Kolegium Nauk Przyrodniczych
Instytut Informatyki

Przedmiot:

Badawczy projekt zespołowy

Dokumentacja projektu:

Porównanie operacji na różnych silnikach baz danych

Zespół projektowy:

Łukasz Mączka

Adam Młynek

Filip Papiernik

Patryk Paściak

Prowadzący: **dr inż. Piotr Grochowalski**

1. Opis projektu

W niniejszej dokumentacji opisany jest projekt porównywania wydajności silników bazodanowych takich jak MySQL, PostgreSQL, ClickHouse, MongoDB i ArangoDB. Wszystkie bazy danych są uruchomione w środowisku Docker, a ich interakcja z aplikacją jest zrealizowana przy użyciu języka Node.js. Interfejs użytkownika został stworzony w React. Celem projektu jest uzyskanie pełnego i obiektywnego porównania wydajności tych silników bazodanowych i udostępnienie informacji dotyczących ich efektywności dla różnych rodzajów zapytań. Wyniki te będą niezbędne dla projektantów aplikacji, którzy muszą wybrać odpowiedni silnik bazodanowy, aby zapewnić optymalne działanie swoich aplikacji.

2. Wykorzystane technologie

a. Back-end

- Node.js
- TypeScript

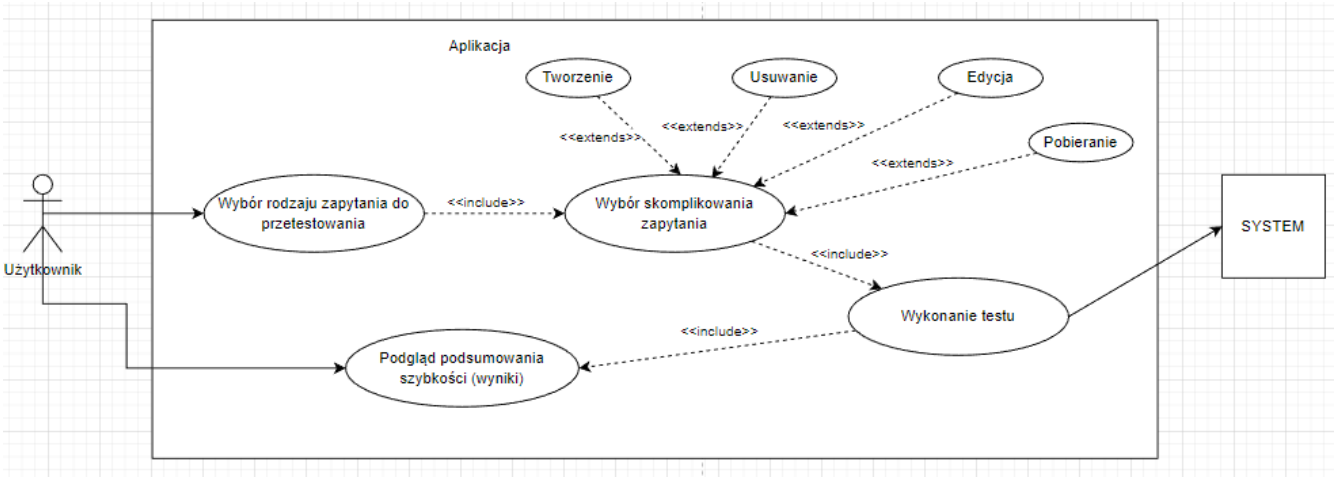
b. Front-end

- React
- TypeScript
- Styled Components

c. Oprogramowanie wirtualizacji

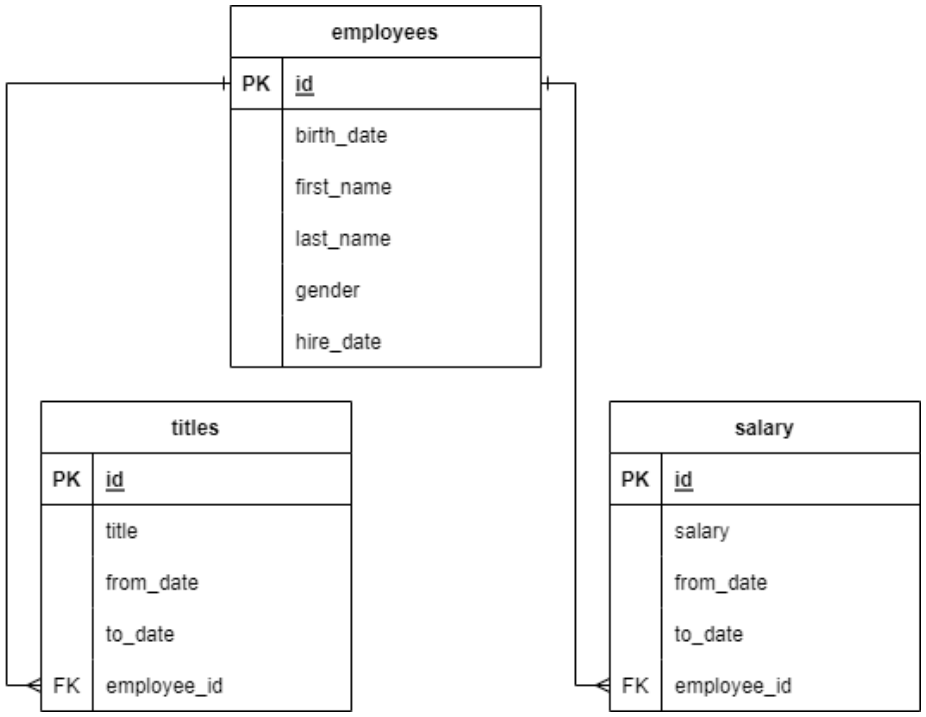
- Docker

3. Diagram UML



4. **Struktura bazy danych**

W tabelii “employees” znajduje się 300 rekordów, titles zawiera 3000 rekordów a tabela salary milion.



5. **Zapytania**

-INSERT

Mysql, Pgsq

```
INSERT INTO salary(employee_id, salary, from_date, to_date)
VALUES ${salary .splice(0, amount) .map( (val) =>
`(${val.employee_id}, ${val.salary}, ${val.from_date},
${val.to_date})`, ).join(',')};
```

Clickhouse

```
this.conn.insert({
                table: 'salary',
                values: JSON.parse('[valuesToAdd]'),
                format: 'JSONEachRow',
              });
```

MongoDB

```
this.conn.db.collection('salary').insertMany(salary.splice(0, amount));
```

ArangoDB

```
this.conn.collection('salary').import(salary.splice(0, amount));
```

-Easy Select

Mysql, Pgsq, ClickHouse

```
SELECT * FROM salary s WHERE s.salary >= 3000
```

MongoDB

```
this.conn.db.collection('salary').find({ salary: { $gte: 5000, $lt: 8000 } }).toArray();
```

ArangoDB

```
FOR doc IN salary
        FILTER doc.salary >= 5000 AND doc.salary < 8000
        RETURN doc;
```

