25.05.2023, Kraków

Akademia Górniczo-Hutnicza im. Stanisława Staszica
w Krakowie

WYDAJNOŚĆ ZŁĄCZEŃ I ZAGNIEŻDŻEŃ DLA SCHEMATÓW ZNORMALIZOWANYCH I ZDENORMALIZOWANYCH

Anna Staniszewska

Nr albumu: 411344

Wstęp

Cel pracy stanowi porównanie wydajności przeprowadzanych operacji: złączeń i zagnieżdżeń dla danych znormalizowanych i zdenormalizowanych. Przedmiotem analizy są dane z tabeli stratygraficznej.

W badaniu szybkości wykonywanych zapytań wykorzystano dwa systemy bazodanowe – MySQL i PostgreSQL.

Konfiguracja sprzętowa i programowa

Każdy test przeprowadzono na komputerze o danych parametrach:

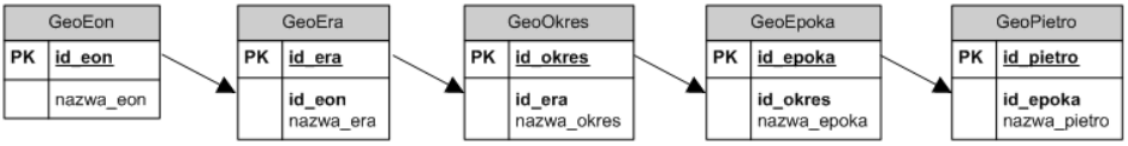
- CPU: AMD Ryzen 7 3700X 8-Core, 3.60 GHz
- RAM: 16 GB
- SSD: KINGSTON SA2000M8500G
- S.O.: Windows 10 Pro

Do analizy użyto następujące wersje systemów zarządzającymi bazami danych:

- MySQL Community Server 8.0.33
- PostgreSQL 15.3-1

Przygotowanie danych

Przed rozpoczęciem testów wydajności utworzono tabelę geochronologiczną[1] w postaci znormalizowanej (rys. 1) poprzez stworzenie tabel dla pięciu jednostek geologicznych – eonów, er, okresów, epok i pięter. Następnie każdą z nich wypełniono odpowiednimi wartościami.



Rys. 1. Tabela stratygraficzna w postaci znormalizowanej [2].

Schemat zdenormalizowany skonstruowano poprzez złączenie naturalne tabel powstałych w poprzednim kroku widoczne na rys. 2. Klucz główny ustawiono na kolumnę id_pietro.

```
CREATE TABLE projekt.geotabela AS (SELECT * FROM projekt.geopietro
NATURAL JOIN projekt.geoepoka NATURAL JOIN projekt.geookres
NATURAL JOIN projekt.geoera NATURAL JOIN projekt.geoeon );
ALTER TABLE projekt.geotabela ADD PRIMARY KEY(id_pietro);
```

Rys. 2. Utworzenie tabeli stratygraficznej w formie zdenormalizowanej.

Testy wydajnościowe

Wydajność schematów znormalizowanych i zdenormalizowanych sprawdzano za pomocą czterech zapytań złączeń i zagnieżdżeń, które łączyły dane z tabeli stratygraficznej z tabelą Milion, posiadającą wartości z rozkładu równomiernego od 0 do 999 999. Utworzenie tabeli Dziesiec (rys. 3), zawierającej cyfry od 0 do 9, pomogło w skonstruowaniu tabeli Milion (rys. 4).

```
CREATE TABLE projekt.dziesiec(cyfra int, bit int);
INSERT INTO projekt.dziesiec VALUES (0,1);
```

Rys. 3. Utworzenie tabeli Dziesiec.

```
CREATE TABLE projekt.milion(liczba int,cyfra int, bit int);
INSERT INTO projekt.milion SELECT a1.cyfra + 10* a2.cyfra + 100*a3.cyfra + 1000*a4.cyfra +
10000*a5.cyfra + 10000*a6.cyfra AS liczba, a1.cyfra AS cyfra, a1.bit AS bit
FROM projekt.dziesiec a1, projekt.dziesiec a2, projekt.dziesiec a3, projekt.dziesiec a4,
projekt.dziesiec a5, projekt.dziesiec a6;
```

Rys. 4. Utworzenie tabeli Milion.

Testy obejmowały dwa etapy:

- pierwszy bez nałożonych indeksów na tabele (jedynymi indeksami były te utworzone dla kolumn, będącymi kluczami głównymi),
- drugi obejmował indeksy nałożone na każdą kolumnę, która brała udział w złączeniach.

