JOURNAL of COMPUTER SCIENCES INSTITUTE

JCSI 19 (2021) 151–158
Received: 21 April 2021
Accepted: 13 May 2021

Analysis of the possibilities of optimizing SQL queries

Analiza możliwości optymalizacji zapytań SQL

Piotr Rymarski*, Grzegorz Kozieł

Department of Computer Science, Lublin University of Technology, ul. Nadbystrzycka 38, 20-618 Lublin, Poland

Abstract

Most of today's web applications run on relational database systems. Communication with them is possible through statements written in Structured Query Language (SQL). This paper presents the most popular relational database management systems and describes common ways to optimize SQL queries. Using the research environment based on fragment of the imdb.com database, implementing OracleDb, MySQL, Microsoft SQL Server and PostgreSQL engines, a number of test scenarios were performed. The aim was to check the performance changes of SQL queries resulting from syntax modification while maintaining the result, the impact of database organization, indexing and advanced mechanisms aimed at increasing the efficiency of operations performed, delivered in the systems used. The tests were carried out using a proprietary application written in Java using the Hibernate framework.

Keywords: SQL; query; optimization; analysis

Streszczenie

Większość obecnie istniejących aplikacji internetowych działa w oparciu o relacyjne systemy baz danych. Komunikacja z nimi jest możliwa poprzez instrukcje zapisywane w Structured Query Language (SQL). Niniejsza publikacja prezentuje najbardziej popularne systemy do zarządzania relacyjnymi bazami danych oraz opisuje powszechne sposoby optymalizacji zapytań SQL. Wykorzystując środowisko badawcze, w którym zaimportowano część bazy danych serwisu imdb.com oraz silniki OracleDb, MySQL, Microsoft SQL Server i PostgreSQL wykonano szereg scenariuszy testowych. Celem było sprawdzenie zmiany wydajności zapytań SQL wynikających z modyfikacji składni przy zachowaniu rezultatu, wpływu organizacji bazy danych, indeksowania oraz zaawansowanych mechanizmów, mających na celu wzrost efektywności wykonywanych operacji, dostarczanych w wykorzystanych systemach. Testy zostały przeprowadzone przy pomocy autorskiej aplikacji napisanej w języku Java z wykorzystaniem szkieletu programistycznego Hibernate.

Słowa kluczowe: SQL; zapytanie; optymalizacja; analiza

1. Wstęp

Większość współczesnych aplikacji internetowych działa w oparciu o architekturę model-widok-kontroler. Widok jest odpowiedzialny za wygląd i zachowanie aplikacji po stronie użytkownika, oraz wysyłanie żądań i danych do aplikacji serwerowej (ang. backend). Choć niedostrzegalny przez odbiorcę końcowego, to właśnie backend dostarcza wymaganych danych. Serwer aplikacji odpowiada za zapis, przetwarzanie i odczyt danych z 'bazy danych. Jedną z najważniejszych cech działania aplikacji internetowej jest jej wydajność. Choć istnieje możliwość, aby część logiki aplikacji została przeniesiona do bazy, zazwyczaj nie jest to pożądane działanie, a baza danych służy tylko i wyłącznie do magazynowania danych. Aby umieścić, odczytać, zmodyfikować lub usunąć dane z bazy, wykorzystywane są specjalne instrukcje. W przypadku relacyjnych baz danych do komunikacji z bazą używa się instrukcji języka SQL (ang. Structured Query Language). Do manipulacji danymi w SQL używane są zapytania typu DML (ang. Data Manipulation Language) [5].

Podczas działania bazy danych, w oparciu o którą działają sesje inicjowane przez użytkowników, wykonywane są symultanicznie tysiące operacji SCAN (służących do pozyskiwania danych przy pomocy instrukcji SELECT) oraz DML. Każde niewłaściwie sformułowane zapytanie generuje opóźnienie spowodowane złożonością obliczeniową operacji. Im większa liczba operacji, tym większe obciążenie serwera bazy danych. Wynikiem czego może być długi czas oczekiwania, lub nawet awaria serwera [8]. Jak istotny jest wpływ składni zapytań SQL na ich wydajność? Przy jakich technologiach i wielkościach baz danych jest to najbardziej zauważal-

*Corresponding author
Email address: piotr.rymarski@pollub.edu.pl (P. Rymarski)

ne, a przy jakich marginalne? Niniejszy artykuł pozwala w pewnym stopniu odpowiedzieć na powyższe pytania, prezentując wyniki przeprowadzonych badań.