-Medium Select

Mysql, Pgsq

```
SELECT id,
        first_name,
        last_name,
        gender,
        hire_date,
        s.how_many_withdrawals,
        s.smallest_payout,
        s.biggest_payout,
        s.sum_salary,
        t.how_many_titles,
        t.last_promotion
    FROM employees AS e
    LEFT JOIN
      (SELECT count(salary) AS how_many_withdrawals,
            max(salary) AS biggest_payout,
            min(salary) AS smallest_payout,
            sum(salary) AS sum_salary,
            employee_id
       FROM salary
```

```
    GROUP BY employee_id) AS s ON e.id = s.employee_id
  LEFT JOIN
   (SELECT count(title) AS how_many_titles,
         employee_id,
         MAX(from_date) AS last_promotion
    FROM titles
    GROUP BY employee_id) AS t ON e.id = t.employee_id
  WHERE gender = 'F'
   AND hire_date < '2015-01-01'
   AND last_promotion < '2020-01-01'
   AND sum_salary > 100000
  ORDER BY sum_salary DESC
```

ClickHouse

```
SELECT
        e.id, e.first_name, e.last_name, e.gender, e.hire_date,
        s.how_many_withdrawals,s.smallest_payout, s.biggest_payout,
s.sum_salary,
        t.how_many_titles, t.last_promotion
      FROM
        employees e,
        (SELECT
           count(salary) as how_many_withdrawals,
           min(salary) as smallest_payout,
           max(salary) as biggest_payout,
           sum(salary) as sum_salary,
           employee_id
         FROM
           salary
         GROUP BY
           employee_id) AS s,
        (SELECT
           count(title) as how_many_titles,
           MAX(from_date) as last_promotion,
           employee_id
         FROM
           titles
         GROUP BY
           employee_id) AS t
      WHERE
        e.id = s.employee_id AND
```

```
        e.id = t.employee_id AND
        e.gender = 'F' AND
        e.hire_date < '2015-01-01' AND
        s.sum_salary > 100000
      ORDER BY
        s.sum_salary DESC
```

MongoDB

```
this.conn.db.collection('salary').find().toArray()
```

ArangoDB

```
FOR doc IN salary
        RETURN doc
```

-Hard Select

Mysql, Pgsq, ClickHouse

```
SELECT * FROM salary AS s, employees AS e, titles AS t WHERE e.id = t.employee_id AND title LIKE '%BackEnd%' AND e.id = s.employee_id
```

MongoDB

```
this.conn.db.collection('salary').find({ salary: { $gte: 5000 } }).toArray();
```

ArangoDB

```
FOR doc IN salary
        FILTER doc.salary >= 5000
        RETURN doc
```

-Easy Update

Mysql, Pgsq

```
UPDATE salary SET salary = 2500 WHERE salary < 2000
```

ClickHouse

```
ALTER TABLE
            salary
          UPDATE
            salary = 2500
          WHERE
            salary < 2000
```

MongoDB

```
this.conn.db.collection('salary').updateMany({ salary: { $gte: 5000, $lt:
8000 } }, { $set: { salary: 2500 } });
```

ArangoDB

```
FOR doc IN salary
            FILTER doc.salary >= 5000 AND doc.salary < 8000
            UPDATE doc WITH { salary: 2500 } IN salary
            RETURN doc
```

-Medium Update

Mysql

```
UPDATE employees AS e
            INNER JOIN salary AS s
              ON s.employee_id = e.id
          SET s.salary = 4500
          WHERE e.gender = 'M' AND s.salary < 3000 AND e.hire_date
> '2000-01-01'
```

Pgsq

```
UPDATE salary AS s
            SET salary = 4500
          FROM employees AS e
          WHERE s.employee_id = e.id
            AND e.gender = 'M'
            AND s.salary < 3000
            AND e.hire_date > '2000-01-01'
```

ClickHouse

```
ALTER TABLE
            salary
          UPDATE
            salary = 4500
          WHERE
            employee_id IN (
              SELECT
                id
              FROM
                employees
              WHERE
                gender = 'M' AND
                hire_date > '2000-01-01'
            ) AND
            salary < 3000
```

MongoDB

```
this.conn.db.collection('salary').updateMany({ salary: { $gte: 5000 } },
{ $set: { salary: 2500 } })
```

ArangoDB

```
FOR doc IN salary
            FILTER doc.salary >= 5000
            UPDATE doc WITH { salary: 2500 } IN salary
            RETURN doc
```

-Hard Update

Mysql

```
UPDATE employees
        JOIN (SELECT e.id
             FROM salary s
             JOIN employees e ON e.id = s.employee_id
             JOIN titles t ON e.id = t.employee_id
             WHERE s.salary > 2000 AND s.salary < 10000) t ON t.id =
employees.id
        SET hire_date = '2023-01-01';
```

Pgsq

```
UPDATE employees
        SET hire_date = '2023-01-01'
        WHERE id IN (
        SELECT e.id
        FROM salary s
        JOIN employees e ON e.id = s.employee_id
        JOIN titles t ON e.id = t.employee_id
        WHERE s.salary > 2000 AND s.salary < 10000
```

ClickHouse

```
ALTER TABLE
        employees
      UPDATE
        hire_date = '2023-01-01'
      WHERE
        id IN (
          SELECT DISTINCT
            e.id
          FROM
            salary s,
            employees e,
            titles t
          WHERE
            e.id = s.employee_id AND
            e.id = t.employee_id AND
            s.salary > 2000 AND
            s.salary < 10000 )
```

MongoDB

```
this.conn.db.collection('salary').updateMany({}, { $set: { salary: 2500 } });
```

ArangoDB

```
FOR doc IN salary
        UPDATE doc WITH { salary: 2500 } IN salary
        RETURN doc
```

-Easy Delete

Mysql, Pgsql

```
DELETE FROM titles WHERE title = 'Junior BackEnd';
```

ClickHouse

```
ALTER TABLE
        titles
      DELETE WHERE
        title = 'Junior BackEnd'
```

MongoDB

```
this.conn.db.collection('salary').deleteMany({ salary: { $gte: 5000, $lt: 8000 } });
```

ArangoDB

```
FOR doc IN salary
        FILTER doc.salary >= 5000 AND doc.salary < 8000
        REMOVE doc IN salary
        RETURN doc
```

-Medium Delete

Mysql, Pgsql

```
DELETE FROM salary WHERE salary > 1500 AND salary < 7500 AND from_date > '2011-01-01' AND to_date < '2020-01-01'
```

ClickHouse

```
ALTER TABLE
        salary
      DELETE WHERE
        salary > 1500 AND
        salary < 7500 AND
        from_date > '2011-01-01' AND
        to_date < '2020-01-01'
```

MongoDB

```
this.conn.db.collection('salary').deleteMany({});
```

ArangoDB

```
FOR doc IN salary
        REMOVE doc IN salary
        RETURN doc
```

-Hard Delete

Mysql, Pgsql

```
DELETE FROM
        employees
      WHERE
        id IN (SELECT DISTINCT e.id FROM salary s, employees e,
titles t WHERE e.id = s.employee_id AND e.id = t.employee_id  AND
s.salary > 2000)'
```

ClickHouse

```
ALTER TABLE
        employees
      DELETE WHERE
        id IN (
          SELECT DISTINCT
            e.id
          FROM
            salary s,
            employees e,
            titles t
          WHERE
            e.id = s.employee_id AND
            e.id = t.employee_id AND
            s.salary > 2000
          )
```

