JOURNAL
COMPUTER SCIENCES INSTITUTE

JCSI 34 (2025) 44–53
Received: 11 October 2024
Accepted: 23 November 2024

Comparative analysis of the performance of relational and non-relational databases in applications implemented in C#

Analiza porównawcza wydajności relacyjnych i nierelacyjnych baz danych w aplikacjach zaimplementowanych w języku C#

Patryk Baliński*, Łukasz Chudy*, Maria Skublewska-Paszkowska

Department of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract
The article focuses on comparing relational and non-relational databases using applications written in C#. The aim of the study is to understand in which cases relational databases are preferred and when it is worth considering the use of non-relational databases. The research examines the speed of data retrieval, updating, and deletion, in the context of five different databases, including relational ones like PostgreSQL, MySql, Oracle, and non-relational ones such as Neo4j and MongoDB. The data consists of 1,578,098 records. In the case of relational databases, a unified database model was applied, while in NoSQL databases, the data model was appropriately adjusted to the specific type of non-relational database. Differences in the execution time of database queries are analyzed based on indexing strategies and query complexity. Special attention is given to query performance efficiency in the context of using C# and libraries that facilitate the connection between the application and databases. The conducted research indicates that the PostgreSQL database achieves the lowest average query response times.

Keywords: relational databases; non-relational databases; performance analysis; performance comparison; C#

Streszczenie
Artykuł dotyczy porównania relacyjnych i nierelacyjnych baz danych, z użyciem aplikacji napisanych w języku C#. Praca ma na celu zrozumienie, w jakich przypadkach preferowane są bazy relacyjne, a kiedy warto rozważyć zastosowanie baz nierelacyjnych. Badania dotyczą szybkości pobierania, aktualizacji i usuwania danych, w kontekście 5 różnych baz danych, w tym relacyjnych takich jak PostgreSQL, MySql, Oracle, oraz nierelacyjnych Neo4j i MongoDB. Dane obejmują 1 578 098 rekordów. W przypadku baz danych relacyjnych zastosowano jednolity model bazodanowy, natomiast w bazach danych nierelacyjnych model danych został odpowiednio dostosowany do specyfiki danego typu bazy nierelacyjnej. Analizowane są różnice w szybkości wykonywania poleceń bazodanowych w zależności od strategii indeksowania, czy złożoności zapytań. Szczególną uwagę zwraca się na efektywność szybkości zapytań w kontekście użycia języka C# i bibliotek służących do połączenia pomiędzy aplikacją a bazami danych. Z przeprowadzonych badań wynika, że najmniejsze średnie czasy zapytań osiąga baza danych PostgreSQL.

Słowa kluczowe: relacyjne bazy danych; nierelacyjne bazy danych; analiza wydajności; porównanie wydajności; C#

*Corresponding authors
Email address: patryk.balinski@pollub.edu.pl (P. Baliński), lukasz.chudy@pollub.edu.pl (Ł. Chudy)

1. Wstęp

Środowisko informatyczne podlega dynamicznemu rozwojowi, co wymaga nie tylko zaawansowanych funkcjonalności, ale także wydajnego systemu zarządzania danymi. Determinuje to wybór adekwatnej technologii bazodanowej. W systemach informatycznych popularnym rozwiązaniem do przechowywania i zarządzania danymi jest użycie relacyjnych baz danych (Relational Database Management System, RDBMS). Ich struktura oparta na tabelach, relacjach pomiędzy nimi i transakcjach, zapewnia stabilność i integralność danych [1], co jest kluczowe w aplikacjach, gdzie wymagane jest precyzyjne organizowanie informacji.

Wraz z ewolucją wymagań aplikacyjnych, zauważalny stał się rosnący nacisk na elastyczność, skalowalność i wydajność systemów przechowujących i przetwarzających dane [2]. W odpowiedzi na te wyzwania, nierelacyjne bazy danych (NoSQL) zaczęły zdobywać na popularności, oferując alternatywne podejście do gromadzenia danych [3, 4]. Wśród środowisk związanych z big data, aplikacjami internetowymi, czy systemami obsługującymi duże liczby użytkowników, nierelacyjne bazy danych przynosiły korzyści, w postaci elastyczności struktury bazy danych, łatwej skalowalności poziomej w przypadku wzrostu obciążenia, obsługi dużej liczby zapytań równocześnie [5], czy szybkości operacji na dużych zbiorach danych.

Biorąc pod uwagę te zmiany, istnieje potrzeba dogłębnego zrozumienia i oceny korzyści i ograniczeń wynikających z zastosowania obu rodzajów baz danych, w zróżnicowanych scenariuszach zastosowań. Analiza porównawcza szybkości operacji relacyjnych i nierelacyjnych baz danych w kontekście aplikacji napisanych w języku C# staje się istotnym zagadnieniem z uwagi na wysoką popularność tego języka programowania w branży i jego rolę w ekosystemie technologii Microsoft.

Celem niniejszego artykułu jest przeprowadzenie szczegółowego porównania wydajności relacyjnych i nierelacyjnych baz danych w aplikacjach napisanych w języku programowania C#, w kontekście operacji odczytu, modyfikacji i usuwania danych. Analiza ta obejmuje szybkość operacji, z uwzględnieniem strategii indeksowania, w różnych scenariuszach.

Na potrzeby pracy zdefiniowano następujące hipotezy badawcze:

H1: *Aplikacje zaimplementowane w języku C# wykorzystujące nierelacyjne bazy danych (NoSQL) będą charakteryzować się lepszą wydajnością pod względem szybkości przetwarzania zapytań w przypadku prostych zapytań.*

H2: *Relacyjne bazy danych (SQL) w aplikacjach napisanych w języku C# zapewnią lepszą wydajność czasową podczas obsługi złożonych zapytań wymagających operacji na ustrukturyzowanych danych, w porównaniu do nierelacyjnych baz danych (NoSQL).*

H3: *Zastosowanie indeksacji baz danych poprawi wydajność czasową wykonywania zapytań zarówno w bazach danych relacyjnych jak i nierelacyjnych.*

2. Wprowadzenie do technologii

2.1. Relacyjne bazy danych

Model relacyjny opiera się na koncepcji normalizacji, której głównym celem jest zlikwidowanie redundancji danych i utrzymanie integralności. Normalizacja polega na organizowaniu danych w taki sposób, aby nie było duplikacji informacji, czego skutkiem jest oszczędność miejsca. Pomaga ona również w utrzymaniu spójnej i jednolitej bazy w całym systemie. Kluczową rolę odgrywają relacje, które umożliwiają operacje łączenia kilku tabel i przeprowadzanie skomplikowanych zapytań. Relacyjne bazy danych umożliwiają także wykorzystanie transakcji, które gwarantują atomowość, spójność, trwałość i izolację (ACID - atomicity, consistency, isolation, durability). Wprowadzenie transakcji zapewnia, że operacje na bazie są wykonane w całości, albo w ogóle nie są realizowane, co jest kluczowe do zapewnienia integralności danych w przypadku awarii systemu.