2. Wybrane technologie

Już w 1970 roku idea relacyjnych baz danych została opisana przez F. Codda [1] i jej założenia nie zmieniły się do dziś. Relacyjna baza danych używa konceptu połączonych ze sobą dwuwymiarowych tabel składających się z rzędów i kolumn. Kolumny reprezentują poszczególne atrybuty encji, w rzędach zaś zapisywane są kolejne rekordy, stanowiące oddzielne instancje opisywanego obiektu. Obecnie w czołówce silników bazodanowych [6], nie tylko w ujęciu relacyjnych baz danych, znajdują się kolejno:

1. Oracle
2. MySQL
3. Microsoft SQL Server
4. PostgreSQL

2.1. Oracle Database

Oracle Database (Oracle DBMS lub tylko Oracle) to wielomodelowy system zarządzania relacyjnymi bazami danych - RDBMS (ang. Relational Database Management System) wyprodukowany i dystrybuowany przez Oracle Corporation. Początki systemu datuje się już na 1977 rok, kiedy to L. Ellison wraz z kolegami utworzył SDL (ang. Software Development Laboratories).

Obecnie najnowszą, stabilną wersją Oracle DBMS jest 19c. Silnik jest dostępny w wielu wariantach, z których wszystkie, za wyjątkiem wersji Express, która nie może być wykorzystywana w warunkach produkcyjnych, wymagają zakupu licencji do zastosowań komercyjnych. Główną cechą OracleDB jest multiplatformowość. System obsługuje ponad 100 platform hardware'owych oraz ponad 20 protokołów sieciowych [7], co zapewnia bezpieczeństwo przy planowaniu aktualizacji i modyfikacji środowiska, w którym pracuje baza. Kolejną zaletą jest możliwość przeniesienia części logiki aplikacji bezpośrednio do serwera bazy danych. Wszystkie wersje silnika zawierają języki i interfejsy, które pozwalają programistom na dostęp i manipulację danymi bezpośrednio w bazie.

Silnik Oracle jest ceniony także za dostarczanie rozbudowanych systemów przywracania Oracle Recovery Manager (RMAN) [2]. RMAN pozwala na jednorazową konfigurację przywracania, zautomatyzowane zarządzanie kopiami zapasowymi i zarchiwizowanymi logami, wznawianie procesów tworzenia kopii zapasowych, ich przywracania oraz na testowanie tych procesów.

2.2. MySQL

MySQL to system zarządzania reelacyjnymi bazami danych stworzony w 1995 roku przez szwedzką firmę MySQLAB. W 2008 roku MySQLAB została nabyta przez Sun Microsystems, która od 2010 jest częścią Oracle Corporation. MySQL Server i jego biblioteki są dystrybuowane w dwóch zakresach licencyjnych. Można ich używać w oparciu o GPL 2 (General Public Use License version 2) lub inne licencje, nie zezwalające na bezpłatne komercyjne wykorzystywanie. System zyskał wielu zwolenników, na co wskazuje fakt użycia MySQL w serwisach takich jak: Amazon, Uber, Twitter, czy Netflix.

Pierwotnie założeniem MySQL było dodanie indeksowania danych, aby przyspieszyć zapytania dla serwera mSQL, poprzez użycie metod sekwencyjnego dostępu ISAM. Osiągnięto to dzięki zastosowaniu specjalnego algorytmu zarządzania danymi nazwanego MyISAM storage engine, co okazało się wielkim sukcesem. W pierwszych fazach rozwoju MySQL stawiano na zapewnienie jak największej szybkości przy ograniczeniu dostępnych funkcjonalności. Brakowało na przykład mechanizmu transakcji, w rezultacie odstraszało to potencjalnych użytkowników. Jednak z czasem i wraz ze zmianami właścicieli systemu sytuacja uległa diametralnej poprawie, jeśli chodzi o zapewniane funkcjonalności. Dziś MySQL jest czymś więcej niż systemem zarządzania bazami danych, zapewniającym szybkie wykonywanie zapytań. Jego rozwój ciągle trwa i w kolejnych wersjach dodawane są coraz to nowe udogodnienia i rozwiązania.

2.3. Microsoft SQL Server

Microsoft SQL Server to system zarządzania relacyjnymi bazami danych wspierany i rozwijany przez Microsoft. Jest platformą typu klient/server. Jego podstawową funkcją jest przechowywanie i zapewnianie danych wymaganych przez aplikacje, które mogą działać w obrębie tego samego urządzenia lub zdalnie przez sieć. Historia tego RDBMS zaczęła się wydaniem SQL Server v1.0, będącego projektem bazującym na Synbase SQL Server. Pierwsza wersja była owocem kooperacji Microsoft z Synbase Inc. oraz Ashton-Tate Corp. W ciągu kolejnych lat firmy zaniechały dalszej współpracy, a Synbase zmieniło nazwę swojego produktu, pozostawiając SQL Server na wyłączność Microsoft.

Obecną najnowszą wersją systemu jest Microsoft SQL Server 2019, choć produkt jest oficjalnie wspierany do pięciu wersji wstecz (Microsoft SQL Server 2012). System dostępny jest w wielu różnych wersjach z podziałem na licencje, np. Enterprise i Express oraz specyfikację użycia, np. Developer i Datawarehouse Appliance Edition [3]. Microsoft SQL Server zawiera szeroki zestaw narzędzi administracyjnych (SQLCMD, SQL Server Management Studio, Azure Data Studio), analitycznych (SQL Server Analysis Services) oraz Business Inteligence.

