

Kraków, 24.05.2023



**AKADEMIA GÓRNICZO-HUTNICZA
IM. STANISŁAWA STASZICA W KRAKOWIE**

**Wydział Geologii, Geofizyki
i Ochrony Środowiska**

Przedmiot Bazy Danych

Ćwiczenie nr 9

Geoinformatyka

Tomasz Dąbrowa

Nr albumu 411821

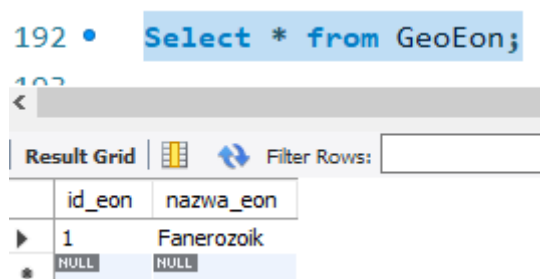
Wstęp

W poniższym sprawozdaniu zaprezentowano wyniki badań związane z czasem przetwarzania zapytań na bazach danych o dużej objętości danych. Posłużono się w tym celu następującymi serwerami baz danych – MySQL i PostgreSQL.



Przygotowanie baz

Najpierw przygotowano bazę geochronologia. Następnie utworzono odpowiednie, znormalizowane tabele.



```
192 • Select * from GeoEon;
```



	id_eon	nazwa_eon
▶	1	Fanerozoik
*	NULL	NULL

Rys. 1. Tabela GeoEon w MySQL

193 • `Select * from GeoEra;`

194

<

Result Grid |   Filter Rows:



	id_era	id_eon	nazwa_era
▶	1	1	Kenozoik
	2	1	Mezozoik
	3	1	Paleozoik
*	NULL	NULL	NULL

Rys. 2. Tabela GeoEra w MySQL

194 • `Select * from GeoOkres;`

195

<



Result Grid |   Filter Rows: | E

	id_okres	id_era	nazwa_okres
▶	1	1	Czwartorzęd
	2	1	Neogen
	3	1	Paleogen
	4	2	Kread
	5	2	Jura
	6	2	Trias
	7	3	Perm
	8	3	Karbon
	9	3	Dewon
*	NULL	NULL	NULL

Rys. 3. Tabela GeoOkres w MySQL

195 • `Select * from GeoEpoka;`

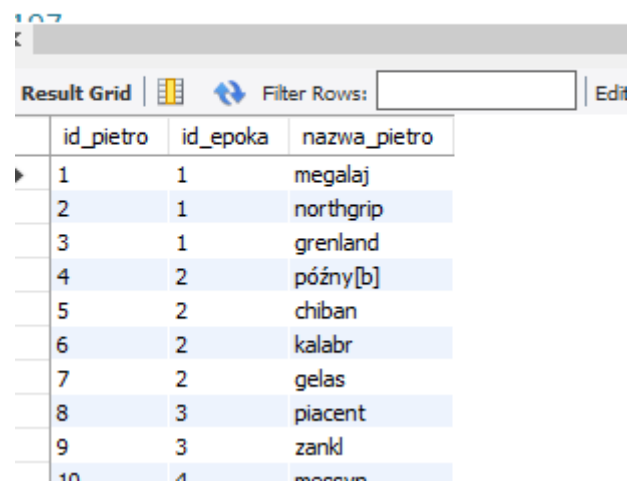
196

Result Grid |   Filter Rows:

	id_epoka	id_okres	nazwa_epoka
▶	1	1	Halocen
	2	1	Plejstocen
	3	2	Pilocen
	4	2	Miocen
	5	3	Oligocen
	6	3	Eocen
	7	3	Paleocen
	8	4	Gorna

Rys. 4. Tabela GeoEpoka w MySQL

196 • `Select * from GeoPietro;`

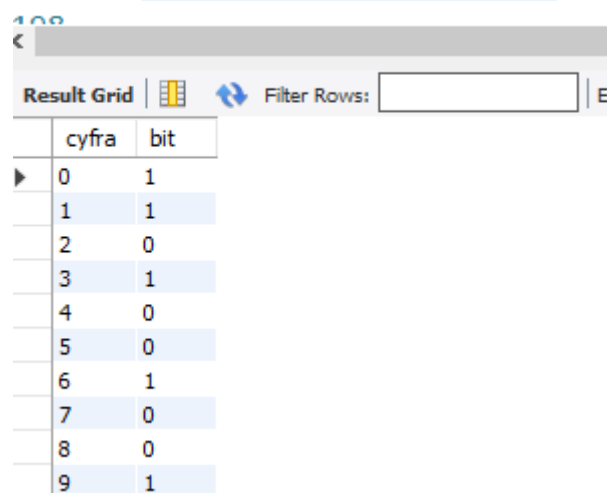


The screenshot shows a MySQL query result for the table 'GeoPietro'. The interface includes a 'Result Grid' tab, a 'Filter Rows' input field, and an 'Edit' button. The table has three columns: 'id_pietro', 'id_epoka', and 'nazwa_pietro'. There are 10 rows of data.

	id_pietro	id_epoka	nazwa_pietro
▶	1	1	megabaj
	2	1	northgrip
	3	1	grenland
	4	2	późny[b]
	5	2	chiban
	6	2	kalabr
	7	2	gelas
	8	3	piacent
	9	3	zankl
	10	4	mesur

Rys. 5. Tabela GeoPietro w MySQL

197 • `Select * from dziesięć;`



The screenshot shows a MySQL query result for the table 'dziesięć'. The interface includes a 'Result Grid' tab, a 'Filter Rows' input field, and an 'Edit' button. The table has two columns: 'cyfra' and 'bit'. There are 10 rows of data.

	cyfra	bit
▶	0	1
	1	1
	2	0
	3	1
	4	0
	5	0
	6	1
	7	0
	8	0
	9	1

Rys. 6. Tabela dziesięć z losowymi bitami od 0 do 1

Tabelę dziesięć przygotowano następująco:

```
CREATE TABLE Dziesięć
```

```
(
```

```
    cyfra INTEGER NOT NULL PRIMARY KEY,
```

```
    bit INTEGER NOT NULL
```

```
);
```

Przy ręcznym wpisaniu dziesięciu rekordów losowo przydzielano wartość 0 lub 1 do kolumny przechowującej bit.