W MySQL indeksy dla kluczy głównych automatycznie dodano przy ich utworzeniu. Natomiast, w PostgreSQL należało je nałożyć ręcznie (rys. 5).

```
CREATE INDEX ieon ON projekt.geoeon(id_eon);
CREATE INDEX iera ON projekt.geoera(id_era);
CREATE INDEX iokres ON projekt.geookres(id_okres);
CREATE INDEX iepoka ON projekt.geoepoka(id_epoka);
CREATE INDEX ipietro ON projekt.geopietro(id_pietro);
CREATE INDEX ipietro_t ON projekt.geotabela(id_pietro);
```

Rys. 5. Nałożenie indeksów na klucze główne poszczególnych tabel.

W drugim etapie nałożono indeksy także na pozostałe kolumny, biorące udział w złączeniach (rys. 6).

```
CREATE INDEX ieon_nazwa ON projekt.geoeon(nazwa_eon);
CREATE INDEX iera_nazwa ON projekt.geoera(nazwa_era);
CREATE INDEX ieon_fk ON projekt.geoera(id_eon);
CREATE INDEX iokres_nazwa ON projekt.geookres(nazwa_okres);
CREATE INDEX iera_fk ON projekt.geookres(id_era);
CREATE INDEX iepoka_nazwa ON projekt.geoepoka(nazwa_epoka);
CREATE INDEX iokres_fk ON projekt.geoepoka(id_okres);
CREATE INDEX ipietro_nazwa ON projekt.geopietro(nazwa_pietro);
CREATE INDEX iepoka_fk ON projekt.geopietro(id_epoka);
CREATE INDEX iliczba ON projekt.milion(liczba);
CREATE INDEX icyfra ON projekt.milion(cyfra);
CREATE INDEX ibit ON projekt.milion(bit);
CREATE INDEX itabela ON projekt.geotabela(id_eon, id_era, id_okres, id_epoka,
nazwa_eon, nazwa_era, nazwa_okres, nazwa_epoka, nazwa_pietro);
```

Rys. 6. Nałożenie indeksów na pozostałe kolumny.

Dla każdego etapu odczytano minimalny i średni czas wykonywania się czterech zapytań:

- 1 ZL – połączenie syntetycznej tabeli zawierającej milion rekordów ze zdenormalizowaną tabelą geochronologiczną. W tym złączeniu zastosowano operację modulo, która dopasowuje zakresy wartości w kolumnach podczas warunku łączenia (rys. 7).

```
SELECT COUNT(*) FROM projekt.milion m INNER JOIN projekt.geotabela t
ON mod(m.liczba,76)=(t.id_pietro);
```

Rys. 7. Zapytanie 1.

- 2 ZL – połączenie syntetycznej tabeli zawierającej milion rekordów ze znormalizowaną tabelą geochronologiczną (rys. 8).

```
SELECT COUNT(*) FROM projekt.milion m INNER JOIN projekt.geopietro p ON
(mod(m.liczba,76)=p.id_pietro) NATURAL JOIN projekt.geoepoka NATURAL JOIN
projekt.geookres NATURAL JOIN projekt.geoera NATURAL JOIN projekt.geoeon;
```

Rys. 8. Zapytanie 2.

- 3 ZG – połączenie syntetycznej tabeli zawierającej milion rekordów ze zdenormalizowaną tabelą geochronologiczną. Złączenie wykonano za pomocą zagnieżdżenia skorelowanego (rys. 9).

```
SELECT COUNT(*) FROM projekt.milion m WHERE
mod(m.liczba,76) = (SELECT id_pietro FROM projekt.geotabela
WHERE mod(m.liczba,76) = (id_pietro));
```

Rys. 9. Zapytanie 3.

- 4 ZG – połączenie syntetycznej tabeli zawierającej milion rekordów ze znormalizowaną tabelą geochronologiczną. Złączenie wykonano za pomocą zagnieżdżenia skorelowanego, a zapytanie wewnętrzne było złączeniem tabel danych jednostek stratygraficznych (rys. 10).

```
SELECT COUNT(*) FROM projekt.milion m WHERE mod(m.liczba,76) IN
(SELECT projekt.geopietro.id_pietro FROM projekt.geopietro NATURAL JOIN projekt.geoepoka NATURAL JOIN
projekt.geookres NATURAL JOIN projekt.geoera NATURAL JOIN projekt.geoeon);
```

Rys. 10. Zapytanie 4.