Warto zaznaczyć, że język zapytań SQL (Structured Query Language) jest integralną częścią ekosystemu relacyjnych baz danych. Operacje CRUD (Create, Read, Update, Delete) udostępnione przez ten język stanowią kompleksowe narzędzie do zarządzania danymi [6].

W procesie dostosowywania się do nowych wyzwań, relacyjne bazy danych nadal pozostają niezwykle ważnym narzędziem w projektowaniu systemów informatycznych. Ich mocne strony, takie jak klarowna struktura, możliwość stosowania kompleksowych zapytań, oraz bezpieczeństwo wynikające z mechanizmu transakcji, sprawiają, że są one nadal preferowane ponad bazami nierelacyjnymi w wielu scenariuszach aplikacyjnych [7].

2.2. Nierelacyjne bazy danych

Atutem nierelacyjnych baz danych jest elastyczność struktury [8]. Pozwalają one na przechowywanie danych w formie nieustrukturyzowanej, półustrukturyzowanej lub ustrukturyzowanej. W przeciwieństwie do relacyjnych baz danych, nie są ograniczone sztywnymi schematami, co ułatwia pracę w środowisku z dynamicznie zmieniającymi się wymaganiami aplikacyjnymi. Umożliwia to efektywne przechowywanie i przetwarzanie danych o zmiennej strukturze.

Nierelacyjne bazy danych występują w wariantach obsługujących różnorodne struktury danych, takie jak dokumenty, grafy czy dane klucz-wartość [9]. To je czyni uniwersalnym narzędziem, umożliwiającym łatwość dostosowania do różnych scenariuszy aplikacyjnych.

Wynika z tego, że nierelacyjne bazy danych stanowią istoty kierunek w dziedzinie przetwarzania i przechowywania danych, w dynamicznych środowiskach aplikacyjnych, bądź w przypadku pracy z ogromną ilością informacji. Elastyczność i skalowalność czynią je wartościowym wyborem w przypadku nowoczesnych projektów informatycznych, które wymagają szybkości i adaptacyjności w zarządzaniu danymi [10].

2.3. Język C#

W kontekście dziedziny programowania aplikacji bazodanowych, język C# (C-Sharp) stanowi popularny wybór [11, 12]. Jest to obiektowy język programowania, opracowany przez Microsoft, jako część platformy .NET. Charakteryzuje się wydajnością i elastycznością, co pozwala na tworzenie różnorodnych aplikacji, w tym takich, które łączą się z różnymi bazami danych.

Udostępnia szeroki zakres funkcji do manipulacji na danych. Oferuje wiele bibliotek, dzięki którym w prosty sposób możliwe jest połączenie się z wybraną bazą danych. W przypadku relacyjnych baz danych, możliwe jest utworzenie jej struktury przy pomocy technologii migracji. Wykorzystując technologię ORM (Object-Relational Mapping) możliwe jest utworzenie odpowiednich obiektów, odpowiadających strukturze relacyjnej bazy danych, co pozwala na operacje CRUD, poprzez manipulację na obiektach.

W kontekście analizy porównawczej wydajności relacyjnych i nierelacyjnych baz danych w aplikacjach napisanych w języku C#, istnieje potrzeba zrozumienia sposobu, w jaki ten język współpracuje z różnymi typami baz danych. C# zapewnia interfejsy programistyczne (API) do zarządzania połączeniami z bazami danych, wykonywania zapytań i przetwarzania wyników. To umożliwia programistom integrację aplikacji napisanych w C# z różnymi systemami baz danych, w tym zarówno relacyjnymi, jak i nierelacyjnymi.

3. Przegląd literatury

Współczesne bazy danych można podzielić na dwie główne kategorie: relacyjne (SQL) oraz nierelacyjne (NoSQL). W kontekście wydajności, szczególnie w przypadku aplikacji o dużych wymaganiach w zakresie przetwarzania danych, wybór odpowiedniego typu bazy danych może mieć kluczowe znaczenie.

Relacyjne bazy danych są od dekad standardem w zarządzaniu danymi w dużych aplikacjach biznesowych, charakteryzując się wysoką integralnością danych i możliwością złożonego modelowania relacji. W literaturze można znaleźć wiele analiz wydajności relacyjnych baz danych w różnych kontekstach [13]. Jednym z przykładów takich artykułów jest [14]. Autorzy zbadali wydajność relacyjnych baz danych na przykładzie bazy danych Oracle oraz MSSQL w kontekście aplikacji desktopowej. Artykuł potwierdził hipotezę, że baza danych MSSQL wykazuje szybsze wykonywanie zapytań INSERT i UPDATE w porównaniu do bazy danych Oracle.

Nierelacyjne bazy danych, często określane jako NoSQL, zdobyły na popularności głównie ze względu na swoją elastyczność oraz zdolność do obsługi dużych, nieustrukturyzowanych zbiorów danych. W kontekście aplikacji internetowych i systemów przetwarzania dużych wolumenów danych, NoSQL oferuje podejście bardziej skalowalne, ale kosztem pewnych kompromisów, takich jak bezpieczeństwo danych [15]. Artykuł [16] poświęcony jest przeglądowi baz danych NoSQL i nierelacyjnych systemów zarządzania bazami danych. Dzięki przeanalizowaniu przyczyn gwałtownego wzrostu popularności nierelacyjnych baz danych, autorzy wyjaśnili, jak mogą one zwiększyć produktywność firm.

W literaturze istnieje wiele badań porównujących wydajność baz danych SQL i NoSQL [17]. W artykule [18] autorzy porównują oba modele pod kątem ich zastosowania w aplikacjach internetowych, wskazując, że bazy NoSQL lepiej radzą sobie z dynamicznie zmieniającymi się zbiorami danych i wymaganiami dotyczącymi skalowalności. Z drugiej strony, systemy SQL oferują bardziej stabilną strukturę danych i lepsze wsparcie dla skomplikowanych zapytań.

Kolejnym przykładem artykułu skupiającym się na porównaniu relacyjnych i nierelacyjnych baz danych jest [19]. Autorzy, opierając się na istniejącej literaturze, wskazali, że relacyjne bazy danych najlepiej sprawdzają się w przypadku przetwarzania niewielkiej ilości uporządkowanych danych, natomiast bazy NoSQL zostały zaprojektowane z myślą o skalowalności i wydajności, choć oferują jedynie ograniczony poziom bezpieczeństwa oraz używają niestandardowych języków zapytań.