MongoDB

```
this.conn.db.collection('salary').deleteMany({ salary: { $gte: 5000 } })
```

ArangoDB

```
FOR doc IN salary
        FILTER doc.salary >= 5000
        REMOVE doc IN salary
        RETURN doc
```

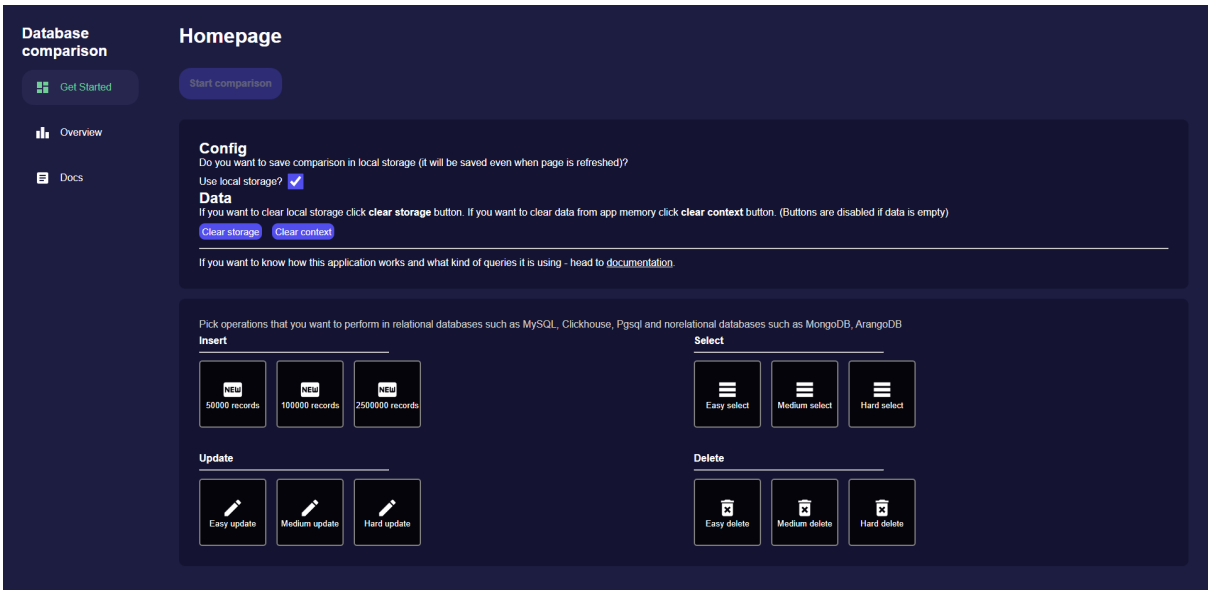
6. **Wymagania systemowe oraz wymagane oprogramowanie**
    a. **Wymagania systemowe**
    - Windows 11 64-bit: Home lub Pro wersja 21H2 lub wyżej, Enterprise lub Education wersja 21H2 lub wyżej.
    - Windows 10 64-bit: Home lub Pro 21H1 (build 19043) lub wyżej, Enterprise lub Education 20H2 (build 19042) lub wyżej.
    - 64-bit procesor wspierający Second Level Address Translation (SLAT)
    - minimum 16 GB RAM
    - uruchomiona wirtualizacja w BIOS

    b. **Oprogramowanie**
    - Program Docker
    - GitBash - wymagany do uruchomienia skryptu “docker.sh” który automatycznie konfiguruje bazy danych w dockerze
7. **Instrukcja uruchomienia**
    a. Wypakować folder projektu
    b. Przejść do folderu “docker” i uruchomić skrypt docker.sh
    c. Przejść do folderu “client”, w terminalu uruchomić polecenie “npm install”, “npm run build” a następnie “npm run preview”
    d. Przejść do folderu “server”, zmienić nazwę pliku .env.sample na .env
    e. W tym samym folderze “server”, w terminalu uruchomić polecenie “npm install”, a następnie “npm run dev”
    f. Projekt dostępny jest pod adresem ‘http://localhost:5173/’

8. Interfejs aplikacji

a. Get Started

Na tej stronie można uruchomić wybrane zapytanie i zmienić konfigurację.

