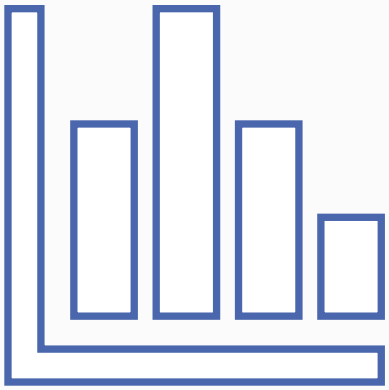
ĆWICZENIE 9

Wydajność złączeń i zagnieżdżeń skorelowanych dla systemów zarządzania bazami danych
SQL Server i PostgreSQL



Anna Piekarska

406700

Geoinformatyka

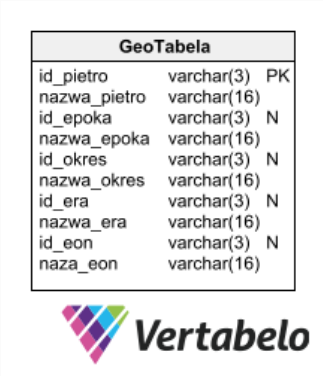
I. WPROWADZENIE

Ćwiczenie wykonan na podstawie artykułu Łukasza Jajeśnica i Adama Piórkowskiego (Akademia Górniczo – Hutnicza, Katedra Geoinformatyki i Informatyki Stosowanej) pt. “Wydajność złączeń i zagnieżdżeń dla schematów znormalizowanych i zdenormalizowanych”.

II. KONSTRUKCJA WYMIARU GEOCHRONOLOGICZNEGO

Tabela geochronologiczna obrazuje przebieg historii na podstawie następstwa procesów i warstw skalnych.

W celu przeprowadzenia testów została utworzona nowa baza danych – TabelaGeochronologiczna. W tabeli przedstawiono jednostki geochronologiczne w postaci tabel mających wymiar czasowy: eon, era, okres, epoka, piętro oraz odpowiadające im jednostki stratygraficzne. Tabela geochronologiczna została utworzona za pomocą złączenia tabel schematu znormalizowanego (schemat płatka śniegu). Konstrukcja została osiągnięta za pomocą utworzenia tabeli o nazwie GeoTabela (schemat gwiazdy), która zawiera wszystkie dane z powyższych tabel (rys. 1).



Rys. 1. Zdenormalizowany schemat tabeli geochronologicznej.

SQL Server / PostgreSQL

```
-- Zdenormalizowany schemat tabeli (schemat gwiazdy)
SELECT GeoPiętro.id_pietro, GeoPiętro.nazwa_pietro,
            GeoEpoka.id_epoka, GeoEpoka.nazwa_epoka,
            GeoOkres.id_okres,GeoOkres.nazwa_okres,
            GeoEra.id_era, GeoEra.nazwa_era,
            GeoEon.id_eon, GeoEon.nazwa_eon
INTO GeoTabela
FROM GeoEon
INNER JOIN GeoEra
On GeoEon.id_eon = GeoEra.id_eon
INNER JOIN GeoOkres
On GeoEra.id_era = GeoOkres.id_era
INNER JOIN GeoEpoka
On GeoOkres.id_okres = GeoEpoka.id_okres
```

```
INNER JOIN GeoPiętro
On GeoEpoka.id_epoka = GeoPiętro.id_epoka;

ALTER TABLE GeoTabela
      ADD PRIMARY KEY (id_pietro);
```

Następnym krokiem było przeprowadzenie testu wydajności porównującego wydajność złączeń oraz zapytań zagnieżdżonych wykonanych na tabelach o dużej ilości danych. Utworzona została tabela Dziesiec (wypełniona liczbami od 0 do 9), na podstawie której powstała tabela Milion, którą wypełniono kolejnymi liczbami naturalnymi od 1 do 999 999.