2.4. PostgreSQL

PostgreSQL (Postgres) to darmowy RDBMS dostępny na zasadach PostgreSQL License (open-source). System zapewnia podobny zakres rozwiązań jak w przypadku konkurencji. Został zaprojektowany, aby radzić sobie z różnymi środowiskami, od pojedynczych maszyn

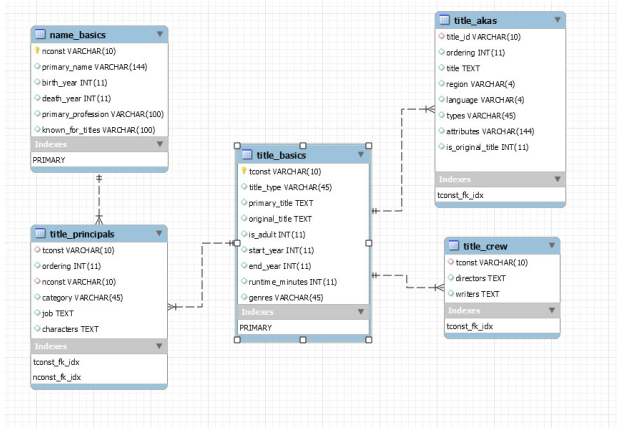
do hurtowni danych oraz serwisów internetowych z wieloma użytkownikami. W odróżnieniu od konkurencji, jego popularność gwałtownie rośnie i z kolejnymi latami przybywa entuzjastów tego rozwiązania.

Jedną z największych zalet silnika jest bardzo duże przystosowanie składni do oryginalnego SQL. W obecnej wersji Postgres spełnia 170 z 179 wymaganych funkcji standardu SQL:2016. Przy czym warto nadmienić, iż żaden z dostępnych RDBMS nie spełnia tego standardu w 100% [4].

Przy integracji z innymi systemami pomocny może być mechanizm FDW (ang. Foreign Data Wrappers), pozwalający na pozyskiwanie danych z zewnątrz (serwisy internetowe, system plików oraz inne bazy danych). Oznacza to, że zwykłe zapytania mogą używać tych źródeł pojedynczo lub dowolnie je łączyć i wykorzystywać i traktować je jak tabele zdefiniowane w bazie. Silnik zapewnia także programowanie proceduralne poprzez przechowywane funkcje i dyrektywę DO. Istotną funkcją PostgreSQL jest implementacja obiektowości poprzez dziedziczenie tabel. Pozwala to na uniknięcie pustych kolumn przy opisywaniu pokrewnych obiektów.

3. Metodyka badawcza

Do badania czasu wykonania zapytań wymagana była baza danych posiadająca tabele magazynujące duże liczby rekordów. Zdecydowano się wykorzystać dane udostępnione przez portal imdb.com [9]. Do niekomercyjnego użytku osobistego można pobrać 7 zestawów danych, spośród których do badania wybrano 5. Na Rysunku 1 przedstawiono diagram ERD bazy danych powstałej w wyniku importu wybranych zestawów.



Rysunek 1: Diagram ERD testowej bazy danych.

Istotna dla przebiegu badań była możliwość podglądu planu wykonania przygotowanych zapytań dla każdego z użytych systemów. Plan wykonania dostarcza wielu istotnych informacji o kolejnych etapach, koszcie i czasie wykonania operacji. Pozwala również na podgląd kolejnych etapów wykonania zapytania, a przede wszystkim faktu wykorzystania, bądź pominięcia istniejących indeksów.

W celach badawczych przygotowano aplikację pozwalającą na pomiar czasu wykonywania zapytań w kontrolowanych warunkach. Kod aplikacji napisano w języku Java z wykorzystaniem narzędzia Maven dla ułatwienia organizacji wymaganych bibliotek – zależności. Istotnym dla działania programu było zastosowanie wzorca programistycznego Hibernate z uwzględnieniem interfejsu Statistics. Pozwoliło to na prostą konfigurację i pomiar wymaganych wartości.

Wszystkie badania zostały przeprowadzone w takim samym środowisku testowym, tj. na tym samym komputerze (Tabela 1), za pośrednictwem lokalnych baz danych oraz przy wykorzystaniu tej samej aplikacji testowej z analogiczną konfiguracją. Aby zapewnić jak najdokładniejsze pomiary w momencie prowadzenia badań na komputerze uruchomione były wyłącznie:

- system operacyjny
- aplikacja testowa
- badany silnik bazy danych

Komputer był odłaczony od sieci, a pozostałe aplikacje, przede wszystkim oprogramowanie antywirusowe oraz zapora sieciowa, były wyłączone.

