Sprawozdanie

Porównanie wydajności złączeń i zagnieżdżeń

AGH University of Science and Technology

Szymon Trojak

WGGIOŚ

Semestr 4

Cel sprawozdania

Celem analizy było porównanie wydajności kwerend bazujących na złączeniach i zagnieżdżeniach dla tabeli geologicznej.

Do wykonania zostały użyte trzy system zarządzania relacyjnymi bazami danych :

- PostgreSQL
- SQL Server
- MySQL

Konfiguracja sprzętowa

Komputer

- CPU: Intel(R) Core(TM) i5-1035G1 CPU @ 1.00GHz, 1190 MHz, Rdzenie: 4
- RAM: 16
- GPU: NVIDIA GeForce MX250
- System operacyjny: Microsoft Windows 11 Home

Programy

- PostgreSQL 15.3.1- Windows-x64
- SQL Server for Windows
- MySQL 8.0.33

Konstrukcja baz danych

Jako materiał do analizy posłużyła tabela geochronologiczna, która obrazuje przebieg historii Ziemi na podstawie następstwa procesów i warstw skalnych .Obecnie przyjęta tabela geochronologiczna została ustalona przez Międzynarodową Komisję Stratygrafii (ICS). W *tabeli 1* przedstawiono taksonomię dla pięciu jednostek geochronologicznych: eonu, ery, okresu, epoki wieku. Na podstawie części poniższych danych (68 rekordów) wykonano bazę danych znormalizowanych . Osobno dla SQL Server, PostgreSQL.