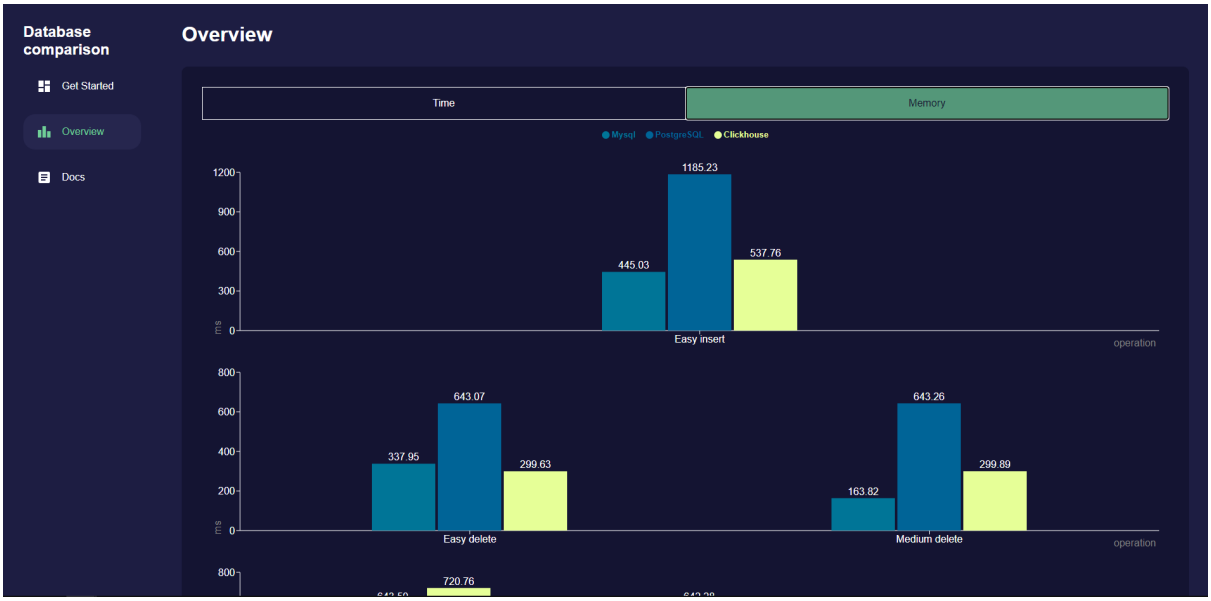


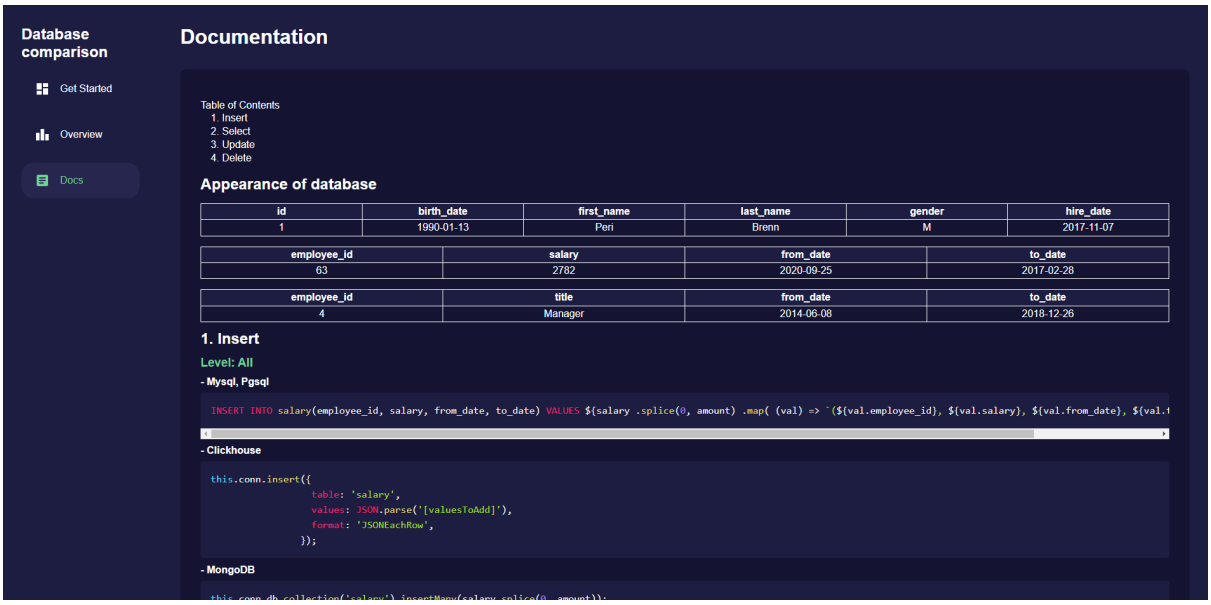
b. Overview (Time)

Na tej stronie widoczne są wyniki poszczególnych zapytań. Wyniki przedstawione zostały za pomocą wykresów i pogrupowane względem zapytań. Wykresy na stronie podzielone zostały na wykresy przedstawiające czas potrzebny na wykonanie zapytania oraz pamięć zużytą przez bazę do

wykonania zapytania.

c. Overview (Memory)


Database comparison
Get Started
Overview
Docs
Overview
Time
Memory
Mysql
PostgreSQL
Clickhouse
1200
900
600
300
0
ms
445.03
1185.23
537.76
Easy insert
operation
800
600
400
200
0
ms
337.95
643.07
299.63
Easy delete
163.82
643.26
299.89
Medium delete
operation
800
720.76

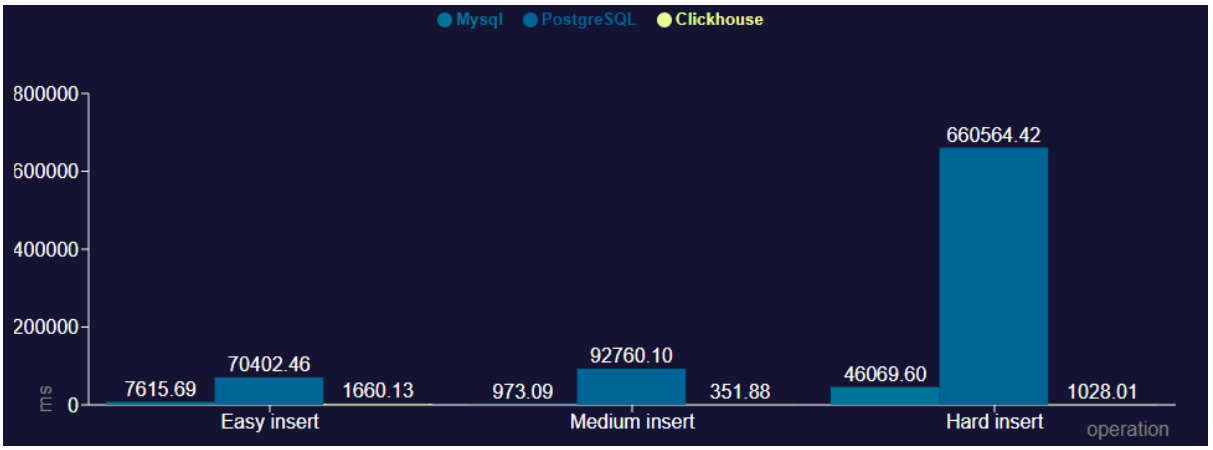

Database comparison
Get Started
Overview
Docs
Documentation
Table of Contents
1. Insert
2. Select
3. Update
4. Delete
Appearance of database
id
birth_date
first_name
last_name
gender
hire_date
1
1990-01-13
Peri
Brenn
M
2017-11-07
employee_id
salary
from_date
to_date
63
2782
2020-09-25
2017-02-28
employee_id
title
from_date
to_date
4
Manager
2014-06-08
2018-12-26
1. Insert
Level: All
- Mysql, Pgsql
INSERT INTO salary(employee_id, salary, from_date, to_date) VALUES ${salary .splice(0, amount) .map((val) => `(${val.employee_id}, ${val.salary}, ${val.from_date}, ${val.t
- Clickhouse
this.conn.insert({
table: 'salary',
values: JSON.parse('[valuesToAdd]'),
format: 'JSONEachRow',
});
- MongoDB

9. Wyniki działania projektu

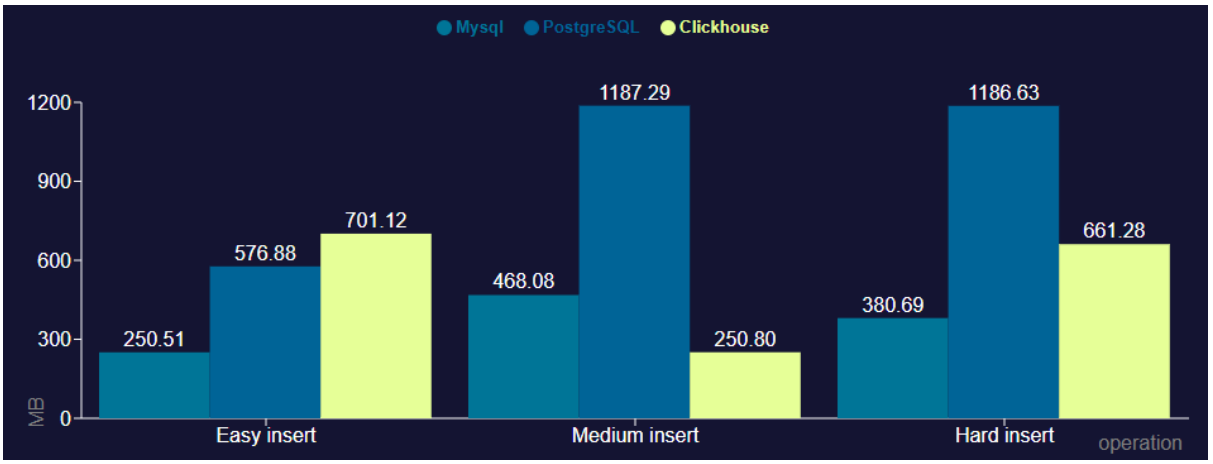
Testy zostały przeprowadzone na komputerze z następującą specyfikacją:
System: Windows 10 64-bitowy,
Procesor: AMD Ryzen 7 5800H 3.2 GHz
Pamięć RAM: 16GB DDR4

a. Insert (SQL)