III. KONFIGURACJA SPRZĘTOWA

Wszystkie testy wykonano na komputerze o następujących parametrach:

- CPU: Intel(R) Core(TM) i5-10210U CPU @ 1.60GHz 2.11 GHz
- RAM: 8,00 GB (dostępne: 7,72 GB)
- S.O: Windows 11

Jako system zarządzania bazami danych wybrano oprogramowanie wolno dostępne:

- SQL Server, wersja v18.11.1
- PostgreSQL, wersja 6.7

IV. KRYTERIA TESTÓW

Wykonano zapytania sprawdzające wydajność złączeń i zagnieżdżeń z tabelą geochronologiczną w wersji zdenormalizowanej i znormalizowanej. Procedurę testową przeprowadzono w dwóch etapach:

- Zapytania bez nałożonych indeksów na kolumny danych,
- Zapytania z nałożonymi indeksami na wszystkie kolumny biorące udział w złączeniu.

Zastosowano cztery poniższe zapytania:

- Zapytanie 1 (1 ZL).

SQL Server:

```
SET STATISTICS TIME ON;
SELECT COUNT(*)
FROM Milion
INNER JOIN GeoTabela
ON Milion.liczba%68 = GeoTabela.id_pietro;
SET STATISTICS TIME OFF;
```

PostgreSQL:

```
SELECT COUNT(*)
FROM Milion
INNER JOIN GeoTabela
ON (mod(Milion.liczba,68) = (GeoTabela.id_pietro));
```

- Zapytanie 2 (2 ZL).

SQL Server:

```
SET STATISTICS TIME ON;
SELECT COUNT(*)
FROM Milion
INNER JOIN GeoPiętro
ON (Milion.liczba%68=GeoPiętro.id_pietro)
INNER JOIN GeoEpoka
ON GeoPiętro.id_epoka =GeoEpoka.id_epoka
INNER JOIN GeoOkres
ON GeoEpoka.id_okres = GeoOkres.id_okres
INNER JOIN GeoEra
ON GeoEra.id_era = GeoOkres.id_era
INNER JOIN GeoEon
ON GeoEon.id_eon = GeoEra.id_eon
SET STATISTICS TIME OFF;
```

PostgreSQL:

```
SELECT COUNT(*)
FROM Milion
INNER JOIN  GeoPiętro
ON (mod(Milion.liczba,68) = GeoPiętro.id_pietro)
NATURAL JOIN GeoEpoka
NATURAL JOIN GeoOkres
NATURAL JOIN GeoEra
NATURAL JOIN GeoEon;
```

- Zapytanie 3 (3 ZG).

SQL Server:

```
SET STATISTICS TIME ON;
SELECT COUNT(*)
FROM Milion
WHERE liczba%68 =
      (SELECT id_pietro
      FROM GeoTabela
      WHERE Milion.liczba%68=id_pietro)
SET STATISTICS TIME OFF;
```

PostgreSQL:

```
SELECT COUNT(*)
FROM Milion
WHERE mod(Milion.liczba,68) = (SELECT id_pietro
                               FROM GeoTabela
                               WHERE mod(Milion.liczba,68) = (id_pietro));
```

- Zapytanie 4 (4 ZG).

SQL Server:

```
SET STATISTICS TIME ON;
SELECT COUNT(*)
FROM Milion
WHERE Milion.liczba%68 IN (SELECT GeoPiętro.id_pietro
                           FROM GeoPiętro
                           INNER JOIN GeoEpoka
                           ON GeoPiętro.id_epoka = GeoEpoka.id_epoka
                           INNER JOIN GeoOkres
                           ON GeoEpoka.id_okres = GeoOkres.id_okres
                           INNER JOIN GeoEra
                           ON GeoEra.id_era = GeoOkres.id_era
                           INNER JOIN GeoEon
                          ON GeoEon.id_eon = GeoEra.id_eon);
```

PostgreSQL:

```
SELECT COUNT(*)
FROM Milion
WHERE mod(Milion.liczba,68) IN (SELECT GeoPiętro.id_pietro
                                FROM GeoPiętro
                                NATURAL JOIN GeoEpoka
                                NATURAL JOIN GeoOkres
                                NATURAL JOIN GeoEra
                                NATURAL JOIN GeoEon);
```