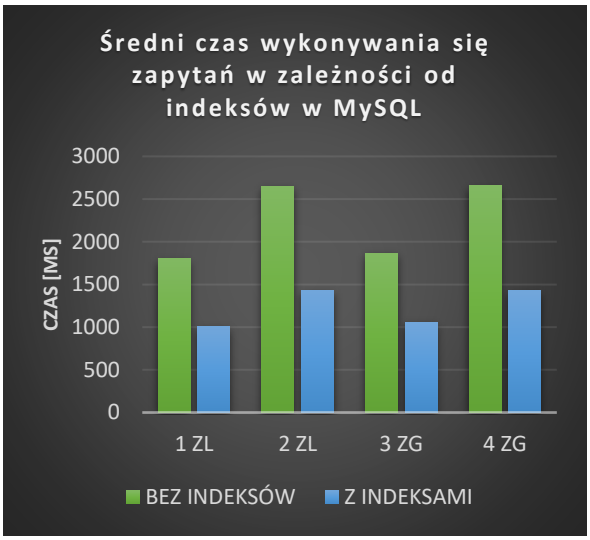
Wyniki

Każdy test przeprowadzono kilka razy, z czego wyciągnięto średnią i wartość minimalną. Wyniki testów zaprezentowano w tabeli 1.

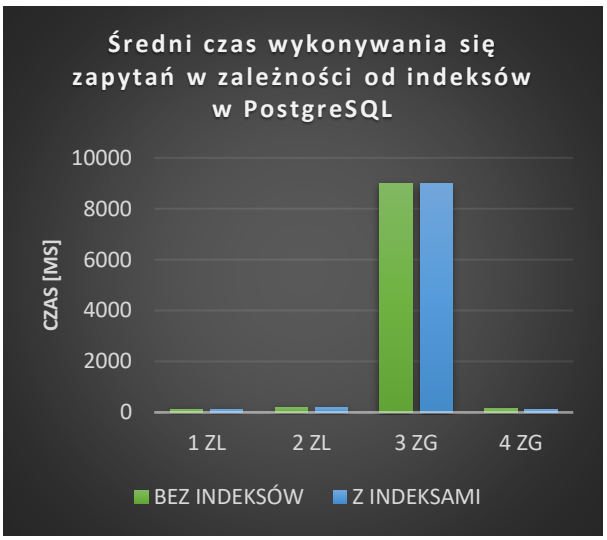
Tab. 1. Wyniki pomiarów.

	1 ZL		2 ZL		3 ZG		4 ZG	
BEZ INDEKSÓW	**MIN**	**ŚR**	**MIN**	**ŚR**	**MIN**	**ŚR**	**MIN**	**ŚR**
MySQL	1797	1803	2610	2647	1859	1864	2625	2658
PostgreSQL	117	123	193	199	8927	9009	117	137
Z INDEKSAMI								
MySQL	1000	1006	1421	1430	1047	1058	1406	1425
PostgreSQL	115	118	190	194	8925	9009	118	124

Porównanie wydajności dla dwóch etapów (bez lub z indeksami) testów przedstawiono na wykresach 1, 2.

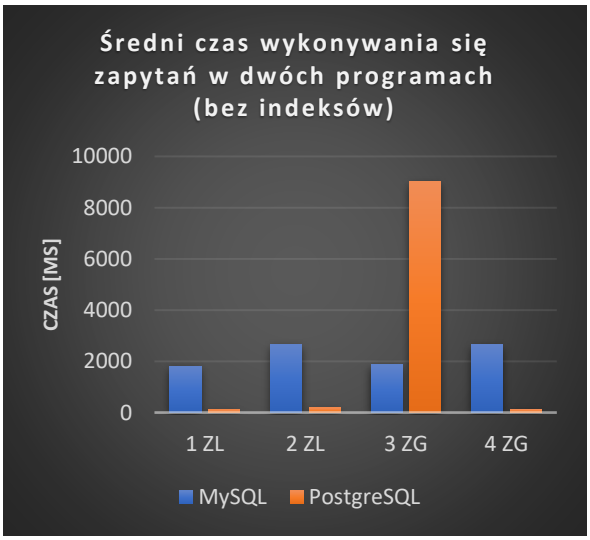


Wykres 1.

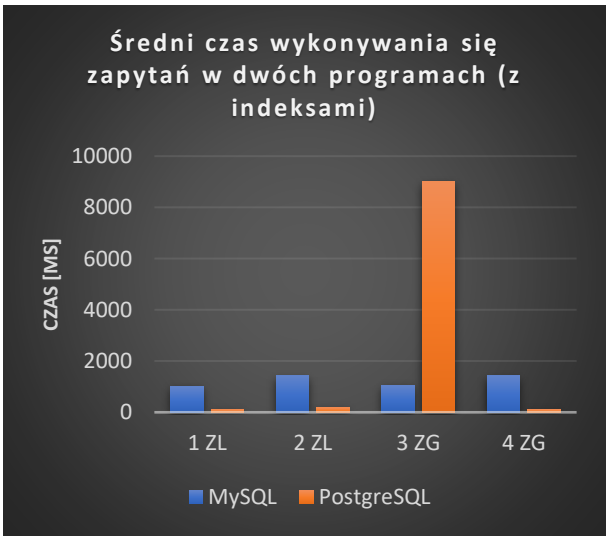


Wykres 2.