- **Czas**

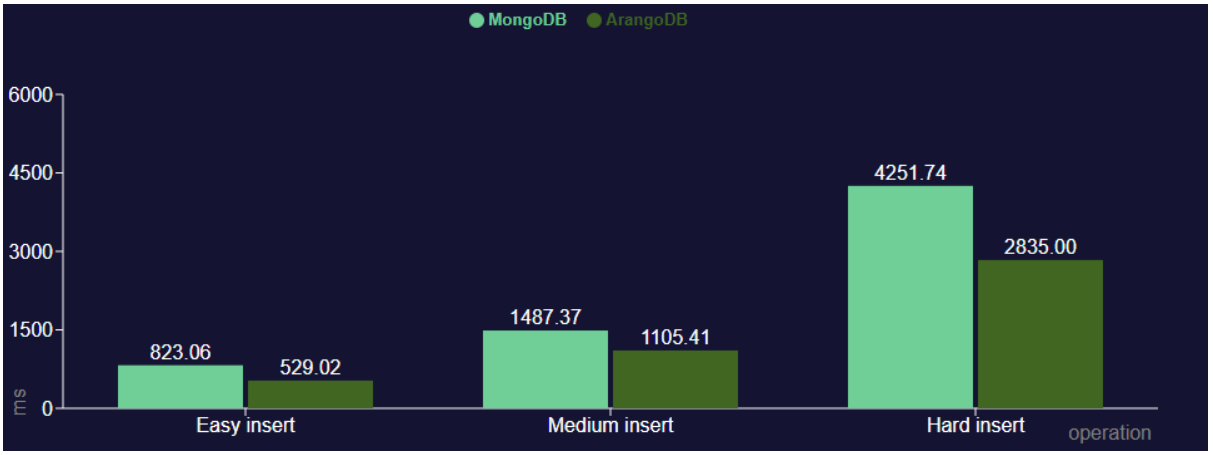


- **Pamięć**

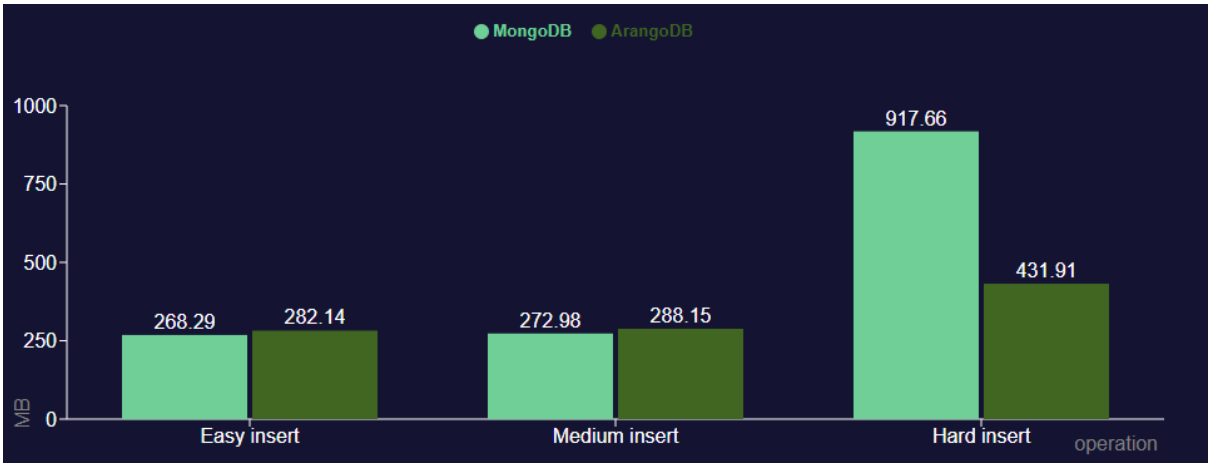


b. Insert (NoSQL)

- **Czas**

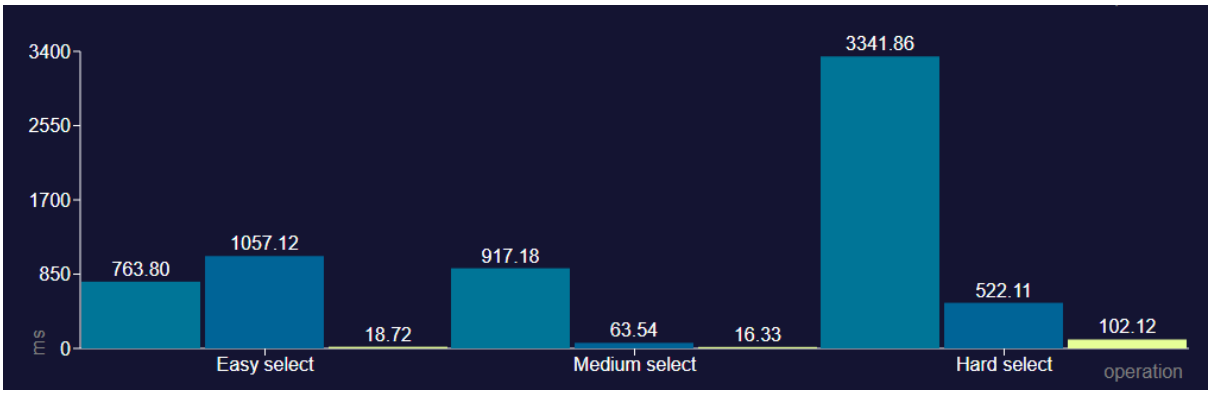


- **Pamięć**

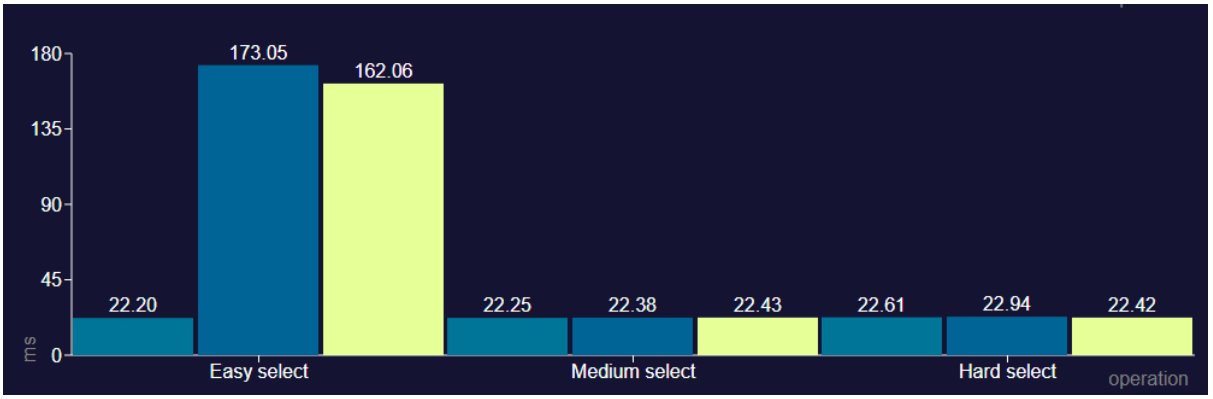


c. Select (SQL)