Tabela 1: Specyfikacja komputera, na którym przeprowodzono badania

System operacyjny	Windows 10 Home 64-bit
Procesor	AMD Ryzen 5 2600
Pamięć RAM	16 GB 3000MHz CL15
Dysk	Samsung SSD 970 EVO 1TB

Przed wykonaniem badania przygotowano szereg scenariuszy, z których każdy wykonano dziesięciokrotnie. Po odrzuceniu wyników skrajnych obliczono średni czas wykonania każdego z zapytań dla wszystkich badanych systemów i przedstawiono wyniki badań na wykresach.

4. Wyniki badań

Poniżej przedstawiono wybrane scenariusze badawcze wraz z diagramami prezentującymi ich wyniki.

4.1. Wpływ ograniczania ilości zwracanych danych na wydajność zapytania

Pierwszy scenariusz badawczy polegał na sprawdzeniu wpływu ograniczenia ilości pozyskiwanych jednorazowo danych na czas wykonania zapytania. Badana baza danych zawierała tabele magazynujące miliony rekordów (Tabela 2), więc spodziewano się znacznego spadku czasu wykonania zapytań. Omawiany scenariusz wykonano na dwa sposoby: ograniczając liczbę zwracanych kolumn (scenariusz 1.a) oraz ograniczając liczbę zwracanych rekordów (scenariusz 1.b).

Tabela 2: Liczebność rekordów każdej z zaimportowanych tabel

Tabela	Liczba rekordów
name_basics	10 628 431
title_basics	7 498 564
title_akas	24 757 132
title_principals	42 496 909
title_crew	7 495 387

Ograniczenie ilości zwracanych danych można rozpatrywać w 2 aspektach:

a) Zmniejszenia liczby zwracanych kolumn;

b) Zmniejszenia liczby zwracanych rekordów.

Tabela 3: Czas wykonania zapytań w scenariuszu 1.a

Silnik bazodanowy	Czas wykonania zapytania 1. [s]	Czas wykonania zapytania 2. [s]
Oracle	0,14	0,08
MySQL	7,59	6,63
SQL Server	0,27	0,04
PostgreSQL	4,86	1,05

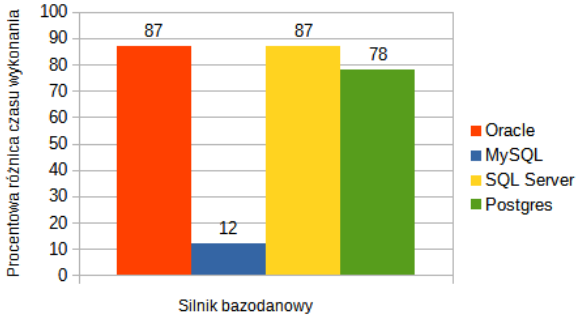
Tabela 3 przedstawia zestawienie czasu wykonania zapytań (Listing 1) w scenariuszu 1.a. Aby jednoznacznie zobrazować efektywność każdego badanego zabiegu optymalizacyjnego, wszystkie wykresy przedstawione dalej w artykule pokazują procentową różnicę pomiędzy średnim czasem wykonania zapytań z uwzględnieniem wszystkich badanych silników bazodanowych. Owa różnica może być interpretowana jako procentowy wzrost wydajności zapytania (chyba, że na wykresie zaznaczono inaczej).

Zdecydowano się na takie rozwiązanie, ponieważ różnice czasu wykonania zapytań pomiędzy pomiarami w przypadku badanych silników były na tyle znaczące, że wynikowe wykresy zawierające dokładne otrzymane wyniki w mikrosekundach były nieczytelne. W przypadku MySQL czas wykonania zapytania potrafił ponad tysiąckrotnie przekraczać wyniki otrzymane z konkurencyjnych silników.

Listing 1: Zapytania użyte w scenariuszu 1.a

```
--wszystkie kolumy
SELECT * FROM title_basics
WHERE start_year = 2010;

--tylko 3 wybrane kolumy
SELECT tconst, primary_title, start_year
FROM title_basics
WHERE start_year = 2010;
```

W scenariuszu 1.a zbadano wpływ zmiany liczby zwracanych kolumn ze wszystkich (9) do tylko 3 wybranych: tconst, primary_title i start_year. Wynik badania zobrazowano na Rysunku 2.

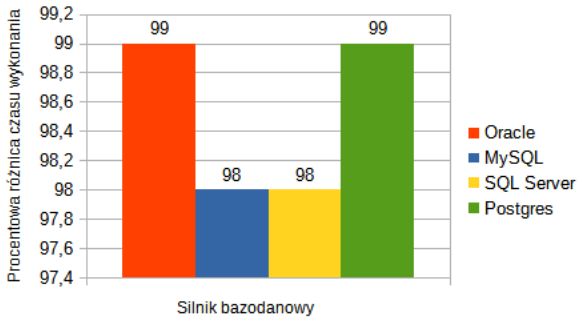


Rysunek 2: Diagram przedstawiający procentowy spadek czasu wykonania przy ograniczeniu liczby zwracanych kolumn.