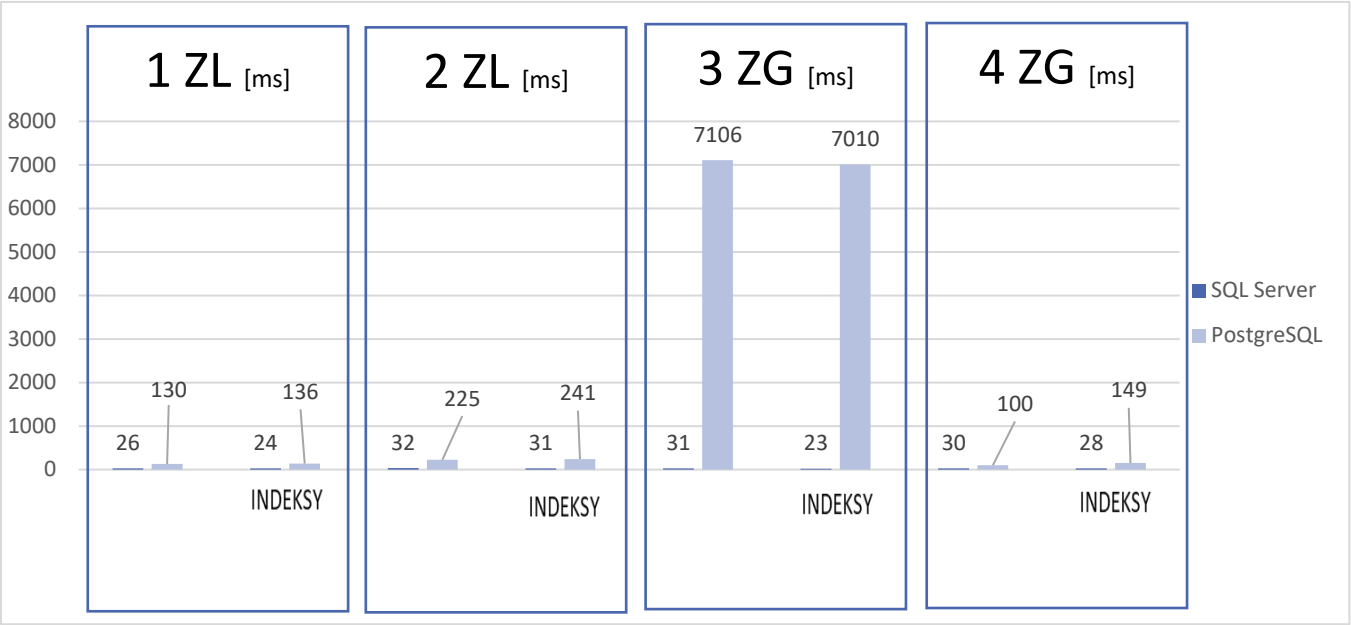
V. WYNIKI

Każdy test przeprowadzono wielokrotnie, wyniki skrajne pominięto. Minimalny, maksymalny i średni czas wykonywania zapytań spisano do tabeli:

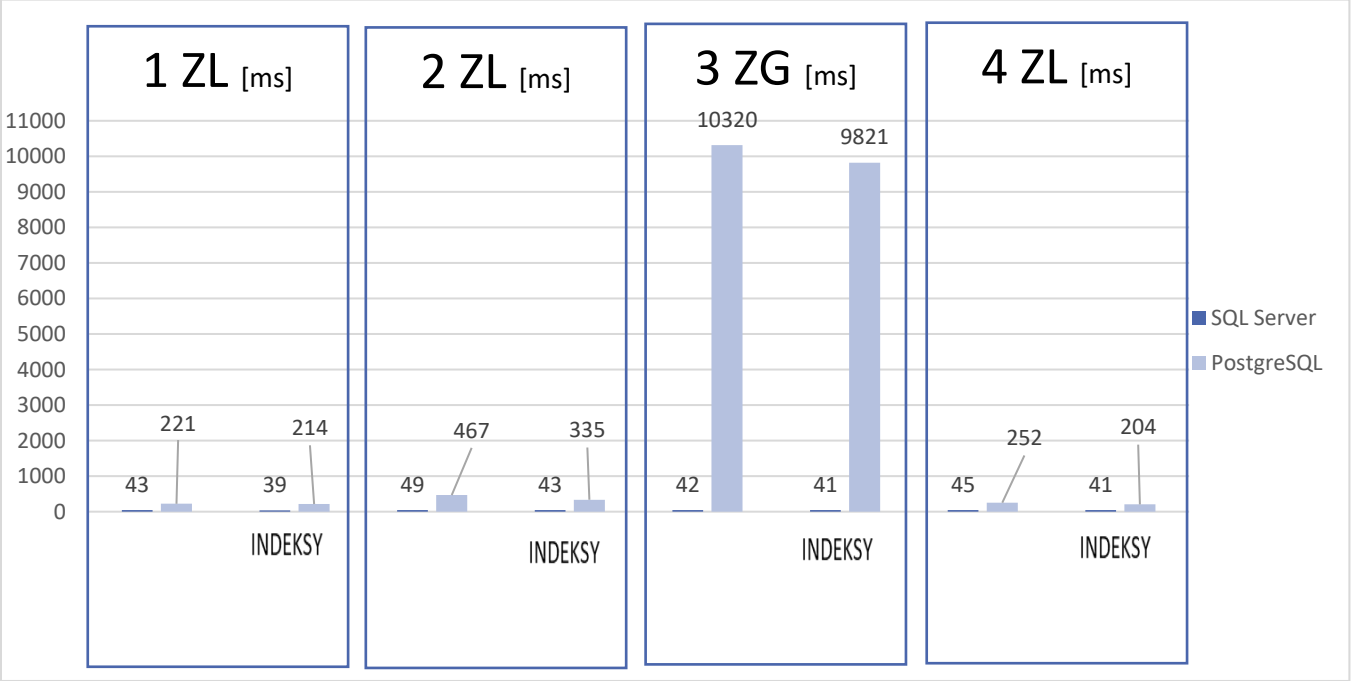
	1 ZL [ms]			2 ZL [ms]			3 ZG [ms]			4 ZG [ms]		
	MIN	MAX	AVG	MIN	MAX	AVG	MIN	MAX	AVG	MIN	MAX	AVG
BEZ INDEKSÓW												
SGL Server	26	43	38.38	32	49	40.87	31	42	36.06	30	45	39.95
PostgreSQL	130	221	166.64	225	467	355.52	7106	10320	8146.17	100	252	176.61
Z INDEKSAMI												
SGL Server	24	39	28.91	31	43	33.97	23	41	28.55	28	41	31.04
PostgreSQL	136	214	162.64	241	335	270.05	7010	9821	8079.86	149	204	169.02

Można zauważyć, że czas wykonania zapytań w PostgreSQL jest dłuższy niż w SQL Server. Ponadto dodanie indeksów spowodowało, że czas w obu systemach zarządzania bazami danych uległ skróceniu.