W środowisku naukowym pojawiają się również propozycje architektury bazodanowej wykorzystującej zarówno model relacyjny jak i nierelacyjny. Jednym z przykładów jest artykuł [20]. Autorzy zaprezentowali architekturę łączącą zalety obydwu rodzajów baz danych poprzez ich integrację jako jednolitego systemu, umożliwiającego wykonywanie operacji bazodanowych za pomocą języka SQL. Celem zaprezentowanej architektury było uproszczenie korzystania z hybrydowych baz, co może okazać się istotne w kontekście współczesnych potrzeb związanych z dynamicznie zmieniającym się środowiskiem informatycznym.

Z przeglądu literatury wynika, że wybór pomiędzy relacyjnymi a nierelacyjnymi bazami danych powinien być uzależniony od konkretnego zastosowania i wymagań aplikacji. Relacyjne bazy danych oferują bogate wsparcie dla integralności danych i złożonych operacji, ale mają swoje ograniczenia w kontekście skalowalności. Z kolei systemy NoSQL, jak MongoDB, oferują większą elastyczność i lepsze wsparcie dla aplikacji, które muszą obsługiwać duże i nieustrukturyzowane zbiory danych. Wydajność obu typów baz danych zależy od specyfiki aplikacji, co czyni analizę porównawczą tych systemów szczególnie istotną w kontekście implementacji w języku C#.

4. Metoda badawcza

4.1. Stanowisko badawcze

Do utworzenia kontenerów z bazami danych zostały zastosowano następujące obrazy Docker:

- „postgres” z wersją PostgreSQL 16.2;
- „gvenzl/oracle-xe” z wersją Oracle 21.3.0.0.0;
- „neo4j” z wersją Neo4j 5.23.0;
- „mysql” z wersją MySql 9.0.1;
- „mongo” z wersją MongoDB 7.0.12.

Aplikacja C# została napisana w .NET 8.0. Do obsługi połączeń z bazami danych użyto bibliotek:

- MongoDB.EntityFrameworkCore w wersji 8.1.1;
- Neo4jClient w wersji 5.1.16;
- Npgsql.EntityFrameworkCore.PostgreSQL w wersji 8.0.8;
- Oracle.EntityFrameworkCore w wersji 8.23.50;
- Pomelo.EntityFrameworkCore.MySql w wersji 8.0.2.

Specyfikacja komputera wykorzystanego do testów:

- procesor Intel Core i5-11400H 2.70GHz;
- 32 GB RAM DDR4 3200 ;
- dysk SSD M.2 2280 512GB NVMe ESR512GTLG-E6GBTNB4;
- system operacyjny Windows 10 64 bit.

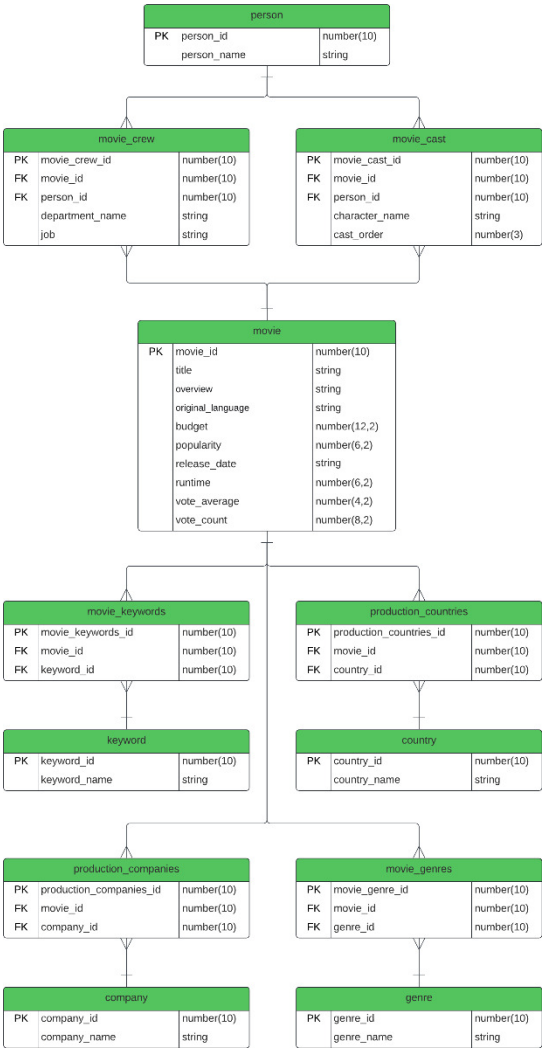
4.2. Przygotowanie zbioru danych

W procesie przygotowania zbioru danych do analizy porównawczej w ramach metody badawczej, kluczowym etapem było znalezienie odpowiedniego zbioru danych o znacznej liczbie rekordów, który stanowiłby reprezentatywny materiał do przeprowadzenia badań. Wybór skoncentrował się na zbiorze danych, który nie tylko charakteryzuje się dużą liczbą rekordów, ale również odzwierciedla różnorodność struktury i typów danych. Zadaniem było znalezienie zbioru, który pozwoliłby na rzetelne porównanie wydajności zarówno relacyjnych, jak i nierelacyjnych baz danych w kontekście obsługi dużej ilości informacji.

Zdecydowano się na zbiór dotyczący filmów, posiadający 1578098 rekordów, z czego 45433 rekordów dotyczących filmów [21]. Zbiór ten zawiera szczegółowe informacje o filmach, w tym zarówno dane tekstowe, jak opisy fabuły i tytuły, jak i dane liczbowe, takie jak budżet, popularność czy oceny użytkowników. Dodatkowo, zbiór obejmuje relacyjne powiązania z

innymi elementami, takimi jak gatunki filmowe, obsada, ekipa filmowa, kraje produkcji, firmy produkcyjne oraz powiązane słowa kluczowe, co czyni go idealnym do porównania efektywności relacyjnych i nierelacyjnych baz danych.

Kolejnym etapem było dostosowanie zbioru do wymagań zarówno relacyjnych, jak i nierelacyjnych baz danych. Dla baz relacyjnych dane zostały znormalizowane, a relacje między tabelami, np. między filmami a obsadą, zostały odwzorowane za pomocą kluczy głównych i obcych (Rysunek 1).

