**Obraz zawierający tekst, Czcionka, logo, wizytówka

Opis wygenerowany automatycznie**

**Mateusz Bocak i Gabriela Bieniek**

**Nr albumu studentów: 125104, 125101**

**Porównanie wydajności silników bazodanowych: MariaDB vs MongoDB**

**Praca projektowa nr 1 z IMEBD**

Prowadzący: dr inż. Piotr Grochowalski

Rzeszów 2024

Spis treści

[Cel badania 2](#_Toc182951732)

[Silniki 2](#_Toc182951733)

[Opis zbiorów 3](#_Toc182951734)

[Opis kwerend 3](#_Toc182951735)

[Procedura importu danych 3](#_Toc182951736)

[MariaDB 3](#_Toc182951737)

[MongoDB 4](#_Toc182951738)

[Wnioski 5](#_Toc182951739)

[Operacje na zbiorach 5](#_Toc182951740)

[MariaDB 5](#_Toc182951741)

[MongoDB 5](#_Toc182951742)

[Inne zastosowania baz 5](#_Toc182951743)

[Kroki do odtworzenia 5](#_Toc182951744)

[Repozytorium Projektu](https://github.com/JustFiesta/DB-engine-comparison)

# Cel projektu

Celem jest eksperyment porównawczy wydajność obu silników w zakresie:

* Czasu wykonania zapytań
* Zużycia zasobów systemowych (RAM, CPU, Dysk)

# Silniki

Projekt ma na celu porównanie wydajności dwóch różnych typów silników bazodanowych:

1. **MariaDB** - relacyjna baza danych SQL, oparta na strukturze tabelarycznej
2. **MongoDB** - nierelacyjna baza danych dokumentowa, opierająca się na kolekcjach dokumentów JSON

W obu przypadkach zastosowano optymalizacje systemów bazodanowych:

* MariaDB – utworzenie indeksów w tabelach
* MongoDB – optymalizacja zapytań pod kątem wydajności (możliwości są niemal nieograniczone).

# Opis zbiorów

W projekcie wykorzystano następujące zbiory danych:

* **Wizyty lekarskie** ([link](https://github.com/JustFiesta/DB-engine-comparison/tree/main/Generator)) - generowany poprzez skrypt Python (1,000,000 lekarzy, 2,500,000 pacjentów, 6,500,000 wizyt)
* **Loty komercyjne linii lotniczych w 2015 r.** ([link](https://mavenanalytics.io/data-playground?order=date_added%2Cdesc&search=airline%20flight)) - zestawione na potrzeby raportu U.S. DOT Air Travel Consumer Report, ok. 6 000 000 rekordów
* **Podróże rowerowe** ([link](https://s3.amazonaws.com/tripdata/index.html)) - ok. 1 000 000 rekordów

# Opis kwerend

Użyto różnych typów kwerend, do badania wyników w przypadku coraz to bardziej złożonych zapytań. Wykorzystano:

* Proste zapytania wyszukujące

Zapytanie pobiera identyfikator lotu, numer lotu i opóźnienie przylotu w przypadku lotów z opóźnieniami przekraczającymi 60 minut.

MariaDB

SELECT FLIGHT\_ID, FLIGHT\_NUMBER, ARRIVAL\_DELAY

    FROM Flights

    WHERE ARRIVAL\_DELAY > 60;

MongoDB

{

'collection': 'Flights',

   'query': {"ARRIVAL\_DELAY": {"$gt": 60}},

   'projection': {"\_id": 0, "FLIGHT\_NUMBER": 1, "ARRIVAL\_DELAY": 1}

}

* Zapytania agregujące

Zapytanie grupuje rekordy według roku urodzenia i oblicza całkowity czas trwania podróży dla każdego rocznika.