W tym przypadku scenariusza 1.b zbadano wpływ zmniejszenia liczby zwracanych rekordów (z 76344 do 100 pierwszych) na czas wykonania zapytania (Rysunek 3). Warto zaznaczyć, że w celu osiągnięcia tego samego efektu wymagane było użycie odmiennej składni dla wybranych silników (Listing 2).

Listing 2: Zapytania użyte w scenariuszu 1.b

```
--wszystkie rekordy
SELECT tconst FROM title_basics
WHERE start_year = 2010;

--tylko 100 rekordów
--ORACLE
SELECT  tconst FROM title_basics
WHERE start_year = 2010 AND ROWNUM <= 100;

--MySQL i PostgreSQL
SELECT tconst FROM title_basics
WHERE start_year = 2010 LIMIT 100;

--SQL Server
SELECT TOP 100 tconst FROM title_basics
WHERE start_year = 2010;
```



Rysunek 3: Diagram przedstawiający procentowy spadek czasu wykonania przy ograniczeniu liczby zwracanych rekordów.

Jak łatwo zauważyć, ograniczenie zestawu zwracanych danych w obu przypadkach znacząco wpływa na czas wykonania zapytania.

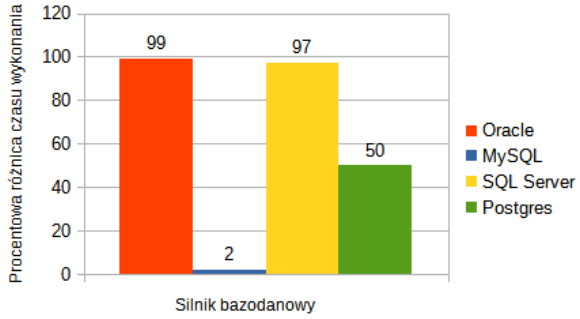
4.2. Optymalizacja klauzuli UNION

W scenariuszu 2 zbadano wpływ użycia klauzuli UNION ALL w miejsce klauzuli UNION na czas wykonania zapytania (Listing 3). Należy zauważyć jednak, iż w celu osiągnięcia identycznego zestawu danych w obu przypadkach nie może on zawierać duplikatów.

Listing 3: Zapytania użyte w scenariuszu 2

```
--UNION
SELECT tconst FROM title_basics
WHERE start_year = 1999
UNION
SELECT tconst FROM title_basics
WHERE start_year = 2000

--UNION ALL
SELECT tconst FROM title_basics
WHERE start_year = 1999
UNION ALL
SELECT tconst FROM title_basics
WHERE start_year = 2000;
```

Rysunek 4 przedstawia wynik przeprowadzonego badania. Wykonana operacja pozwoliła na zaoszczędzenie znacznej ilości czasu. Dzieje się tak, ponieważ operacja UNION ALL nie wykonuje niepotrzebnego filtrowania w poszukiwaniu duplikatów.



Rysunek 4: Diagram przedstawiający procentowy spadek czasu wykonania przy zastosowaniu klauzuli UNION ALL.

4.3. Porównanie wydajności klauzuli filtrującej z konstrukcją JOIN

Rysunek 5 przedstawia wyniki badań przeprowadzonych według scenariusza testowego 3, w którym sprawdzono wpływ zastąpienia klauzuli łączącej JOIN przy pomocy instrukcji WHERE (Listing 4).

Listing 4: Zapytania użyte w scenariuszu 3

```
--INNER JOIN
SELECT nb.primary_name
FROM name_basics nb
JOIN title_principals tp
ON nb.nconst = tp.nconst
WHERE tp.category = 'director'

--WHERE
SELECT nb.primary_name
FROM name_basics nb, title_principals tp
WHERE nb.nconst = tp.nconst
AND tp.category = 'director';
```

We wszystkich badanych systemach, z wyjątkiem Microsoft SQL Server, dla którego wystąpił spadek wydajności, operacja została potraktowana jako łączenie wewnętrzne.

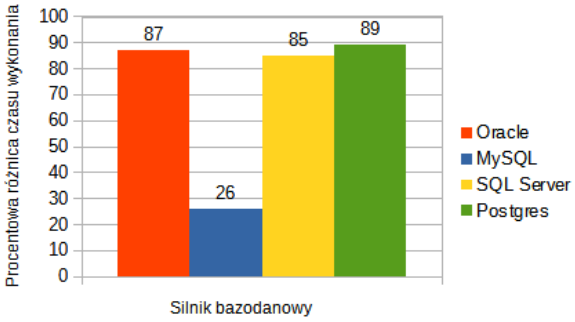
Rysunek 5: Diagram przedstawiający procentowy wzrost czasu wykonania przy zastąpieniu konstrukcji JOIN klauzulą WHERE.