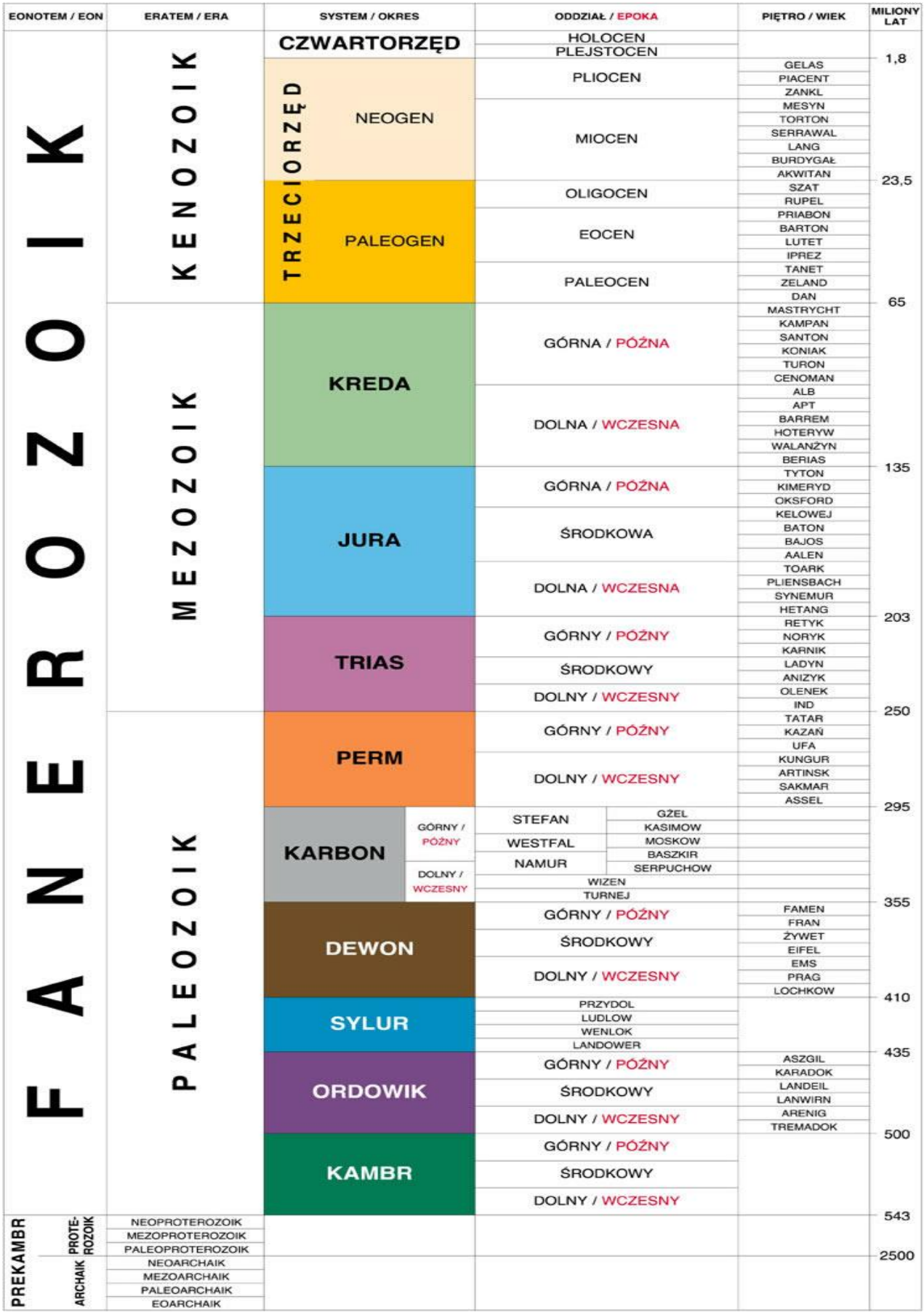


Tabela 1 Tabela stratygraficzna

Zapytania

W teście wykonano szereg zapytań sprawdzających wydajność złączeń i zagnieżdżeń z tabelą geochronologiczną.

Zapytanie 1 , którego celem jest złączenie syntetycznej tablicy miliona wyników z tabelą geochronologiczną w postaci zdenormalizowanej, przy czym do warunku złączenia dodano operację modulo, dopasowującą zakresy wartości złączanych kolumn:

- MySQL

```
#1
SELECT COUNT(*) FROM geo1.Milion INNER JOIN GeoTabela ON
    (mod(Milion.liczba,77)=(GeoTabela.id_pietro));
```

- PostgreSQL

```
SELECT COUNT(*) AS zl1
FROM geo1.Milion
JOIN GeoTabela ON (geo1.Milion.liczba % 68) = GeoTabela.id_pietro;
```

- SQL Server

```
SELECT COUNT(*) AS ZL1

FROM liczby.milion m
JOIN GeoTabela ON (m.liczba % 68) = GeoTabela.id_pietro;
```

Zapytanie 2 , którego celem jest złączenie syntetycznej tablicy miliona wyników z tabelą geochronologiczną w postaci znormalizowanej, reprezentowaną przez złączenia pięciu tabel.

- MySQL

```
#2
SELECT COUNT(*) FROM geo1.Milion INNER JOIN  geo1.GeoPietro
ON
    (mod(Milion.liczba,77)=GeoPietro.id_pietro)
    NATURAL JOIN geo1.GeoEpoka NATURAL JOIN geo1.GeoOkres
NATURAL JOIN geo1.GeoEra NATURAL JOIN geo1.GeoEon;
```

- PostgreSQL

```
SELECT COUNT(*) AS zl2
FROM geo1.Milion
JOIN geo1.GeoPietro ON (geo1.Milion.liczba % 68) = geo1.GeoPietro.id_pietro
JOIN geo1.GeoEpoka ON geo1.GeoEpoka.id_epoka = geo1.GeoPietro.id_epoka
JOIN geo1.GeoOkres ON geo1.GeoOkres.id_okres = geo1.GeoEpoka.id_okres
JOIN geo1.GeoEra ON geo1.GeoEra.id_era = geo1.GeoOkres.id_era
JOIN geo1.GeoEon ON geo1.GeoEon.id_eon = geo1.GeoEra.id_eon;
```

- SQL Server

```
SELECT COUNT(*) AS ZL2
FROM liczby.milion m
JOIN GeoPietro ON (m.liczba % 68) = GeoPietro.id_pietro
JOIN GeoEpoka ON GeoEpoka.id_epoka = GeoPietro.id_epoka
JOIN GeoOkres ON GeoOkres.id_okres = GeoEpoka.id_okres
JOIN GeoEra ON GeoEra.id_era = GeoOkres.id_era
JOIN GeoEon ON GeoEon.id_eon = GeoEra.id_eon
```

Zapytanie 3 , którego celem jest złączenie syntetycznej tablicy miliona wyników z tabelą geochronologiczną w postaci zdenormalizowanej, przy czym złączenie jest wykonywane poprzez zagnieżdżenie skorelowane:

- MySQL

```
#3
SELECT COUNT(*) FROM geo1.Milion WHERE mod(Milion.liczba,77)=
    (SELECT id_pietro FROM GeoTabela   WHERE
mod(Milion.liczba,77)=(id_pietro));
```

- PostgreSQL

```
ELECT COUNT(*) AS zg3
FROM geo1.Milion
WHERE (geo1.Milion.liczba % 68) =
(SELECT id_pietro
FROM GeoTabela
WHERE (geo1.Milion.liczba % 68) = (id_pietro));
```