- **Czas**

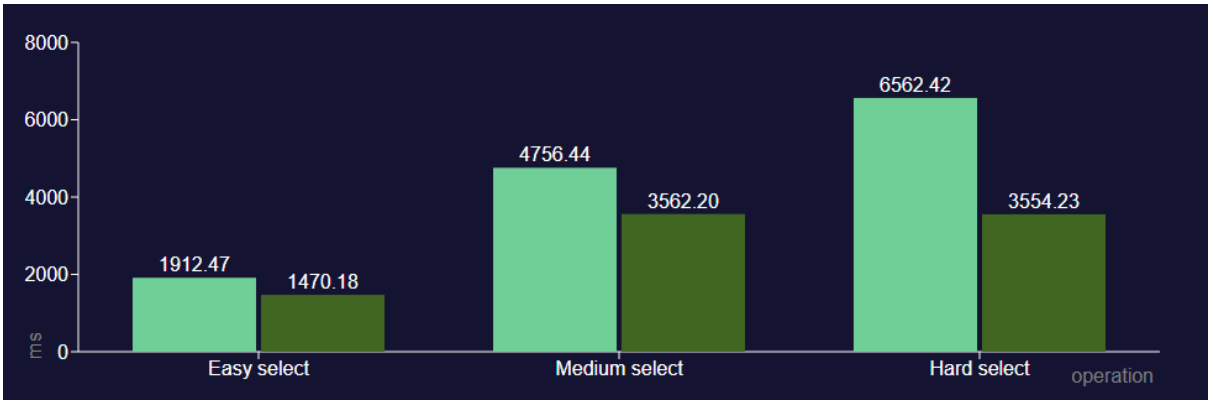


- **Pamięć**

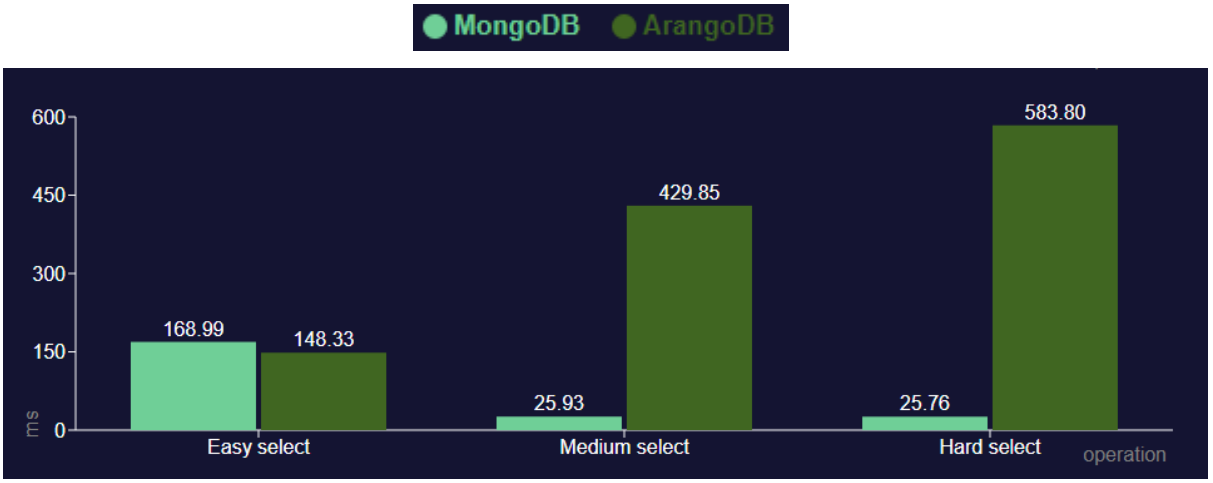


d. Select (NoSQL)

- **Czas**

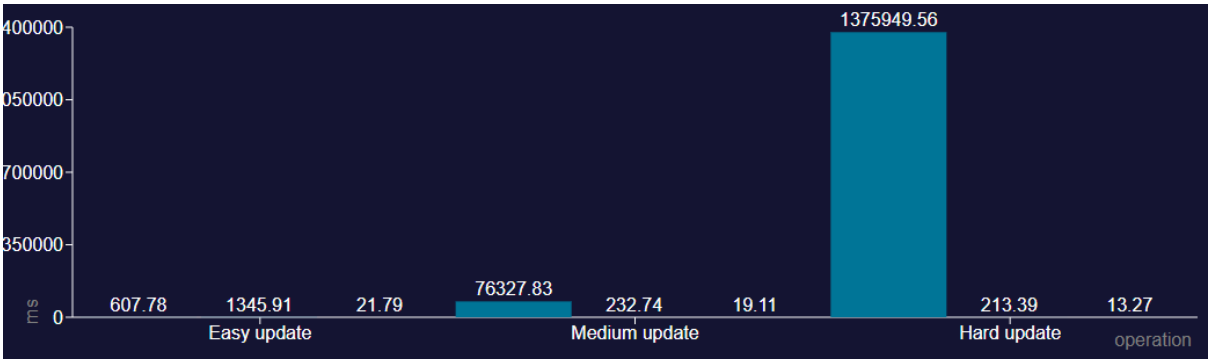


- **Pamięć**

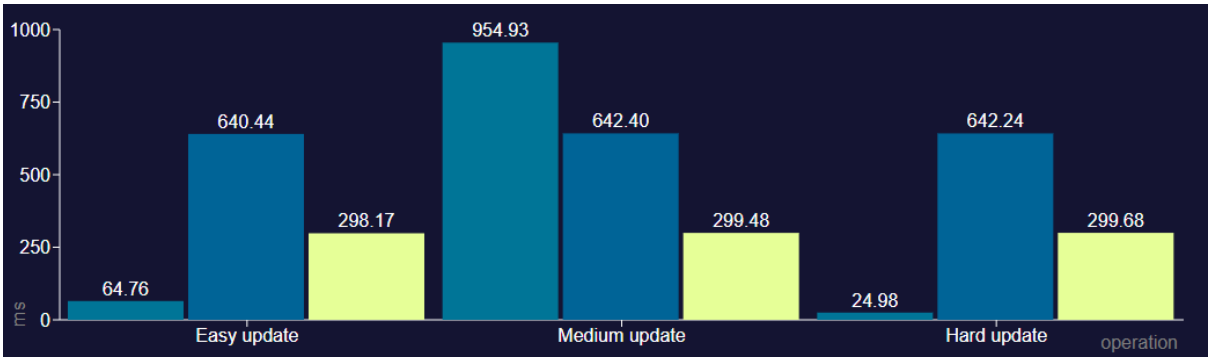


e. Update (SQL)