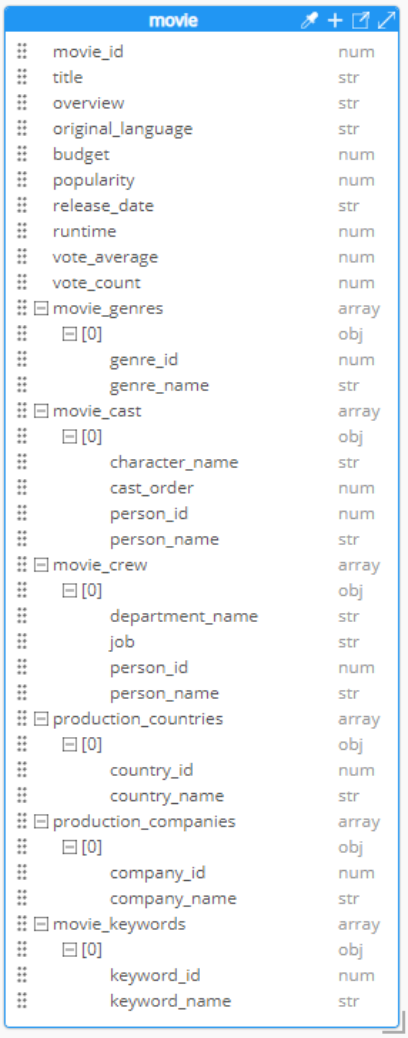


Rysunek 1: Schemat modelu bazy danych relacyjnej dotyczący filmów.

W przypadku bazy danych MongoDB, dane dotyczące pojedynczego filmu zostały przedstawione w postaci pojedynczego dokumentu, a relacje zostały zamienione na zagnieżdżone dane (Rysunek 2).

Natomiast w bazie danych Neo4j, encje słabe z baz relacyjnych, zostały zamienione na relacje z odpowiednimi właściwościami (Rysunek 3).

Ten etap przygotowania danych zapewnił solidną podstawę do przeprowadzenia analizy wydajności różnych rodzajów baz danych, umożliwiając rzetelne porównanie w kontekście rzeczywistych, złożonych danych filmowych.

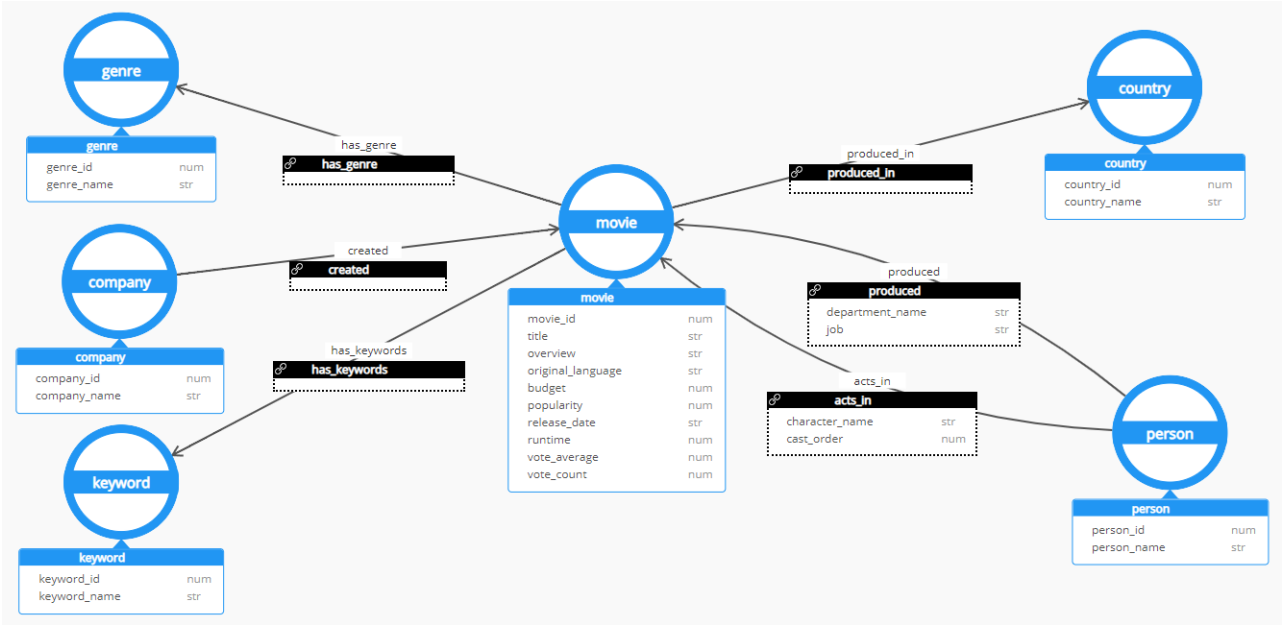


Rysunek 2: Schemat modelu bazy danych dokumentowej dotyczący filmów.

4.3. Opracowanie aplikacji do porównania systemów bazodanowych

Kluczowym etapem było opracowanie aplikacji w języku C#, która umożliwiałaby połączenie się z wybranymi bazami danych relacyjnymi i nierelacyjnymi, mając na celu przeprowadzenie szczegółowych testów porównawczych. Implementacja tej aplikacji miała na celu stworzenie uniwersalnego narzędzia, zdolnego do obsługi różnych baz danych, co pozwalałoby na obiektywne porównanie ich wydajności w zróżnicowanych scenariuszach zastosowań.

W pierwszym kroku, skoncentrowano się na napisaniu aplikacji w języku C#, która umożliwiałaby połączenie z bazami danych relacyjnymi, takimi jak Oracle [22], MySql [23], PostgreSQL [24], oraz nierelacyjnymi: MongoDB [25] i Neo4j [26]. Implementacja ta uwzględniała specyficzne cechy każdej bazy danych, takie jak różnice w językach zapytań czy strukturze danych, co miało na celu stworzenie spójnego środowiska testowego dla porównań wydajnościowych.



Rysunek 3: Schemat modelu bazy danych grafowej dotyczący filmów.

4.4. Analizowane polecenia bazodanowe

Pomiary czasów zostały zrealizowane przy pomocy klasy „Stopwatch", która służy do dokładnego pomiaru czasu. Czasy wykonywania operacji bazodanowych nie obejmowały połączenia z bazą danych, a pomiar zaczynał się przed wysłaniem polecenia. W przypadku zapytań typu SELECT, pomiar trwał do czasu zwrócenia ilości wybranych rekordów. Zapytania typu UPDATE oraz DELETE kończyły pomiar przed wykonaniem operacji „Rollback", z wyjątkiem bazy danych MongoDB, w której taka operacja nie jest wspierana. Niezależnie od bazy danych, zapytania były realizowane w możliwie zbliżony sposób oraz same bazy danych posiadały identyczne dane.