4.4. Optymalizacja klauzuli grupującej

W scenariuszu 4 zbadano wpływ lokowania instrukcji filtrujących w klauzuli grupującej z dyrektywą HAVING (Listing 5). Rysunek 6 przedstawia procentowy wzrost wydajności po wydzieleniu filtrowania do klauzuli WHERE.

Listing 5: Zapytania użyte w scenariuszu 4

```
--Filtrowanie wyłącznie w klauzuli HAVING
SELECT birth_year, COUNT(nconst) AS people_count
FROM name_basics GROUP BY birth_year
HAVING birth_year > 2000 AND COUNT(nconst) > 50;

--Filtrowanie przeniesione do klauzuli WHERE
SELECT birth_year, COUNT(nconst) AS people_count
FROM name_basics WHERE birth_year > 2000
GROUP BY birth_year HAVING COUNT(nconst) > 50;
```



Rysunek 6: Diagram przedstawiający procentowy spadek czasu wykonania zapytania po przeniesieniu filtrowania do klauzuli WHERE.

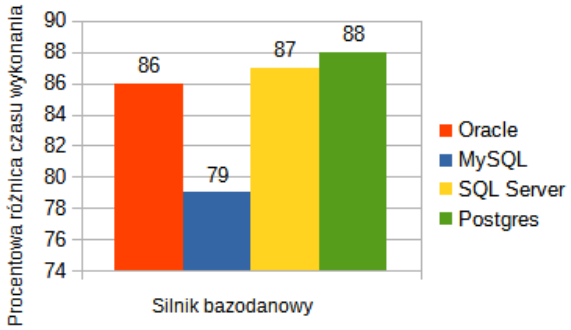
4.5. Zbadanie wydajności operacji full index scan w porównaniu do full table scan

Operacja full index scan ma miejsce wtedy, gdy wszystkie wymagane w zapytaniu kolumny są zdefiniowane wewnątrz indeksu. W takim przypadku nie ma potrzeby odwoływania się do oryginalnej tabeli, a operacja dodatkowo przeprowadzona jest na posortowanym zestawie danych. Zgodnie z założeniami wykonanie zapytania (Listing 6) po tym jak dodano indeks na kolumnie title_basics.start_year prze-

biegło w znacznie krótszym czasie (Rysunek 7).

Listing 6: Zapytanie użyte w scenariuszu 5

```
SELECT start_year, tconst
FROM title_basics
WHERE start_year > 2010;
```



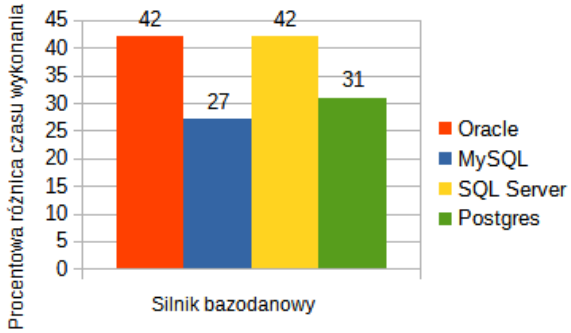
Rysunek 7: Diagram przedstawiający procentowy spadek czasu wykonania zapytania po dodaniu indeksu na filtrowaną kolumnę.

4.6. Sprawdzenie wzrostu wydajności przy pozbyciu się redundantnego sortowania

Domyślnie dane w indeksie posortowane są rosnąco, więc pominięcie dodatkowego sortowania w zapytaniu nie powinno wpłynąć na jego rezultat przy jednoczesnej poprawie wydajności. Aby to sprawdzić porównano czas wykonania zapytania z poprzedniego scenariusza z zapytaniem przedstawionym na Listingu 7. Rezultat badania obrazuje Rysunek 8.

Listing 7: Zapytanie użyte w scenariuszu 6

```
SELECT start_year, tconst
FROM title_basics
WHERE start_year > 2010
ORDER BY start_year;
```



Rysunek 8: Diagram przedstawiający procentowy wzrost czasu wykonania po dodaniu redundantnego sortowania po indeksowanej kolumnie.