- **Czas**

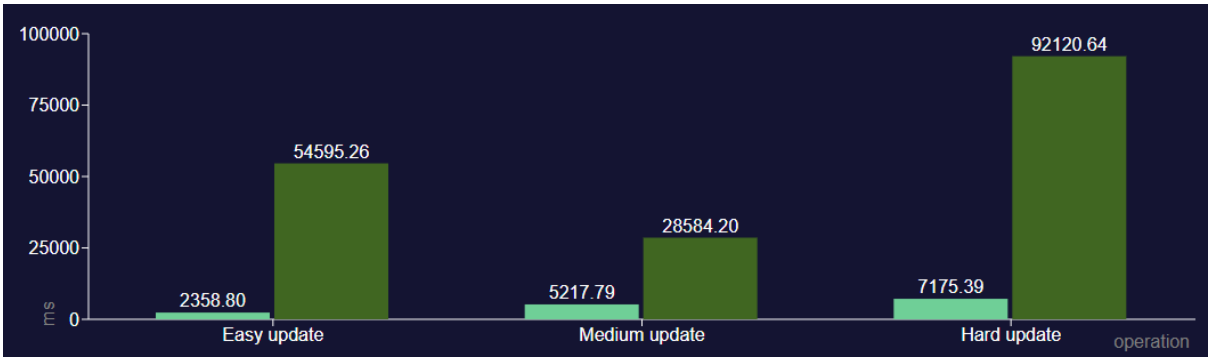


- **Pamięć**

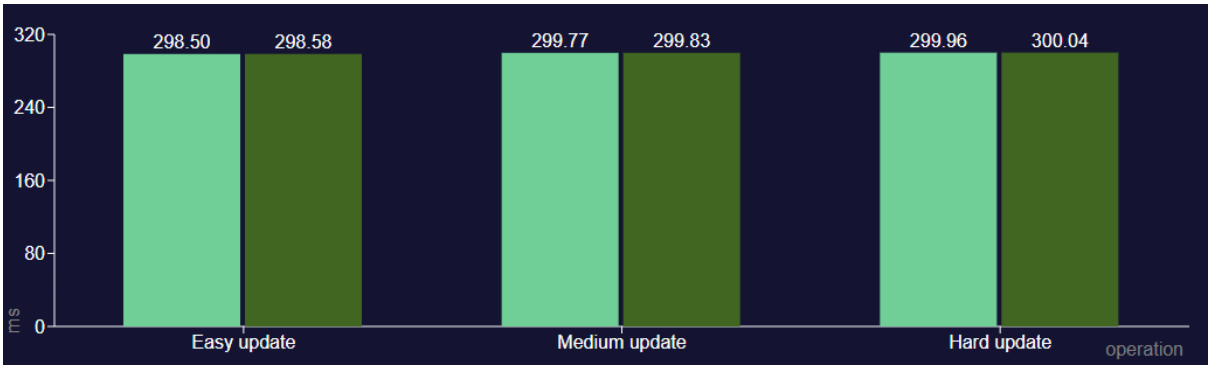


f. Update (NoSQL)

- **Czas**

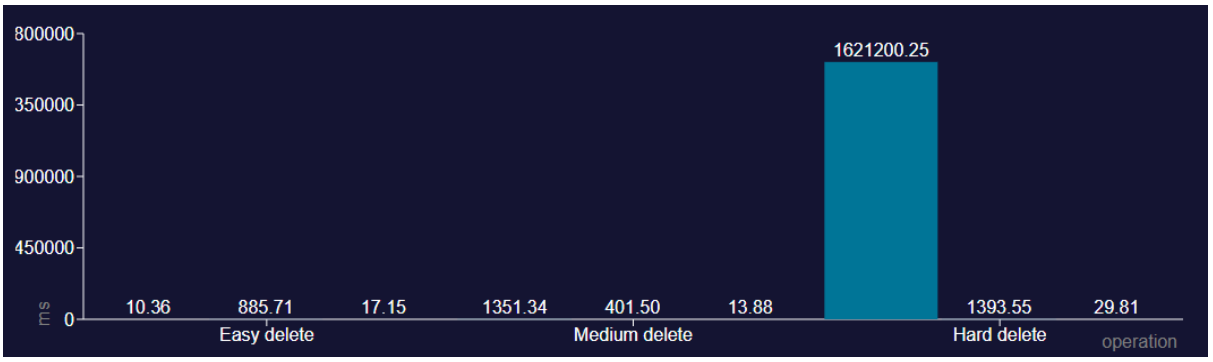


- **Pamięć**

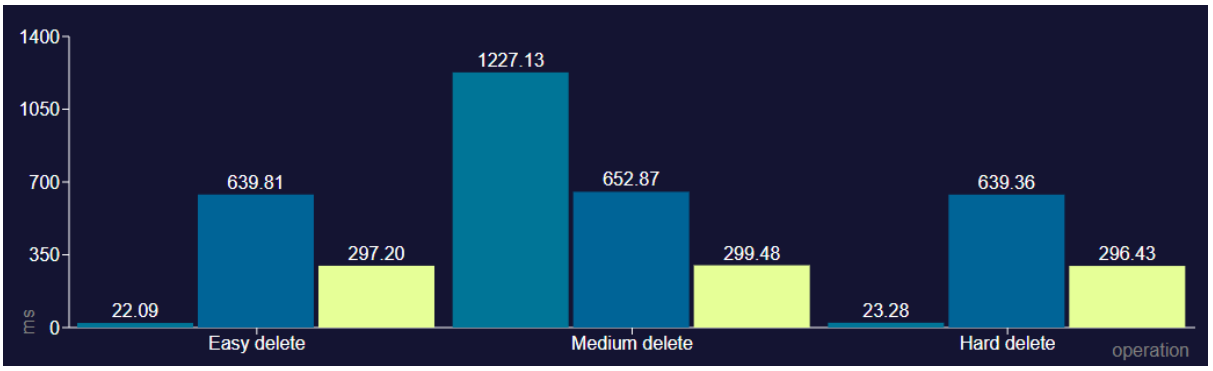


g. Delete (SQL)