4.4.1.Proste zapytanie SELECT

Pierwsze zapytanie miało na celu wybór tytułów wszystkich filmów. Przetestowano wariant z użyciem podstawowych indeksów oraz w przypadku relacyjnych baz danych wariant z pełnym skanowaniem tabel. Listing 1 prezentuje zapytanie SQL w przypadku relacyjnych baz danych. Polecenie do bazy Neo4j znajduje się na Listingu 2, natomiast Listing 3 przedstawia zapytanie do bazy MongoDB.

Listing 1: Kod źródłowy obrazujący polecenie wyboru tytułów wszystkich filmów w języku SQL

```
SELECT m."Title" FROM movie m;
```

Listing 2: Kod źródłowy obrazujący polecenie wyboru tytułów wszystkich filmów do bazy Neo4j

```
MATCH (m:Movie)
RETURN m.title;
```

Listing 3: Kod źródłowy obrazujący polecenie wyboru tytułów wszystkich filmów do bazy MongoDB

```
db.movies.find({}, { title: 1 })
```

4.4.2.Zapytanie warunkowe SELECT

Drugim przetestowanym zapytaniem było zapytanie warunkowe. Jego celem był wybór filmów na podstawie słowa kluczowego „time travel". Przetestowano warianty z pełnym skanowaniem tabel oraz z dodatkowym indeksem na kolumnie z nazwą słowa kluczowego. Listing 4 przedstawia kod w języku SQL, na Listingu 5 widnieje zapytanie do bazy Neo4j, Listing 6 obrazuje polecenie do bazy danych MongoDB.

Listing 4: Kod źródłowy obrazujący polecenie wyboru filmów na podstawie słowa kluczowego w języku SQL

```
SELECT m."Title", ke."Name"
FROM movie m
  JOIN movie_keyword k ON m.movie_id = k.movie_id
  JOIN keyword ke ON ke.keyword_id = k.keyword_id
WHERE ke."Name" = 'time travel';
```

Listing 5: Kod źródłowy obrazujący polecenie wyboru filmów na podstawie słowa kluczowego do bazy Neo4j

```
MATCH (m:Movie)-[:HAS_KEYWORDS]->(k:Keyword {name:
"time travel"})
RETURN m.title, k.name;
```

Listing 6: Kod źródłowy obrazujący polecenie wyboru filmów na podstawie słowa kluczowego do bazy MongoDB

```
db.movies.find(
    { "movie_keywords.name": "time travel" },
    { title: 1, "movie_keywords.name": 1 }
);
```

4.4.3.Zapytanie agregujące SELECT

Jako trzecie zapytanie przetestowano zapytanie agregujące, którego zadaniem było utworzenie zestawienia producentów z liczbą wyprodukowanych przez nich filmów z lat 1990-1995 posortowane malejąco pod względem ilości wyprodukowanych filmów. Przetestowano zapytania z wyłączeniem indeksów, z indeksami podstawowymi oraz z dodatkowymi indeksami na kolumnie z datą wydania. Listing 7 prezentuje kod SQL testujący relacyjne bazy danych, na Listingu 8 widnieje polecenie wysłane do bazy danych Neo4j, natomiast Listing 9 obrazuje zapytanie do grafowej bazy danych MongoDB.

Listing 7: Kod źródłowy obrazujący polecenie utworzenia zestawienia producentów z liczbą wyprodukowanych przez nich filmów z lat 1990-1995 posortowane malejąco w języku SQL

```
SELECT c."Name" AS producer_name, COUNT(m.movie_id)
AS film_count
FROM movie m
  JOIN production_company pc ON pc.movie_id =
m.movie_id
  JOIN company c ON c.company_id = pc.company_id
WHERE m.release_date >= '1990-01-01' AND
m.release_date <= '1995-01-01'
GROUP BY c."Name"
ORDER BY film_count DESC;
```

Listing 8: Kod źródłowy obrazujący polecenie utworzenia zestawienia producentów z liczbą wyprodukowanych przez nich filmów z lat 1990-1995 posortowane malejąco do bazy Neo4j

```
MATCH (m:Movie)<-[:CREATED]-(c:Company)
WHERE m.release_date >= '1990-01-01' AND
m.release_date <= '1995-01-01'
WITH c, count(m) AS filmCount
RETURN c.name AS producerName, filmCount
ORDER BY filmCount DESC;
```

Listing 9: Kod źródłowy obrazujący polecenie utworzenia zestawienia producentów z liczbą wyprodukowanych przez nich filmów z lat 1990-1995 posortowane malejąco do bazy MongoDB

```
db.movies.aggregate([
    {
        $match: {
            release_date: {
                $gte: new ISODate("1990-01-
01T00:00:00Z"),
                $lt: new ISODate("1995-01-
01T00:00:00Z")
            }
        }
    },
    {
        $unwind: "$production_companies"
    },
    {
        $group: {
            _id: "$production_companies.name",
            count: { $sum: 1 }
        }
    },
    {
        $sort: { count: -1 }
    }
]);
```

4.4.4.Polecenie UPDATE

Kolejne zapytanie bazodanowe miało za zadanie aktualizację tytułów filmów, których data produkcji zawierała się w przedziale lat 1985 i 1990. Testy obejmowały zapytania z różnymi strategiami indeksowania, w tym z pełnym skanowaniem tabel, z indeksami podstawowymi oraz z dodatkowym indeksem na kolumnach daty wydania i tytułu filmu. Kod SQL widnieje na Listingu 10, zapytanie do bazy danych Neo4j przedstawiono na Listingu 11, a Listing 12 obrazuje zapytanie do bazy danych MongoDB.

Listing 10: Kod źródłowy obrazujący polecenie aktualizacji tytułu filmów z lat 1985-1990 w języku SQL

```
UPDATE movie m
SET "Title" = 'nowy-tytul' "
WHERE m.release_date >= '1985-01-01' AND
m.release_date <= '1990-01-01';
```

Listing 11: Kod źródłowy obrazujący polecenie aktualizacji tytułu filmów z lat 1985-1990 do bazy Neo4j

```
MATCH (m:Movie) WHERE m.release_date >= '1985-01-
01'
AND m.release_date <= '1990-01-01'
SET m.title = 'nowy-tytul';
```

Listing 12: Kod źródłowy obrazujący polecenie aktualizacji tytułu filmów z lat 1985-1990 do bazy MongoDB

```
db.collection.updateMany(
    {
        release_date: {
            $gte: new ISODate("1985-01-
01T00:00:00Z"),
            $lte: new ISODate("1990-01-
01T00:00:00Z")
        }
    },
    {
        $set: { title: "nowy-tytul" }
    }
);
```

4.4.5.Polecenie DELETE

Ostatnim testowanym poleceniem bazodanowym było polecenie usuwające rekordy. Przetestowano usunięcie wszystkich filmów z lat 1985-1990. Testy zostały wykonane w wariancie z indeksami podstawowymi. Kod SQL został przedstawiony na Listingu 13, zapytanie do bazy danych Neo4j widnieje na Listingu 14, natomiast Listing 15 obrazuje zapytanie do bazy danych MongoDB.