4.7. Zbadanie wpływu indeksu na zapytania z filtrowaniem przy użyciu operatora SARGable i Non-SARGable

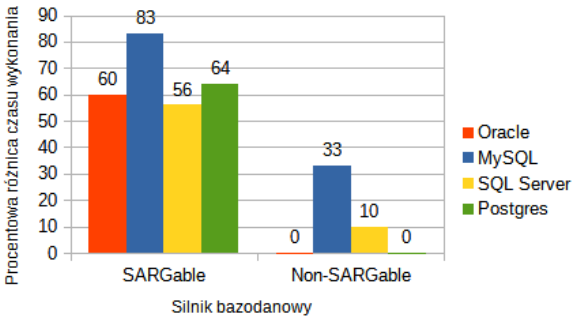
W SQL można wyróżnić 2 typy operatorów porównania:

- SARGable (operatory, dla których serwer może użyć indeksu, mianowicie: =, ¿, ¡, ¿=, ¡=, IN, BETWEEN oraz LIKE, ale tylko w kwestii pasujących przedrostków);
- Non-SARGable (operatory: NOT, NOT IN, ¡¿, LIKE (argument zaczyna się od „%”), a także wszystkie funkcje wykonujące kalkulacje, zmiany oraz konwersje typów na kolumnach tabeli).

W bieżącym scenariuszu zbadano wpływ indeksowania na zapytania zawierającego oba rodzaje operatorów. Aby sprawdzić jak zmieni się czas wykonania dla zapytań zawierających każdy z wymienionych typów operatorów porównania zastosowano parę analogicznych zapytań z Listingu 8 przed oraz po dodaniu indeksu na kolumnie title_basics.is_adult.

Listing 8: Zapytania użyte w scenariuszu 7

```
--SARGable
SELECT primary_title
FROM title_basics
WHERE is_adult = 1
AND start_year > 2010;

--Non-SARGable
SELECT primary_title
FROM title_basics
WHERE is_adult <> 0
AND start_year > 2010;
```



Rysunek 9: Diagram przedstawiający procentowy spadek czasu wykonania po dodaniu indeksu dla funkcji filtrujących typu SARGable i Non-SARGable.

Jednoznaczny wzrost wydajności można zaobserwować dla pierwszego z zapytań, wykorzystującego operator typu SARGable (Rysunek 9). Warto także wspomnieć, że dodanie indeksu miało pozytywny wpływ na czas wykonania drugiego zapytania w środowiskach MySQL oraz SQL Server.

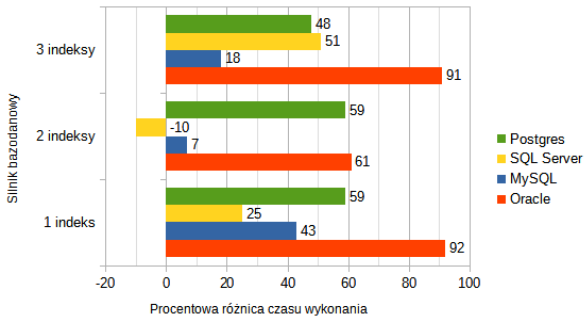
4.8. Indeksowanie a DML

Indeksy odgrywają znaczącą rolę nie tylko w przypadku skanów tabeli, ale także wpływają na wydajność instrukcji DML. Aby to sprawdzić wykonano sekwencję instrukcji DML (Listing 9) kontrolnie przed oraz kolejno po dodaniu indeksów na kolumnach primary_name, birth_year, primary_profession tabeli name_basics.

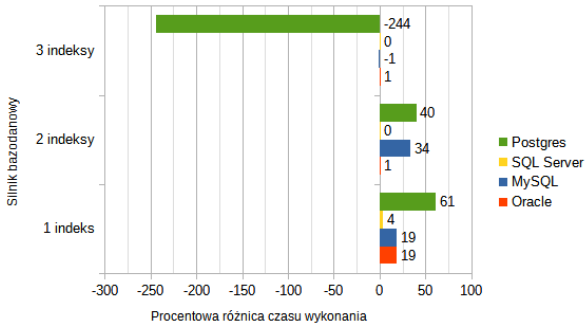
Listing 9: Zapytania użyte w scenariuszu 8

```
--INSERT
INSERT INTO name_basics(
  nconst,
  primary_name,
  birth_year,
  primary_profession
)
VALUES (
  'nm9999999',
  'Unikalne Nazwisko',
  1990,
  'actor'
);

--UPDATE
UPDATE name_basics
SET death_year = 2021
WHERE nconst= 'nm9999999';

--DELETE
DELETE FROM name_basics
WHERE nconst= 'nm9999999';
```

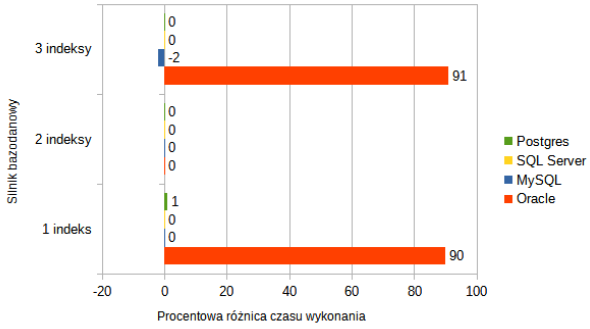
Rysunki 10, 11 i 12 przedstawiają kolejno wyniki badań wpływu indeksowania na operacje INSERT, UPDATE i DELETE.