MariaDB

SELECT birth\_year, SUM(tripduration) AS total\_tripduration

FROM TripUsers GROUP BY birth\_year;

MongoDB

    {

        'collection': 'TripUsers',

        'pipeline': [

            {

                "$group": {

                    "\_id": "$birth\_year",

                    "total\_tripduration": {"$sum": "$tripduration"}

                }

            },

            {

                "$project": {

                    "birth\_year": "$\_id",

                    "total\_tripduration": 1,

                    "\_id": 0

                }

            },

            {

                "$sort": {"birth\_year": 1}

            },

            {

                "$match": {

                    "birth\_year": {"$ne": None}

                }

            }

        ]

    }

* Operacje JOIN/łączenia dokumentów

Zapytanie pobiera szczegóły podróży dłuższych niż 20 minut, w tym nazwy stacji początkowej i końcowej, łącząc tabelę TripUsers z tabelą Stations

MariaDB

SELECT

        t.trip\_id, t.tripduration, s.station\_name AS

start\_stat\_name, e.station\_name AS end\_stat\_name

    FROM TripUsers t

    JOIN Stations s ON t.start\_station\_id = s.station\_id

    JOIN Stations e ON t.end\_station\_id = e.station\_id

    WHERE t.tripduration > 20;

MongoDB

    {

        'collection': 'TripUsers',

        'pipeline': [

            {

                "$match": {

                    "tripduration": {"$gt": 20}

                }

            },

            {

                "$lookup": {

                    "from": "Stations",

                    "localField": "start\_station\_id",

                    "foreignField": "station\_id",

                    "as": "start\_station"

                }

            },

            {

                "$lookup": {

                    "from": "Stations",

                    "localField": "end\_station\_id",

                    "foreignField": "station\_id",

                    "as": "end\_station"

                }

            },

            {

                "$unwind": "$start\_station"

            },

            {

                "$unwind": "$end\_station"

            },

            {

                "$project": {

                    "trip\_id": 1,

                    "tripduration": 1,

                    "start\_station\_name": "$start\_station.station\_name",

                    "end\_station\_name": "$end\_station.station\_name",

                    "\_id": 0

                }

            },

            {

                "$sort": {"trip\_id": 1}

            },

        ]

    }

* Złożone zapytania analityczne

Zapytanie wyświetla listę unikalnych lotnisk (w tym ich miasta i stan), które obsługują loty na dystansie dłuższym niż średnia.

MariaDB

SELECT

        DISTINCT a.AIRPORT,

        a.CITY,

        a.STATE

    FROM Airports a

    JOIN Flights f ON a.IATA\_CODE = f.ORIGIN\_AIRPORT

    WHERE f.DISTANCE > (

        SELECT AVG(DISTANCE)

        FROM Flights

    )

    ORDER BY a.STATE, a.CITY;

MongoDB

{

        'collection': 'Flights',

        'pipeline': [

            {

                "$facet": {

                    "totalAvgDelay": [

                        {

                            "$group": {

                                "\_id": 1,

                                "avg": { "$avg": "$ARRIVAL\_DELAY" }

                            }

                        }

                    ],

                    "airlineDelays": [

                        {

                            "$group": {

                                "\_id": "$AIRLINE",

                                "avgDelay": { "$avg": "$ARRIVAL\_DELAY" }

                            }

                        },

                        {

                            "$lookup": {

                                "from": "Airlines",

                                "localField": "\_id",

                                "foreignField": "IATA\_CODE",

                                "as": "airline\_info"

                            }

                        },

                        {

                            "$unwind": "$airline\_info"

                        }

                    ]

                }

            },

            {

                "$unwind": "$totalAvgDelay"

            },

            {

                "$project": {

                    "results": {

                        "$filter": {

                            "input": "$airlineDelays",

                            "as": "airline",

                            "cond": { "$gt": ["$$airline.avgDelay", "$totalAvgDelay.avg"] }

                        }

                    }

                }

            },

            {

                "$unwind": "$results"

            },

            {

                "$project": {

                    "\_id": "$results.\_id",

                    "airlineName": "$results.airline\_info.AIRLINE",

                    "avgDelay": { "$round": ["$results.avgDelay", 2] }

                }

            },

            {

                "$sort": { "avgDelay": -1 }

            }

        ],

    }

# Procedura importu danych

Silniki różnią się procedurą importu danych oraz optymalizacją struktur/zapytań.

Dla MariaDB zastosowano indexy w tabelach, w przypadku MongoDB zastosowano optymalizacje kwerend.

Przedstawione kroki nawiązują do importu danych tylko do jednej tabeli/kolekcji. Analogiczne kroki należy powtórzyć dla pozostałych tabel.

Poniżej jest przestawienie procedur dla obu z nich.

## MariaDB

Procedura importu w mariadb wymaga zalogowania się za pomocą CLI *mysql*/*mariadb*.