Listing 13: Kod źródłowy polecenia usuwającego filmy z lat 1985-1990 w języku SQL

```
DELETE FROM movie m
WHERE m.release_date >= '1985-01-01' AND
m.release_date <= '1990-01-01';
```

Listing 14: Kod źródłowy obrazujący polecenie aktualizacji tytułu filmów z lat 1985-1990 do bazy Neo4j

```
MATCH (m:Movie)
WHERE m.release_date >= '1985-01-01' AND
m.release_date <= '1990-01-01'
DETACH DELETE m;
```

Listing 15: Kod źródłowy obrazujący polecenie aktualizacji tytułu filmów z lat 1985-1990 do bazy MongoDB

```
db.collection.deleteMany({
    release_date: {
        $gte: new ISODate("1985-01-01T00:00:00Z"),
        $lte: new ISODate("1990-01-01T00:00:00Z")
    }
});
```

4.5. Opis eksperymentu

W badaniu przeanalizowano wydajność relacyjnych i nierelacyjnych baz danych w aplikacjach napisanych w języku C#. Testy obejmowały zapytania typu SELECT, UPDATE oraz DELETE. Każde zapytanie wykonano 10 razy, a następnie wyciągnięto średnią ze zmierzonych czasów. Kontener docker, zawierający bazę danych był restartowany przed każdym zapytaniem, dzięki czemu pamięć podręczna i plany zapytań były usuwane. W testach uwzględniono różne strategie indeksowania takie jak: brak indeksów, indeksy na kluczach głównych i obcych oraz dodatkowe indeksy na wybranych polach. Ważnym czynnikiem była także implementacja aplikacji w języku C#, oraz wybrane biblioteki służące do połączenia z bazami, co mogło wpłynąć na optymalizację wykonywania zapytań.

Przebieg pojedynczego testu:

1. Zbudowanie aplikacji testującej wybrane zapytanie bazodanowe.
2. Uruchomienie kontenera z bazą danych.
3. Pomiar czasu wykonania zapytania bazodanowego.
4. Wyłączenie kontenera, w celu usunięcia pamięci podręcznej i planów zapytań.

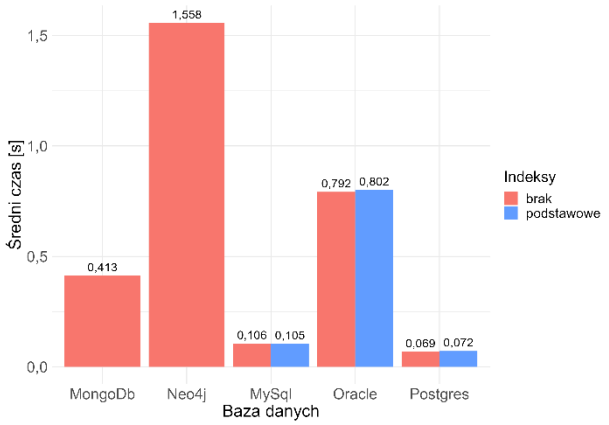
5. Wyniki Badań

Przeprowadzone badania skupiły się na pomiarze czasu wykonania poszczególnych poleceń bazodanowych. Uwzględniono zarówno przypadki, w których badana baza danych mogła skorzystać z indeksów w celu przyspieszenia operacji, jak i sytuacje, w których konieczne było wykonanie pełnego skanowania bazy danych bez użycia indeksów, co w niniejszym badaniu określono jako "brak indeksów". Dodatkowo, w kontekście tabel z indeksami rozróżniono dwa scenariusze: pierwszy, w którym wykorzystywane były indeksy domyślnie utworzone przez narzędzie ORM lub samą bazę danych (opisane jako "indeksy podstawowe"), oraz drugi, w którym zastosowano indeksy utworzone przez użytkownika (opisane jako "indeksy dodatkowe"). Analiza wyników pozwoliła na wyciągnięcie wniosków dotyczących wydajności badanych baz danych w różnych kontekstach użycia.

5.1. Wybór tytułów wszystkich filmów

Pierwsze zapytanie miało na celu wybór wszystkich tytułów filmów z tabeli (Listing 1, 6, 8). Wyniki testów pokazały, że baza PostgreSQL uzyskała najlepszy średni czas, co czyni ją najwydajniejszą w tym przypadku, z kolei najsłabszy wynik uzyskała baza Neo4j, co wskazuje, że hipoteza H1 jest nieprawdziwa.

W przypadku tak prostego zapytania, indeksy nie wnosiły żadnej znaczącej różnicy. Bazy relacyjne domyślnie zakładają indeksy na kolumnach będących kluczami głównymi lub obcymi. Baza MongoDB posiada indeks tylko na kluczu głównym, którego nie da się wykluczyć z zapytania, natomiast Neo4j nie tworzy automatycznie indeksów na węzłach czy relacjach.



Rysunek 4: Średnie czasy wykonania polecenia wybierającego tytuły wszystkich filmów.

5.2. Wybór filmów na podstawie słów kluczowych

Drugie zapytanie miało na celu wybór filmów na podstawie słowa kluczowego „time travel” (Listing 2, 6, 8). W tym przypadku zastosowanie indeksów miało istotny wpływ na czas wykonania.

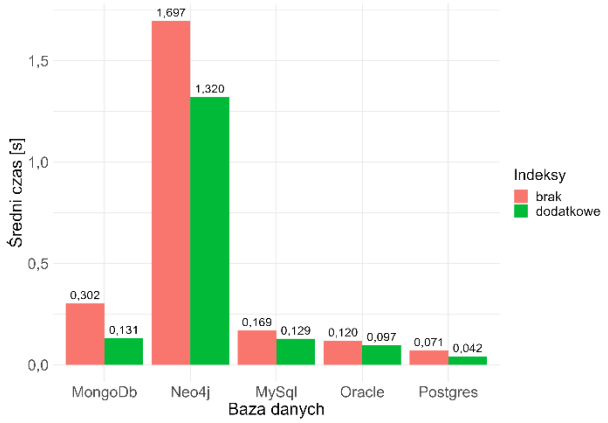
Indeksy na polu przedstawiającym nazwę słowa kluczowego znacząco zmniejszyły średni czas wykonywania zapytań do nierelacyjnych baz danych. Zauważalna jest również poprawa średnich czasów zapytań w przypadku relacyjnych baz danych, z uwagi na zastosowanie indeksów podczas wykonywania zapytań.

Wyniki zostały zaprezentowane na Rysunku 5.