Rysunek 10: Diagram przedstawiający procentowy spadek czasu wykonania po dodaniu kolejnych indeksów dla operacji INSERT.



Rysunek 11: Diagram przedstawiający procentowy spadek czasu wykonania po dodaniu kolejnych indeksów dla operacji UPDATE.



Rysunek 12: Diagram przedstawiający procentowy spadek czasu wykonania po dodaniu kolejnych indeksów dla operacji DELETE.

Wbrew oczekiwaniom, dodanie indeksów spowodowało głównie wzrost wydajności wykonanych operacji manipulacji danymi.

5. Wnioski

Już podczas przeglądu literatury dało się zauważyć, iż pomimo wspólnego przeznaczenia badanych systemów, istnieją pomiędzy nimi różnice w sposobie konfiguracji, używanym dialekcie SQL oraz czasie wykonania zapytań, które w odczuwalnym stopniu skomplikowały proces badawczy.

Przeprowadzone badania miały na celu wyróżnienie oraz sprawdzenie możliwości optymalizacji zapytań SQL. Nie wszystkie sposoby znalezione w literaturze oraz Internecie znalazły jednak zastosowanie, gdyż w wielu przypadkach optymalizatory badanych systemów wykonywały poprawnie zapytania umyślnie zapisane w nieoptymalny sposób. Przykładem takiej sytuacji było wykonanie optymalnego łączenia wewnętrznego tabel nawet, kiedy do ich łączenia użyto klauzuli WHERE.

Otrzymane wyniki jednoznacznie wskazały ograniczanie zestawu zwracanych danych (kolumn i rekordów) jako najlepszy sposób na zwiększenie wydajności zapytań. Praktykę ograniczania liczebności zbioru danych stosuje się m.in. przy paginacji stosowanej podczas wyświetlania danych. Aplikacje zazwyczaj nie pobierają całego zestawu danych, gdyż jest to bardzo niewydajne. Zamiast tego pobiera się tylko pewien zakres z posortowanych wcześniej danych. Kolejnym ważnym sposobem optymalizacji SQL jest używanie odpowiednich operatorów wewnątrz klauzul filtrujących oraz wykluczenie filtrowanych kolumn z wykonywanych działań. Dzięki temu silnik bazy danych będzie mógł użyć wcześniej utworzonych indeksów. Scenariusze badawcze oparte o wykorzystanie indeksów w znacznym stopniu pokryły się z oczekiwaniami, przynosząc znaczny wzrost wydajności.

Według informacji zebranych w literaturze, liczba indeksów powinna mieć jednoznacznie negatywny wpływ na czas wykonywania instrukcji DML, co nie sprawdziło się podczas procesu badawczego. W badaniu użyto indeksów zbudowanych na pojedynczych kolumnach, więc dostosowywanie indeksu do zmian w oryginalnej tabe-

li miało mniej znaczący wpływ niż korzyści wynikające z użycia indeksów w klauzulach filtrujących. Może to być związane z liczbą kolumn w klauzulach filtrujących w przypadku UPDATE i DELETE oraz złożonością indeksów, które muszą zostać zaktualizowane.

Ze względu na otrzymane rezultaty świadczące o podniesieniu wydajności w wielu scenariuszach badawczych, można jednoznacznie stwierdzić, iż optymalizacja SQL jest możliwa oraz pożądana. Warto także podkreślić fakt, że przy konstrukcji zapytań niezastąpiona jest wiedza na temat bazy danych, przede wszystkim na temat istniejących indeksów, oraz dokładne sprecyzowanie zakresu danych wynikowych.

Literatura

[1] R. Greenwald, R. Stackowiak, J. Stern, Oracle Essentials, Fifth Edition, O'Reilly Media, Sebastopol, 2013.

[2] P. Muryjas, M. Skublewska-Paszkowska, D. Gutek, Współczesne Technologie Informatyczne. Eksploatacja baz danych, Politechnika Lubelska, 2011.

[3] Editions and supported features of SQL Server, https://docs.microsoft.com/en-us/sql/sql-server/editions-and-components-of-sql-server-version-15?view=sql-server-ver15, [29.12.2020].

[4] What is PostgreSQL, https://www.postgresql.org/about/, [29.12.2020].

[5] What is SQL, https://www.infoworld.com/article/3219795/what-is-sql-the-lingua-franca-of-data-analysis.html, [02.01.2021].

[6] DB-Engines Ranking, https://db-engines.com/en/ranking, [02.12.2020].

[7] Oracle Database Features, https://docs.oracle.com/cd/B19306˙01/server.102/b14220/intro.html, [29.12.2020].

[8] Why Do Databases Crash, https://www.zmanda.com/blog/why-do-databases-crash-and-what-to-do-about-it/, [21.01.2021].

[9] IMDb Datasets, https://www.imdb.com/interfaces/, [02.12.2020].