SET foreign\_key\_checks = 0; *-- wyłączenie sprawdzania spójności do poprawy wydajności importowania*

LOAD DATA LOCAL INFILE 'path/to/dataset.csv' *-- ścieżka do datasetu*

INTO TABLE table\_name

FIELDS TERMINATED BY ',' *-- ustawienia delimiterów datasetu*

ENCLOSED BY '"'

LINES TERMINATED BY '\n'

IGNORE 1 ROWS;           *-- ignorowanie nagłówka datasetu*

SET foreign\_key\_checks = 1; *-- ponowne włączenie sprawdzania spójności kluczy*

Warto dodać, że przy procedurze importu dużych danych można skorzystać z dodatkowych opcji optymalizacji, np. dodając opcje do daemona mysql

sudo vim /etc/mysql/mariadb.cnf

*# tą część należy dodać do pliku*

    [mysqld]

    innodb\_buffer\_pool\_size = 15G

    innodb\_log\_buffer\_size = 1G

sudo systemctl daemon-reload

sudo systemctl restart mariadb

## MongoDB

W przypadku Mongo wystarczy utworzyć bazę danych i wykorzystać narzędzie *mongoimport*.

Use db\_name

mongoimport --db db\_name --collection collection\_name --type csv --headerline --file /path/to/dataset.csv

# Kroki do odtworzenia

1. Przygotowanie infrastruktury

Utworzenie maszyny wirtualnej AWS z Ubuntu 24.04 z konfiguracją poprzez skrypt bootstrap. W przypadku tego projektu została wykorzystana maszyna o charakterze generycznym (t3.xlarge – 8 rdzeni, 32GB RAM).

1. Pobranie danych do maszyny

Za pomocą narzędzia *scp* lub *wget* należy pobrać zbiory lotów i podróży rowerowych do maszyny wirtualnej.

Zbiór dla wizyt należy wygenerować na maszynie z repozytorium podlinkowanego wyżej.

1. Import danych do baz
2. Uruchomienie testów wydajnościowych

W repozytorium projektu znajduje się *wrapper* uruchamiający poszczególne testy. Z folderu Tests należy go uruchomić i poczekać na wyniki.

cd Tests/

chmod +x setup\_tests.sh

./setup\_tests.sh

Skrypt ten wyszukuje testy w subfolderze i wykonuje je pojedynczo, zapisując wyniki do pliku CSV. Na końcu importuje CSV do programu Excel.

1. Analiza wyników

Skrypt eksportuje wyniki do pliku xlsx, który należy wysłać za pomocą *scp* na interesującego nas hosta. Po pobraniu można interpretować wyniki w programie Excel za pomocą wykresów, tabel itp.

# Wnioski

## Zużycie zasobów

Podczas analizy wyników zużycia zasobów można zauważyć marginalne wahania, ale bez korzyści dla konkretnego silnika. Stąd można powiedzieć, że zużycie jest tożsame dla obu systemów, niezależnie od wielkości wykorzystywanej obecnie bazy danych.

## Czas wykonania kwerend

Co do czasów wykonania były ono mocno uzależnione od odpowiedniej optymalizacji po stronie MongoDB. Bazy dokumentowe są stworzone do przechowywania nieschematycznych danych i odbieranie ich w małych ilościach. Nie służą do raportowania.

MongoDB oferuje zaawansowany mechanizm agregacji, który pozwala na przetwarzanie danych w ramach zapytań w sposób optymalizowany pod kątem wydajności. Operacje takie jak grupowanie, filtrowanie, sortowanie są przeprowadzane w ramach potoku, co pozwala na bardziej elastyczne i wydajne manipulowanie danymi w porównaniu do SQL. Niestety nie potrafiliśmy wykorzystać tego potencjału w pełni.

Wynikiem są różne wahania w stosunku do MariaDB, zależne od ilości zwracanych dokumentów oraz ograniczeń przeszukiwania – to są momenty, w których MongoDB błyszczy.

Innymi słowy Mongo jest świetne do odzyskiwania specyficznych danych z nieustrukturyzowanych struktur. Warunkiem jest wczesna filtracja, indeksowanie odpowiednich pól w kolekcjach oraz ograniczenie zwracanych wyników.

### Mały zbiór danych

Proste zapytania zwracające dużo wyników lepiej obsługuje SQL. Każdy dodatkowo przeszukany dokument spowalnia znacznie spowalnia Mongo.