Poniżej na wykresach 3, 4 ukazano różnice pomiędzy wydajnością oprogramowań – MySQL i PostgreSQL.

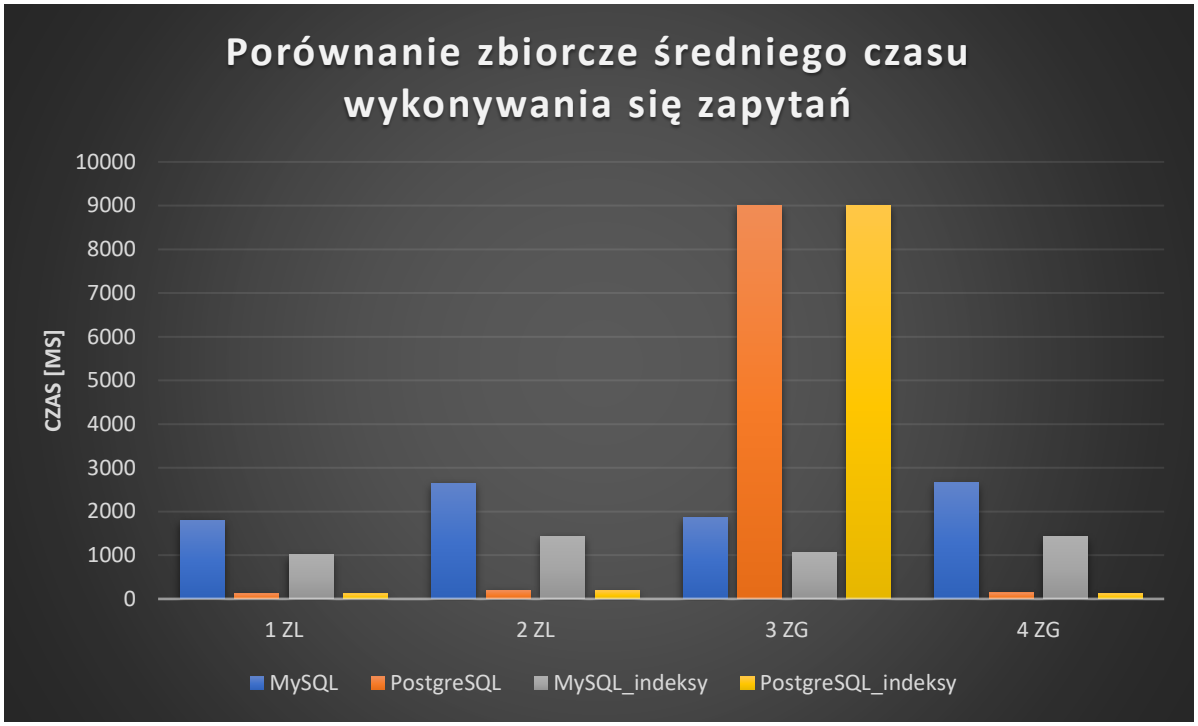


Wykres 3.



Wykres 4.

Dla lepszej analizy wyników wszystkie pomiary zestawiono na wykresie zbiorczym (wyk. 5).



Wykres 5.

Wnioski

Dla powyższych wyników można wyciągnąć następujące konkluzje:

- Z reguły postać zdenormalizowana jest szybsza niż znormalizowana.
- Jedynym przypadkiem, dla którego schemat znormalizowany cechuje się większą wydajnością, są zagnieżdżenia skorelowane w PostgreSQL. Dla wersji zdenormalizowanej uzyskano znacznie wyższe skrajne wartości pomiarów.
- Dodanie indeksów dla wszystkich danych zmniejszyło czas wykonywania się zapytań tylko w MySQL. W PostgreSQL te wartości pozostały bez zmian.
- Generalnie PostgreSQL jest znacznie szybszy od oprogramowania MySQL z wyjątkiem zagnieżdżeń skorelowanych, gdzie pierwszy system ma bardzo małą wydajność.

Podsumowując powyższą analizę, można stwierdzić, że w większości przypadkach normalizacja generuje spadek wydajności.

Bibliografia

[1] *„Tabela stratygraficzna”* ICS, kwiecień 2023 https://pl.wikipedia.org/wiki/Tabela_stratygraficzna
[2] *„WYDAJNOŚĆ ZŁĄCZEŃ I ZAGNIEŻDŻEŃ DLA SCHEMATÓW ZNORMALIZOWANYCH I ZDENORMALIZOWANYCH”*, Jaješnica Ł., Piórkowski A.