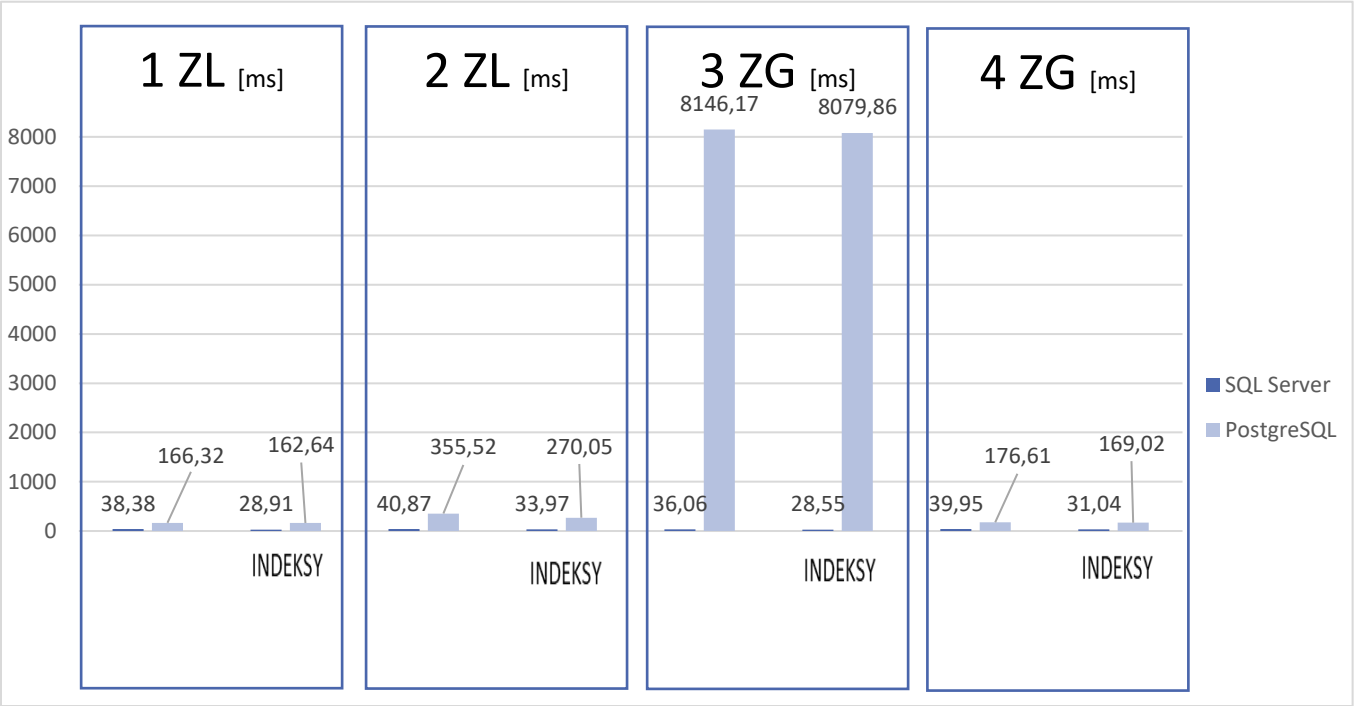
WYKRES PRZEDSTAWIAJĄCY MINIMALNY CZASY ZAPYTAŃ W SQL SERVER ORAZ W POSTGRESQL DLA TABEL BEZ INDEKSÓW I Z INDEKSAMI



WYKRES PRZEDSTAWIAJĄCY MAKSYMALNE CZASY ZAPYTAŃ W SQL SERVER ORAZ W POSTGRESQL DLA TABEL BEZ INDEKSÓW I Z INDEKSAMI



WYKRES PRZEDSTAWIAJĄCY ŚREDNI CZAS ZAPYTAŃ W SQL SERVER ORAZ W POSTGRESQL DLA TABEL BEZ INDEKSÓW I Z INDEKSAMI



VI. WNIOSKI

OTRZYMANE WYNIKI POZWALAJĄ WYCIĄGNĄĆ NASTĘPUJĄCE WNIOSKI

Teza artykułu:

- W większości przypadków wydajniejsza jest postać zdenormalizowana.
- Postać znormalizowana wykonywana jest w krótszym czasie tylko w jednym przypadku – gdy jest to zagnieżdżenie skorelowane (w podzapytaniu wewnętrznym w wersji zdenormalizowanej dokonywany jest odczyt dużej tabeli danych niezaindeksowanych).

Dodatkowe spostrzeżenia wynikające z przeprowadzonych testów:

- Złączenia i zagnieżdżenia skorelowane w SQL Server wykazują podobny czas.
- Znaczna różnica wystąpiła podczas 3 ZG, realizacja zapytania w PostgreSQL zajęła najwięcej czasu.

- Użycie indeksów w obu systemach zarządzania bazami danych we wszystkich rozważanych przypadkach przyśpieszyło wykonanie zapytań, zarówno złączeń jaki i zagnieżdżeń korelowanych.
- System PostgreSQL dokonuje analizy tabeli w związku z czym indeksacja nie ma większego wpływu na wykonania przedstawionych złączeń i zagnieżdżeń.
- Systemy zarządzania bazami danych wykazują różnice w czasie, w których zapytania były realizowane – zarówno złączenia i zagnieżdżenia (bez indeksów jak i z zastosowanymi indeksami) były szybciej wykonywane przez SQL Server. Największa różnica wystąpiła podczas 3 ZG – czas w SQL Server był ponda 200 razy krótszy, niż w PostgreSQL.

Podsumowaniem rozważań jest wniosek, iż dodanie indeksów powoduje skrócenie czasu realizacji zapytań.