# 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 dwa programy umożliwiające pracę na dużych bazach danych :

- PostgreSQL
- SQL Server

# 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

# 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 poniższych danych wykonano bazę danych znormalizowanych . Osobno dla SQL Server i PostgreSQL.

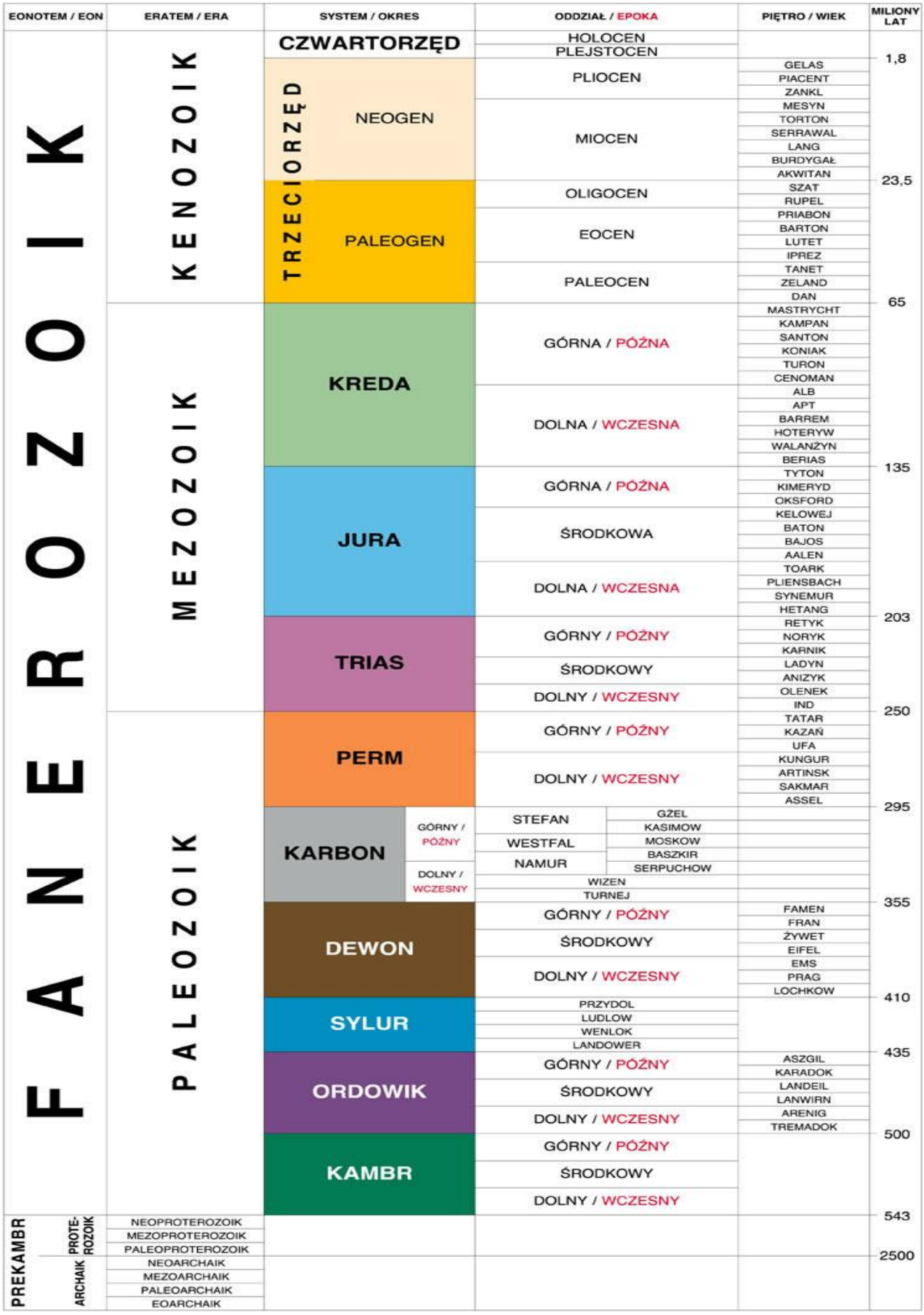



*Tabela 1 Tabela stratygraficzna*

## Zapytania

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

Zapytanie 1 (1 ZL), którego celem jest złączenie syntetycznej tablicy miliona wyników ztabelą geochronologiczną w postaci zdenormalizowanej, przy czym do warunku złącze-nia dodano operację modulo, dopasowującą zakresy wartości złączanych kolumn:

```
-- 1

SELECT COUNT(*) AS ZL1
FROM Milion
JOIN GeoTabela ON (Milion.liczba % 62) = GeoTabela.id_pietro;
```

Zapytanie 2 (2 ZL), którego celem jest złączenie syntetycznej tablicy miliona wyników ztabelą geochronologiczną w postaci znormalizowanej, reprezentowaną przez złączenia pięciu tabel

```
-- 2
SELECT COUNT(*) AS ZL2
FROM Milion
JOIN GeoPietro ON (Milion.liczba % 62) = 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 (3 ZG), którego celem jest złączenie syntetycznej tablicy miliona wyników z tabelą geochronologiczną w postaci zdenormalizowanej, przy czym złączenie jest wykonywane poprzez zagnieżdżenie skorelowane:

```
-- 3
SELECT COUNT(*) AS ZG3
FROM Milion
WHERE (Milion.liczba % 62) =
(SELECT id_pietro
FROM GeoTabela
WHERE (Milion.liczba % 62) = (id_pietro));
```

Zapytanie 4 (4 ZG), 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:

