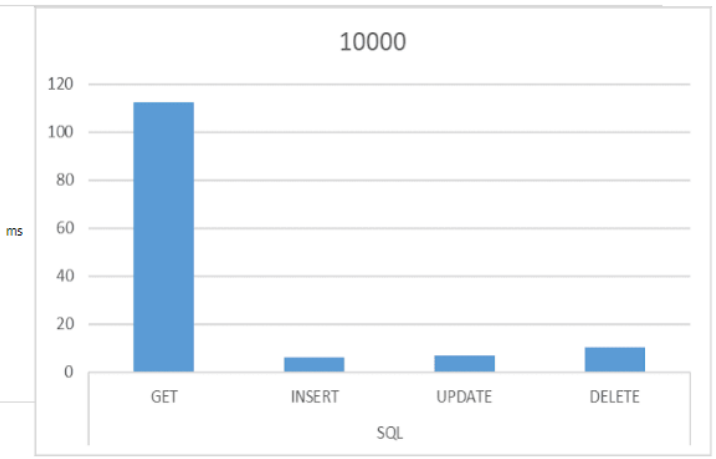
Dosiahnuté výsledky boli nasledovné:

dataset 1000, vľavo noSQL, vpravo SQL, čas v milisekundách



dataset 10000, vľavo noSQL, vpravo SQL, čas v milisekundách


10000
ms
7
6
5
4
3
2
1
0
GET
INSERT
UPDATE
DELETE
noSQL


10000
ms
120
100
80
60
40
20
0
GET
INSERT
UPDATE
DELETE
SQL

6. WIDE - COLUMN STORE

Teória

Wide-column databáza je databáza NoSQL, ktorá organizuje ukladanie údajov do flexibilných stĺpcov, ktoré možno rozložiť na viacero serverov alebo databázových uzlov pomocou viacrozmerného mapovania na referenčné údaje podľa stĺpca, riadka a časovej pečiatky.

Sú vysoko dostupné a škálovateľné, postavené na prácu v distribuovanom prostredí. Väčšina databáz so širokými stĺpcami nepodporuje lineárne škálovanie spojení, pokiaľ ide o výkon čítania a zápisu.