Agregacje na małych zbiorach wychodzą dobrze dla Mongo. Może to być spowodowane małym zbiorem do przeszukania.

Złożenia o dziwo również mongo wykonuje sprawniej.

Podzapytania niestety nie są mocną stroną mongo, ale tylko na małych zbiorach.

Rysunek 1 - mały zbiór danych

### Średni zbiór danych

Na średnim zbiorze widać potencjał optymalizacyjny zapytań w Mongo. Nie da się jednoznacznie określić, w tym wypadku, która baza jest „lepsza”. Odpowiednie zasotowanie mechanizmów optymalizacji mongo umożliwi znaczne przyspieszenie i wyprzedzenie SQL’owego odpowiednika, nawet w joinach.

Proste zapytania zwracające duże ilości odpowiedzi i wymagają przeszukania dużej ilości danych nie są mocną stroną Mongo.

Agregacje są mocno zależne od wielkości zbiorów wejściowych. MariaDB jest stabilna w ich wykonywaniu, nie ma mocnych wahań.

Złożenia i podzapytania zaskakują. Mongo w odpowiedniej konfiguracji jest dużo szybsze.

Rysunek 2 - średni zbiór danych

### Duży zbiór danych

Tutaj można zauważyć ogromne dysproporcje i możliwości Mongo. Wymagają one jednak odmiennego podejścia niż SQL’owy odpowiednik.

Na dużych zbiorach proste zapytania zwracające ogrom danych oraz zapytania łączące są zawsze szybsze niż odpowiedniki w Mongo (ale w przypadku ograniczenia przeszukiwania do jednego pola Mongo jest już szybsze – query 2 i query 3).

Agregacja i podzapytania zależą od umiejętności tworzenia zapytań do silnika MongoDB.

Złączenia tabel są na korzyść MariaDB, w końcu relacje ułatwiają przeszukiwanie struktur.

Rysunek 3 - duży zbiór danych

## Prędkość zapisu

Mongo góruje nad Marią w kontekście prędkości zapisu/importu danych. Spowodowane to jest prostszą strukturą.

Zapis dokumentu w MongoDB jest szybki, ponieważ dane są zapisywane w jednym obiekcie (dokument). Dodatkowo, MongoDB wspiera mechanizm „write concern”, który umożliwia kontrolowanie poziomu potwierdzeń zapisu, co może wpływać na wydajność zapisu.

Bazy SQL wymagają większego narzutu przy operacjach zapisu, zwłaszcza w przypadku używania transakcji i zachowania integralności referencyjnej.

## Podsumowanie

Wydajność MongoDB jest uwarunkowana schematem danych. Gdy dane są ułożone w struktury MariaDB będzie lepszym wyborem, jest powtarzalna i zawsze zwróci wynik.

Jednakże optymalizacja zapytań w Mongo i ograniczenie przeszukiwanych dokumentów wstępną filtracją może przyczynić się do prześcignięcia Marii. Szukałbym jej zastosowania do danych, gdzie struktury nie są jasne, w tym wypadku łatwiej byłoby pokazać zalety przeszukiwania dokumentów vs struktur.

Dodatkowo mongo wspiera mechanizmy wysokiej dostępności i łatwej replikacji, jest to rozwiązanie All-in-one w porównaniu do dodatkowych skomplikowanych konfiguracji baz SQL.

MongoDB jest zoptymalizowane pod kątem operacji, które obejmują duże zbiory danych w formacie dokumentów, szczególnie w przypadkach, gdzie operacje są niezależne od relacji między tabelami. Dobre wyniki uzyskuje się również przy operacjach agregacyjnych na dużych zbiorach dokumentów.

W SQL, szczególnie w przypadku zapytań łączących wiele tabel za pomocą JOIN, zapytania mogą być bardziej złożone i czasochłonne. Bazy SQL lepiej radzą sobie w przypadku złożonych operacji na relacjach między tabelami.

MongoDB jest lepszym wyborem dla aplikacji, które wymagają elastyczności, skalowalności, pracy z dużymi zbiorami danych i nieliniowymi strukturami danych, szczególnie tam, gdzie operacje zapisu i agregacji są intensywne i szybkie.

Bazy SQL są lepsze w przypadkach, gdy dane są silnie zorganizowane w relacjach, wymagają pełnej transakcyjności ACID oraz są poddawane skomplikowanym zapytaniom z wieloma operacjami JOIN.