- SQL Server

```
SELECT COUNT(*) AS ZG3
FROM liczby.milion m
WHERE (m.liczba % 68) =
```

```
(SELECT id_pietro
FROM GeoTabela
WHERE (m.liczba % 68) = (id_pietro));
```

Zapytanie 4 , którego celem jest złączenie syntetycznej tablicy miliona wyników z tabelą geochronologiczną w postaci znormalizowanej, przy czym złączenie jest wykonywane poprzez zagnieżdżenie skorelowane, a zapytanie wewnętrzne jest złączeniem ta-bel poszczególnych jednostek geochronologicznych:

- MySQL

```
#4
SELECT COUNT(*) FROM geo1.Milion WHERE mod(Milion.liczba,77)
IN
    (SELECT geo1.GeoPietro.id_pietro FROM geo1.GeoPietro
NATURAL JOIN
    geo1.GeoEpoka NATURAL JOIN   geo1.GeoOkres NATURAL JOIN
geo1.GeoEra NATURAL JOIN geo1.GeoEon);
```

- PostgreSQL

```
SELECT COUNT(*) AS zg4
FROM geo1.Milion
WHERE (geo1.Milion.liczba % 68) IN
(SELECT geo1.GeoPietro.id_pietro
FROM geo1.GeoPietro
JOIN geo1.GeoEpoka ON geo1.GeoEpoka.id_epoka = geo1.GeoPietro.id_epoka
JOIN geo1.GeoOkres ON geo1.GeoOkres.id_okres = geo1.GeoEpoka.id_okres
JOIN geo1.GeoEra ON geo1.GeoEra.id_era = geo1.GeoOkres.id_era

JOIN geo1.GeoEon ON geo1.GeoEon.id_eon = geo1.GeoEra.id_eon);
```

- SQL Server

```
SELECT COUNT(*) AS ZG4
FROM liczby.milion m
WHERE (m.liczba % 68) IN
(SELECT GeoPietro.id_pietro
FROM GeoPietro
JOIN GeoEpoka ON GeoEpoka.id_epoka = GeoPietro.id_epoka
JOIN GeoOkres ON GeoOkres.id_okres = GeoEpoka.id_okres
JOIN GeoEra ON GeoEra.id_era = GeoOkres.id_era
JOIN GeoEon ON GeoEon.id_eon = GeoEra.id_eon);
```

Testy wydajności

W testach skupiono się na porównaniu wydajności złączeń oraz zapytań zagnieżdżonych, wykonywanych na tabelach o dużej liczbie danych. Testy wykonano w programie:

- PostgreSQL
- SQL Server for Windows
- MySQL

W zapytaniach testowych łączono dane z tabeli geochronologicznej z syntetycznymi danymi o rozkładzie jednostajnym z tabeli *Milion*, wypełnionej kolejnymi liczbami naturalnymi od 0 do 999 999. Aby otrzymać tabelę *Milion* wykonano dodatkową tabelę *Dziesiec* wypełnioną liczbami od 0 do 9, które po złączeniu i niewielkiej modyfikacji umożliwiły otrzymanie oczekiwanego wyniku- tabeli od 0 do 999 999. Celem tego zabiegu było otrzymanie dużej ilości plików, które pozwolą na operację na dużych bazach danych.