```sql
-- 4
SELECT COUNT(*) AS ZG4
FROM Milion
WHERE (Milion.liczba % 62) 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

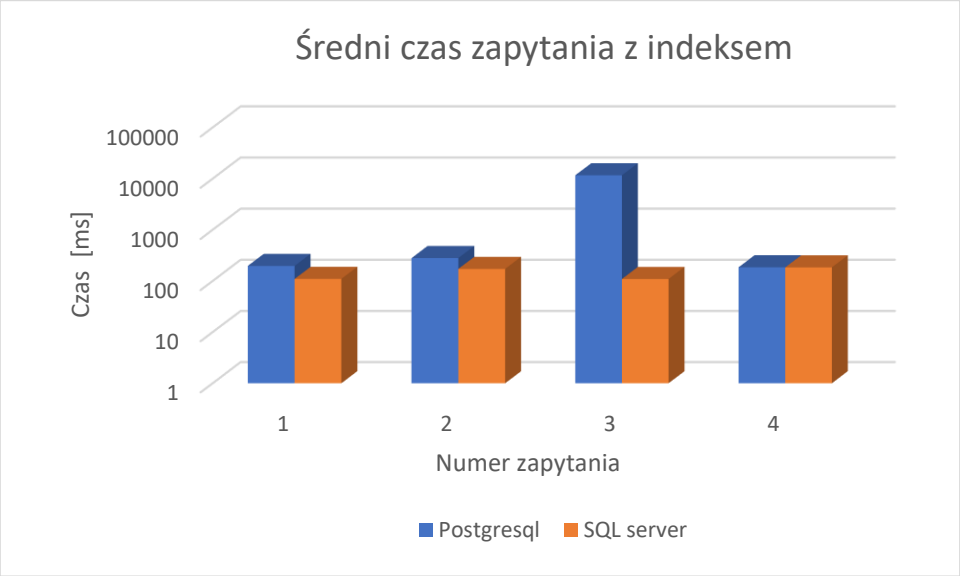
W zapytaniach testowych łączono dane z tabeli geochronologicznej z syntetycznymi danymi o rozkładzie jednostajnym z tabeli *Milion*, wypełnionej kolejnymi liczbami naturalnymi od 0 do 999 999. Aby otrzymać tabelę *Milion* wykonano dodatkową tabelę *Dziesiec* wypełnioną liczbami od 0 do 9, które po złączeniu i niewielkiej modyfikacji umożliwiły otrzymanie oczekiwanego wyniku- tabeli od 0 do 999 999.

```sql
CREATE TABLE Milion
(
liczba INTEGER NOT NULL PRIMARY KEY
);
WITH ID(number)
AS
(
 SELECT 1 AS number
 UNION ALL
 SELECT number + 1
 FROM ID
 WHERE number < 1000000
)
INSERT INTO Milion
```

# Wyniki

Wykonano pomiary czasu dla zapytania z indeksem i bez indeksu. Dla każdej bazy: PostgresSQL i SQL Server 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 program na wykonanie zadanego mu zapytania.

Poniżej znajdują się tabele z średnimi dla danego zapytania .



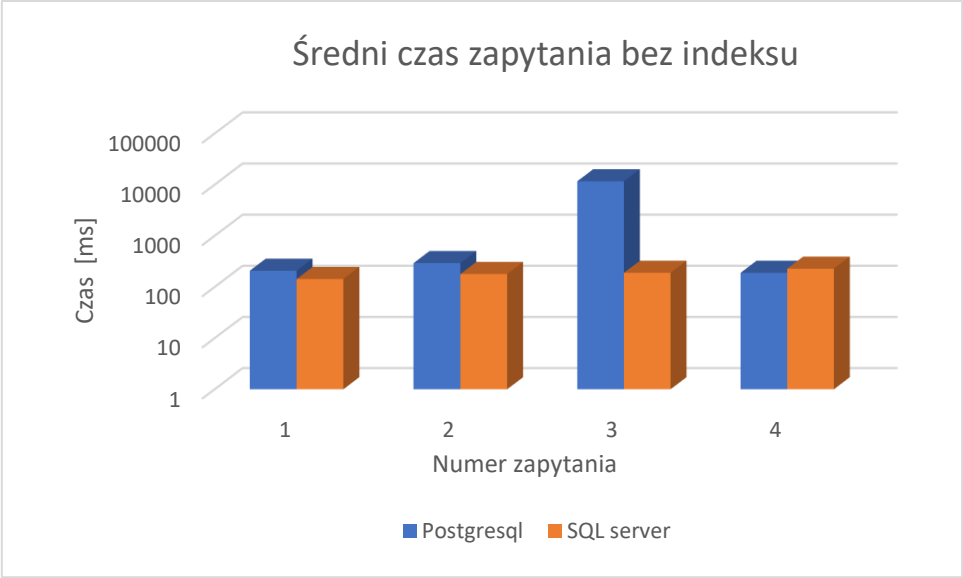

*Wykres 1 Średni czas zapytania dla wartości indeksowanych*

z indeksem (czas w [ms])

| 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 |

| bez indeksu (czas w [ms]) | | | | |
|---|---|---|---|---|
| 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 |

Z powyższych tabel bardzo łatwo można wywnioskować, że zapytania z indeksem są znacząco 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 znacząco 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.
- 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.