- **Czas**

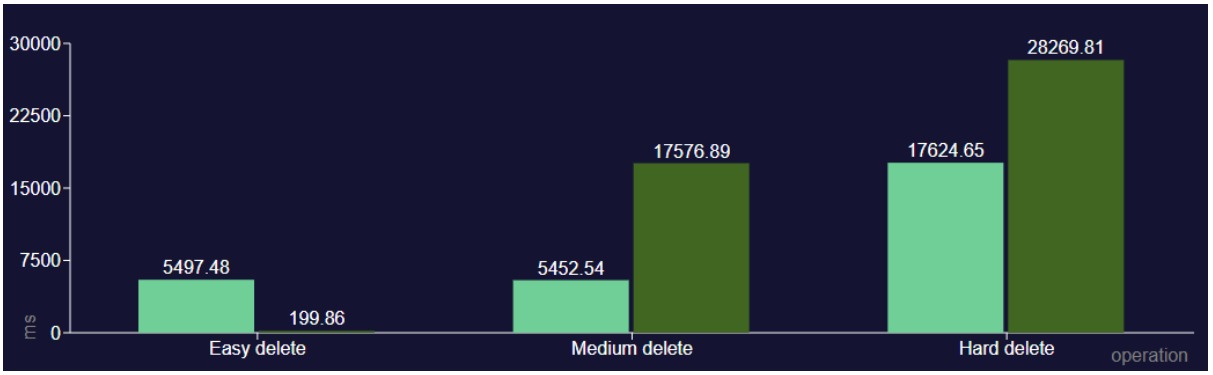


- **Pamięć**

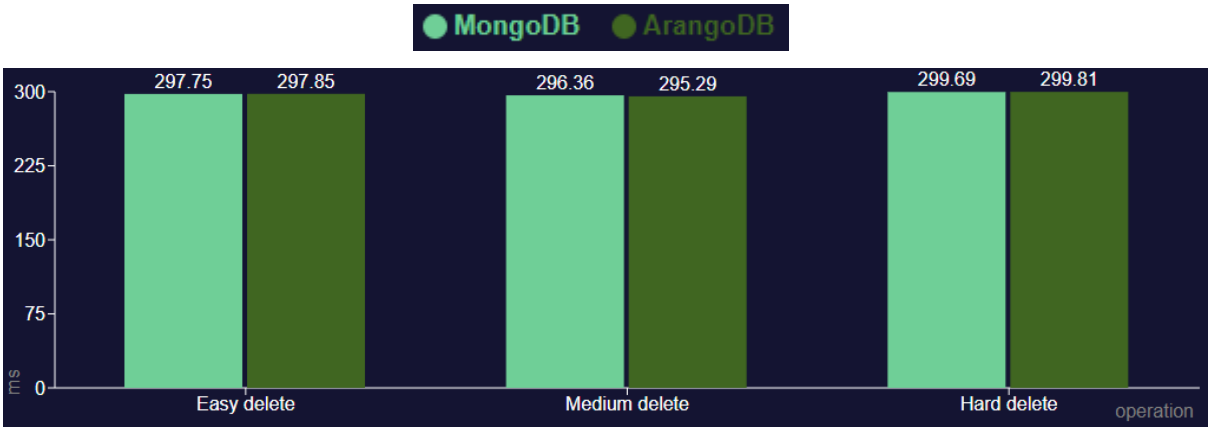


h. Delete (NoSQL)

- **Czas**



- **Pamięć**



10. Wnioski

- Czas

Po uruchomieniu testów wrzucających dane do bazy można zaobserwować, że najlepiej z tym zadaniem radzi sobie ClickHouse który robi to najszybciej, na drugim miejscu znajduje się MySQL który radzi sobie nieco gorzej, natomiast PgSQL radzi sobie kilka razy gorzej niż pozostałe bazy. Gdy zależy nam na szybkości wstawiania do bazy najlepszym rozwiązaniem będzie w kolejności: ClickHouse, MySQL a na końcu PgSQL
W przypadku baz NoSQL, operacja wstawiania jest najszybsza w ArangoDB który jest szybszy w każdym przypadku tj. *50000*, *100000*, *250000*. MongoDB jest wolniejsze zwykle o ok. **30%**.

Operacje usuwania najszybsze są dla ClickHouse, tak naprawde nie gra roli skomplikowanie zapytania - w przeprowadzonych testach operacje usuwania nie przekraczają *30ms*. W przypadku PgSQL długość wykonywania rośnie wraz z jego skomplikowaniem, lecz nie rośnie gwałtownie i w przypadku najcięższej operacji wynosi *1400ms*. Natomiast MySQL rośnie bardzo gwałtownie: Dla łatwej operacji wynosi 10ms, dla średniej *1350ms* a dla

cieżkiej operacji aż *27 minut*, a więc jest to ponad **54 tysiące razy** dłużej niż w przypadku ClickHouse.
Dla baz NoSQL, operacja usuwania jest niejednoznaczna, dla małego skomplikowania szybsza jest baza ArangoDB o ponad *5 sekund,* ale sytuacja szybko się zmienia gdy skomplikowaność zapytań rośnie. MongoDB dla średniego skomplikowania jest szybsza o *69%*, a dla najcieżej operacji o *37%*.

Operacje aktualizowania danych wyglądają bardzo podobnie jak usuwania - w przeprowadzonych testach dalej króluje ClickHouse i nie przekracza *22ms*. PgSQL jest na drugim miejscu i ma odpowiednio: *1345ms*, *232ms*, *213ms*; a więc również wykonuje je dość szybko. Tak samo jak w przypadku usuwania MySQL wypada najgorzej i dla najcięższa operacja trwa ponad *22 minuty*.
W przypadku baz NoSQL, operacje aktualizowania są zdecydowanie szybsze dla bazy MongoDB i wykonują się ok. *80-90%* szybciej niż w przypadku ArangoDB.

Operacje pozyskiwania danych w przypadku baz SQL plasują się następująco: ClickHouse jest najszybszy i zwraca milion rekordów w czasie *100ms.* Na drugim miejscu znajduje się PgSQL, a najgorzej wypada MySQL.
W przypadku baz NoSQL, operacje pobierania danych są najszybsze dla bazy MongoDB i wykonują się ok. *30-40%* szybciej niż te same operacje wykonane w ArangoDB.

- Pamięć

W przypadku użycia pamięci przez silniki bazodanowe dla języków SQL w większości wypadków MySQL posiada najniższe wykorzystanie tego zasobu. Na drugim miejscu, nieco bardziej obciążający jest ClickHouse. Ostatnie miejsce a zajmuje PostreSQL co oznacza że wykorzystuje on najwięcej pamięci spośród silników SQL'owych.

Wstawianie danych było jedną z najbardziej obciążającą pamięć operacją. MySQL i Clickhouse zajmowały podobną ilość pamięci niezależnie od ilości wystawianych danych, z tym że MySQL wykorzystywał jej prawie dwukrotnie mniej. PostgreSQL zajmował dwukrotność pamięci MySQL'a, czyli ~570 mb, w prostym przypadku operacji INSERT (50000 rekordów), a w przypadku średniej i wysokiej trudności zapytania ilość zajętej pamięci wynosiła prawie 1200 mb.

Operacja usuwania danych (DELETE) w najprostszym i najtrudniejszym przypadku dla MySQL wykorzystywała nieco ponad 20 mb, a ta sama operacja

średniej trudności wykorzystywała ponad 1200 mb pamięci. Zostało to spowodowane dużą ilością danych, które spełniały warunki zapytania i musiały zostać zachowane, a nie samym usuwaniem danych. Pamięć dla PostgreSQL i Clickhouse dla powyższego zapytania była taka sama niezależnie od jego skomplikowania.

Takie same wnioski jak dla operacji usuwania można zastosować przy operacji edycji danych (UPDATE).

Operacja pobierania danych (SELECT) dla MySQL niezależnie od poziomu trudności nie zmieniała ilości wykorzystywania pamięci, która była bardzo niska (~20 mb). Dla PostgreSQL i Clickhouse najprostsza operacja pobierała najwięcej pamięci (~160 mb), a średniej i wysokiej trudności operacje pobierały bardzo niskie wartości pamięci, zbliżone do wyników MySQL (~20 mb).

Silniki NoSQL przez większość testów utrzymywały podobne poziomy wykorzystania pamięci, niekiedy nawet identyczne.

Jedyne dostrzegalne różnice pojawiły się przy operacji pobierania danych (SELECT) gdzie wraz ze wzrostem ilości rekordów rosła też ilość pamięci wykorzystywanej przez ArangoDB. Różnice te wynosiły niekiedy nawet 250-cio krotność pamięci MongoDB.

Drugim przypadkiem różnicy, tym razem na korzyść ArangoDB, było wstawianie dużej ilości danych (250000 rekordów), gdzie MongoDB wykorzystywał dwukrotność pamięci (~920 mb) wykorzystanej przez ArangoDB (~430 mb).