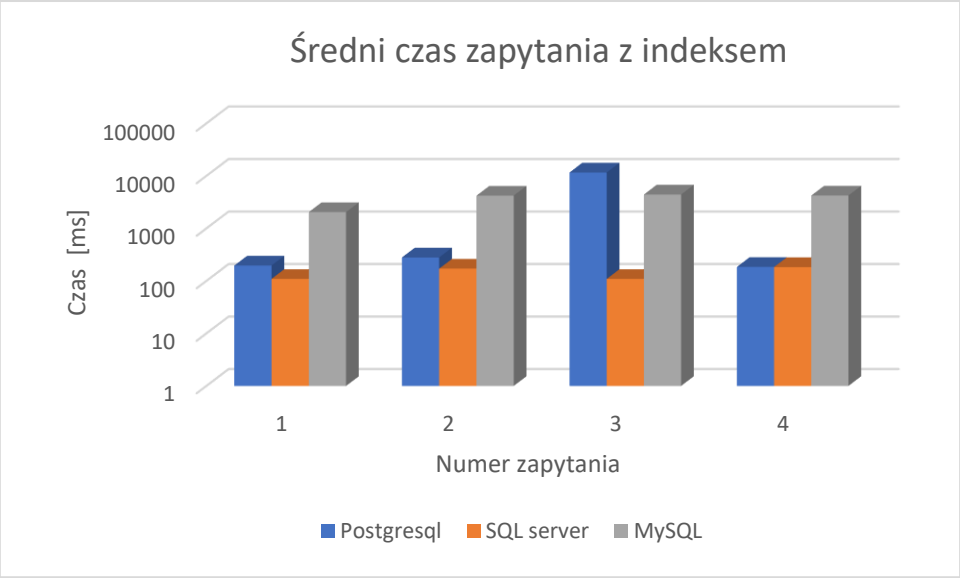
Wyniki

Wykonano pomiary czasu dla zapytania z indeksem i bez indeksu. Dla każdego programu: PostgresSQL, SQL Server i MySQL wykonano ręcznie 10 razy po 4 zapytania, a następnie policzono średnią i przedstawiono wyniki w postaci histogramu w skali logarytmicznej. Każdy z poniższych wykresów obrazuje ilość czasu jaki potrzebował konkretny system na wykonanie zadanego mu zapytania (czas podany w ms).

Poniżej znajdują się tabele z średnim czasem dla każdego zapytania .

z indeksem

nr zapytania	1	2	3	4
Postgresql	196,4	279,9	11614,3	184,7
SQL server	109,4	171,8	109,2	183,7
MySQL	2043	4210	4424,4	4233,5

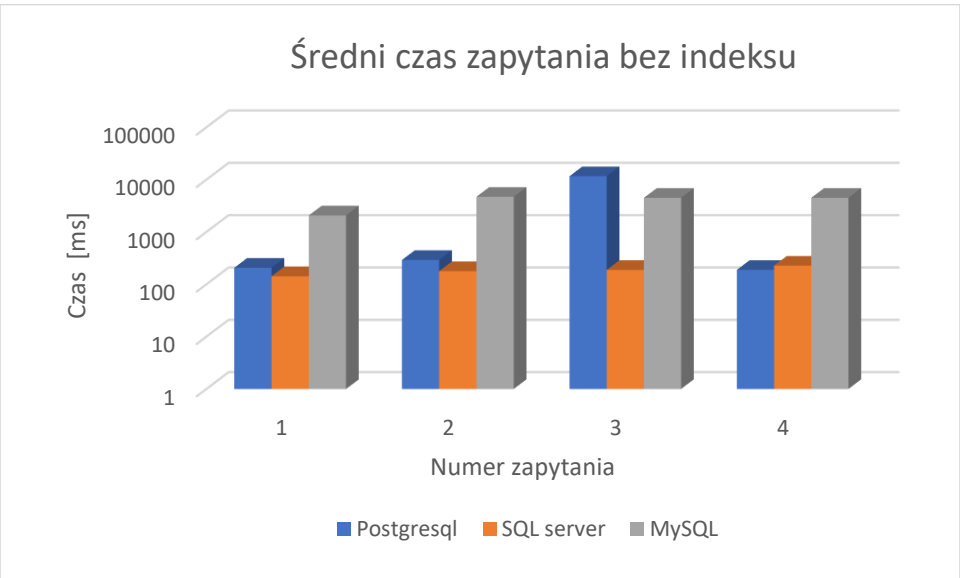


Wykres 1 Średni czas zapytania dla wartości indeksowanych

bez indeksu

nr zapytania	1	2	3	4
Postgresql	207,2	292,3	11629,4	188,8
SQL server	142,5	178,7	188,6	227,5
MySQL	2073,7	4700,667	4494	4493,667

Z powyższych tabel bardzo łatwo można wywnioskować, że zapytania z indeksem są szybsze od zapytań bez indeksu.



Wykres 2 Średni czas zapytania dla wartości nie indeksowanych

Podsumowanie

Wnioski jakie można wyciągnąć odczytując powyższe tabele i wykresy :

- Zapytania indeksowane są szybsze od zapytań nie indeksowanych, niezależnie od środowiska (programu) w którym zostały wykonane.
- SQL Server szybciej niż PostgreSQL przetwarza zadane mu zapytania niezależnie czy są indeksowane czy nie.
- MySQL najdłużej przetwarza dane, natomiast najszybciej przetwarza SQL Server.
- Postać zdenormalizowana w większości przypadków jest szybsza od postaci znormalizowanej

Podsumowując, mino iż normalizacja jest mniej wydajna, pozwala ona na przejrzyste i zrozumiałe przechowywanie danych, przez co zmniejsza szanse na wystąpienie błędów oraz ułatwia zarządzanie takimi danymi.