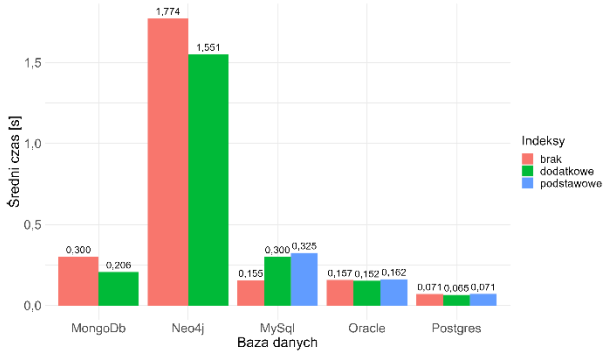
5.3. Zestawienie producentów z liczbą wyprodukowanych przez nich filmów z lat 1990–1995 posortowane malejąco

Dla większości testowanych baz danych, dodanie indeksu na pole daty wydania, znacząco przyśpieszyło szybkość zapytań. Wynik ten pokazuje znaczenie dobrze zoptymalizowanych indeksów w bazach danych, w szczególności w zapytaniach wymagających posortowania danych.

Szybsze okazały się relacyjne bazy danych, co potwierdza słuszność hipotezy H2. Wyniki dla tego zapytania zostały przedstawione na Rysunku 6. Kod źródłowy zaprezentowany jest na Listingach 3, 6 oraz 8.



Rysunek 5: Średnie czasy wykonania polecenia wybierającego producentów z liczbą wyprodukowanych filmów z lat 1990-1995.

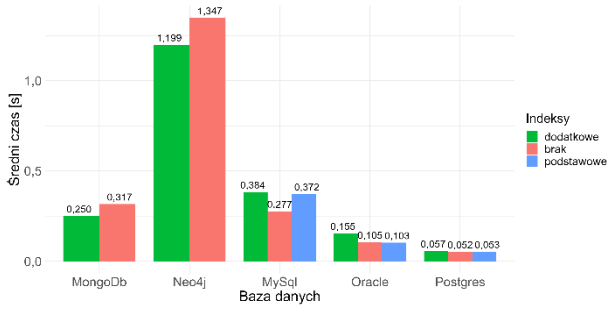


Rysunek 6: Średnie czasy wykonania polecenia tworzącego zestawienie producentów z liczbą wyprodukowanych filmów w latach 1990-1995.

5.4. Aktualizacja tytułów filmów z lat 1985 - 1990

Najlepsza okazała się baza PostgreSQL, jednak w przypadku użycia dodatkowych indeksów na polach daty wydania i tytułu filmu, średni czas operacji się zwiększył.

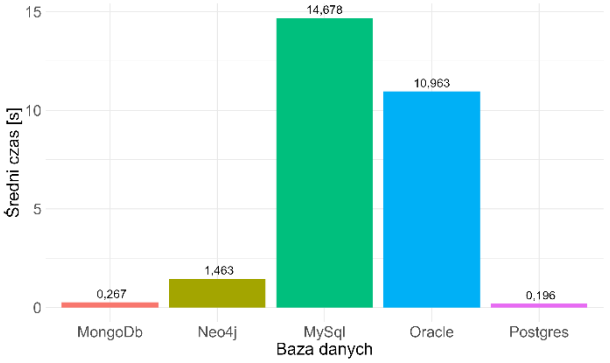
Bazy relacyjne najlepiej poradziły sobie w trybie pełnego skanowania tabel, bez użycia indeksów, natomiast użycie dodatkowych indeksów w bazach nierelacyjnych znacząco zmniejszyło średni czas wykonywania zapytań. Kod źródłowy zaprezentowany jest na Listingach 4, 9, 11. Szczegółowe wyniki przedstawiono na Rysunku 7.



Rysunek 7: Średnie czasy wykonania polecenia aktualizującego tytuły filmów z lat 1985-1990.

5.5. Usuwanie filmów z lat 1985 - 1990

W testach dotyczących usuwania rekordów z zastosowaniem polecenia DELETE (Listing 5, 9, 11) najszybszy okazał się PostgreSQL, następnie bazy nierelacyjne, podczas gdy Oracle oraz MySql osiągnęły znacznie gorsze wyniki. Wyniki dla testów usuwania zilustrowano na Rysunku 8.



Rysunek 8: Średnie czasy wykonania polecenia usuwającego filmy z lat 1985-1990.

6. Wnioski

Istnieje wiele czynników, które mogły wpływać na czasy zapytań. Należy wziąć pod uwagę, że zapytania były wysyłane z aplikacji C#, a połączenie było nawiązywane za pomocą odpowiednich bibliotek. Z racji, że przed każdym zapytaniem kontener był uruchamiany ponownie, każda z baz danych musiała stworzyć plan zapytania, załadować dane i indeksy do pamięci podręcznej, czy wykonać pewne operacje konfiguracyjne związanych z pierwszym zapytaniem do bazy. Najbardziej można to zauważyć na przykładzie bazy Neo4j, gdzie nawet najprostsze zapytania SELECT, osiągały bardzo długi czas wykonania, w porównaniu z innymi bazami.

W operacjach SELECT i UPDATE najlepiej wypadła baza danych PostgreSQL, mocno wyróżniając się na tle innych baz danych, zarówno z użyciem indeksów, jak i przy pełnym skanowaniu tabel. Bazy Oracle oraz MySql uzyskały podobne wyniki w bardziej skomplikowanych zapytaniach, jednak w najprostszym wariancie, lepszy wynik uzyskała baza MySql. Baza nierelacyjna MongoDB miała podobne czasy zapytań jak poprzednie dwie bazy. Najsłabiej wypadła baza Neo4j, która we wszystkich przypadkach była znacznie wolniejsza od pozostałych baz danych. Relacyjne bazy danych uzyskiwały mniejsze średnie czasy wykonywania poleceń SELECT, co dowodzi że hipoteza H1 nie jest prawdziwa. Użycie indeksów zauważalnie zmniejszyło czasy zapytań, co potwierdza słuszność hipotezy H3, w szczególności można to zauważyć w bazach nierelacyjnych.

W operacjach usuwających dane, nadal najlepsza była baza PostgreSQL, co oznacza, że jest dobrze zoptymalizowana pod kątem usuwania danych, jednak pozostałe bazy relacyjne były mniej wydajne w porównaniu do baz nierelacyjnych. W bazach relacyjnych może to wynikać z konieczności usuwania

zarówno indeksów na kluczach głównych i obcych, jak i samych relacji, w szczególności w przypadku relacji wiele do wielu. Natomiast w bazach nierelacyjnych proces ten jest znacznie prostszy ze względu na brak relacji.

Wyniki wskazują, że w przypadku dużych zbiorów danych o złożonych relacjach, najbardziej optymalnym wyborem jest baza danych PostgreSQL. Pod względem czasu wykonywania zapytań zbliżone rezultaty osiągnęły bazy MySql, Oracle oraz MongoDB, natomiast najgorsze wyniki uzyskała baza Neo4j. Zgodnie z oczekiwaniami bazy relacyjne okazały się lepsze w badanych scenariuszach, z uwagi na relacyjny i stały schemat testowanej struktury. W przypadku złożonych zapytań typu SELECT, relacyjne bazy danych osiągnęły mniejszy czas wykonywania zapytań, co potwierdza hipotezę H2.

W przyszłych badaniach warto byłoby rozważyć analizę wpływu różnych wersji bibliotek połączeń na wydajność baz danych, a także ocenić wpływ optymalizacji systemu operacyjnego i ustawień kontenerów na czasy zapytań. Ponadto, warto przeprowadzić testy w scenariuszach z większym obciążeniem, aby sprawdzić, jak bazy danych radzą sobie w warunkach większej liczby równoczesnych operacji.

Literatura

[1] M. Kaufmann, A. Meier, SQL and NoSQL Databases, Springer, Cham, 2023.

[2] V. Abramova, J. Bernardino, P. Furtado, SQL or NoSQL? Performance and scalability evaluation, International Journal of Business Process Integration and Management 7 (2015) 314-321, https://doi.org/10.1504/IJBPIM.2015.073655.

[3] K. K.-Y. Lee, W.-C. Tang, K.-S. Choi, Alternatives to relational database: Comparison of NoSQL and XML approaches for clinical data storage, Computer Methods and Programs in Biomedicine 110 (2013) 99-109, https://doi.org/10.1016/j.cmpb.2012.10.018.

[4] G. Harrison, NoSQL, NewSQL i BigData. Bazy danych następnej generacji, Helion, Gliwice, 2021.

[5] J. Klein, I. Gorton, N. Ernst, P. Donohoe, K. Pham, C. Matser, Performance evaluation of NoSQL databases: a case study, In Proceedings of the 1st Workshop on Performance Analysis of Big Data Systems, Association for Computing Machinery (2015) 5-10, https://doi.org/10.1145/2694730.2694731.

[6] T. Seser, V. Pleština, F. Marjanica, Performance analysis of SQL Prepared Statements in CRUD operations, In 2022 7th International Conference on Smart and Sustainable Technologies (2022) 5-10, https://doi.org/10.23919/SpliTech55088.2022.9854303

[7] D. Yedilkhan, A. Mukasheva, D. Bissengaliyeva, Y. Suynullayev, Performance Analysis of Scaling NoSQL vs SQL: A Comparative Study of MongoDB, Cassandra, and PostgreSQL, In 2023 IEEE International Conference on Smart Information Systems and Technologies (SIST) (2023) 479-483, https://doi.org/10.1109/SIST58284.2023.10223568.

[8] G. K. Kaur, S. Singla, V. Khawas, Database Management System: A Study of Increasing Impact of NoSQL Databases, In 2023 International Conference on Advanced Computing & Communication Technologies (ICACCTech) (2023) 370-374, https://doi.org/10.1109/ICACCTech61146.2023.00067.

[9] D. G. Chandra, BASE analysis of NoSQL database, Future Generation Computer Systems 52 (2015) 13-21, https://doi.org/10.1016/j.future.2015.05.003.

[10] E. Tang, Y. Fan, Performance Comparison between Five NoSQL Databases, In 2016 7th International Conference on Cloud Computing and Big Data (CCBD) (2016) 105-109, https://doi.org/10.1109/CCBD.2016.030.

[11] Dokumentacja Csharp, https://learn.microsoft.com/en-us/dotnet/csharp/, [2024.10.29].

[12] A. Hejlsberg, M. Torgersen, S. Wiltamuth, P. Golde, C# Programming language, Addison-Wesley Professional, Boston, 2010.

[13] Z. Łata, M. Skublewska-Paszkowska, Performance analysis of databases created in virtualized and containerized environment, Journal of Computer Sciences Institute 28 (2023) 264–272, http://doi.org/10.35784/jcsi.3743.

[14] G. Dziewit, J. Korczyński, M. Skublewska-Pawszkowska, Performance analysis of relational databases Oracle and MS SQL based on desktop application, Journal of Computer Sciences Institute 8 (2018) 263-269, https://doi.org/10.35784/jcsi.693.

[15] L. Okman, N. Gal-Oz, Y. Gonen, E. Gudes, J. Abramov, Security Issues in NoSQL Databases, In 2011 IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom) (2011) 541-547, https://doi.org/10.1109/TrustCom.2011.70.

[16] G. K. Kaur, S. Singla, V. Khawas, Database Management System: A Study of Increasing Impact of NoSQL Databases, In 2023 International Conference on Advanced Computing & Communication Technologies (ICACCTech) (2023) 370–374, http://doi.org/10.1109/ICACCTech61146.2023.00067.

[17] N. Thakur, N. Gupta, Relational and Non Relational Databases: A Review, Journal of University of Shanghai for Science and Technology 23 (2021) 117–121, http://doi.org/10.51201/JUSST/21/08341.

[18] C. Győrödi, R. Győrödi, R. Sotoc, A Comparative Study of Relational and Non-Relational Database Models in a Web-Based Application, International Journal of Advanced Computer Science and Applications 6 (2015) 78-83, https://doi.org/10.14569/IJACSA.2015.061111.

[19] D. Kunda, H. Phiri, A Comparative Study of NoSQL and Relational Database, Zambia ICT Journal 1 (2017) 1-4, https://doi.org/10.33260/zictjournal.v1i1.8.

[20] S. Bjeladinovic, Z. Marjanovic, S. Babarogic, A proposal of architecture for integration and uniform use of hybrid SQL/NoSQL database components, Journal of Systems and Software 168 (2020) 1-29, https://doi.org/10.1016/j.jss.2020.110633.

[21] Zbiór danych dotyczących filmów, https://www.kaggle.com/datasets/rounakbanik/the-movies-dataset, [2024.10.29].

[22] Dokumentacja bazy danych Oracle, https://docs.oracle.com/en/database/index.html, [2024.10.29].

[23] Dokumentacja bazy danych MySql, https://dev.mysql.com/doc/, [2024.10.29].

[24] Dokumentacja bazy danych PostgreSQL, https://www.postgresql.org/docs/current/, [2024.10.29].

[25] Dokumentacja bazy danych MongoDB, https://www.mongodb.com/docs/manual/introduction/, [2024.10.29].

[26] Dokumentacja bazy danych Neo4j, https://neo4j.com/docs/, [2024.10.29].