04 • `select * from zdenormalizowana;`

Result Grid | Filter Rows: | Export: | Wrap Cell Content: [fA](#)

	id_eon	id_era	id_okres	id_epoka	id_pietro	nazwa_pietro	nazwa_epoka	nazwa_okres	nazwa_era	nazwa_eon
1	1	1	1	1	1	megalaj	Halocen	Czwartorzęd	Kenozoik	Fanerozoik
1	1	1	1	1	2	northgrip	Halocen	Czwartorzęd	Kenozoik	Fanerozoik
1	1	1	1	1	3	grenland	Halocen	Czwartorzęd	Kenozoik	Fanerozoik
1	1	1	1	2	4	późny[b]	Plejstocen	Czwartorzęd	Kenozoik	Fanerozoik
1	1	1	1	2	5	chiban	Plejstocen	Czwartorzęd	Kenozoik	Fanerozoik
1	1	1	1	2	6	kalabr	Plejstocen	Czwartorzęd	Kenozoik	Fanerozoik
1	1	1	1	2	7	gelas	Plejstocen	Czwartorzęd	Kenozoik	Fanerozoik
1	1	2	2	3	8	piacent	Pilocen	Neogen	Kenozoik	Fanerozoik
1	1	2	2	3	9	zankl	Pilocen	Neogen	Kenozoik	Fanerozoik
1	1	2	2	4	10	messyn	Miocen	Neogen	Kenozoik	Fanerozoik
1	1	2	2	4	11	torton	Miocen	Neogen	Kenozoik	Fanerozoik
1	1	2	2	4	12	serrawal	Miocen	Neogen	Kenozoik	Fanerozoik
1	1	2	2	4	13	lang	Miocen	Neogen	Kenozoik	Fanerozoik
1	1	2	2	4	14	burdygał	Miocen	Neogen	Kenozoik	Fanerozoik

Rys. 7. Zdenormalizowana tabela w MySQL

Zdenormalizowaną GeoTabelę przygotowano w oparciu o następującą kwerendę:

```
SELECT GeoPietro.id_pietro, GeoPietro.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 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;
ALTER TABLE GeoTabela
ADD PRIMARY KEY (id_pietro);
```

Przygotowano również tabelę milion według polecenia:

```
CREATE TABLE milion(liczba int,cyfra int, bit int);
INSERT INTO milion SELECT a1.cyfra +10* a2.cyfra +100*a3.cyfra + 1000*a4.cyfra
+ 10000*a5.cyfra + 100000*a6.cyfra AS liczba , a1.cyfra AS cyfra, a1.bit AS bit
FROM dziesiec a1, dziesiec a2, dziesiec a3, dziesiec a4, dziesiec a5, dziesiec a6;
```

W obu przypadkach należało jeszcze nałożyć indeksy w celu porównania czasu wykonywania kwerend. Wykonano to w następujący sposób

```
CREATE UNIQUE INDEX liczba_index ON Milion (liczba);  
CREATE UNIQUE INDEX id_pietro_index ON GeoTabela (id_pietro);  
CREATE UNIQUE INDEX id_eon_index ON GeoEon (id_eon);  
CREATE UNIQUE INDEX id_era_index ON GeoEra (id_era);  
CREATE UNIQUE INDEX id_okres_index ON GeoOkres (id_okres);  
CREATE UNIQUE INDEX id_epoka_index ON GeoEpoka (id_epoka);  
CREATE UNIQUE INDEX id_pietroG_index ON GeoPietro (id_pietro);
```

Konfiguracja sprzętowa

Sprzęt

- Procesor: Intel Core i7-6500U CPU 2,50 GHz
- Pamięć RAM: DDR3 16 GB
- Dysk twardy: Samsung SSD 850 EVO 500 GB
- Karta graficzna: AMD Radeon R5 M330
- System operacyjny: Windows 10 Home

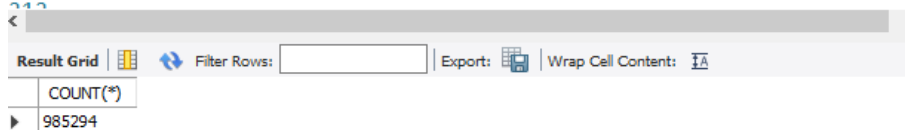
Oprogramowanie

- MySQL Server 8.0.33
- MySQL Workbench 8.0.33
- PostgreSQL 15
- pgAdmin 4

Testy

MySQL

```
207 -- ZAPYTANIE 1 ZL  
208 • SELECT COUNT(*) FROM milion INNER JOIN zdenormalizowana ON  
209 (mod(milion.liczba,68)=(zdenormalizowana.id_pietro));  
210  
211  
212
```



Result Grid
COUNT(*)
985294

Rys. 8. Zapytanie 1 ZL w MySQL

```

211 -- ZAPYTANIE 2 ZL
212
213 • SELECT COUNT(*) FROM milion INNER JOIN GeoPietro ON
214 (mod(milion.liczba,68)=GeoPietro.id_pietro) NATURAL JOIN GeoEpoka NATURAL JOIN
215 GeoOkres NATURAL JOIN GeoEra NATURAL JOIN GeoEon;
216
217

```



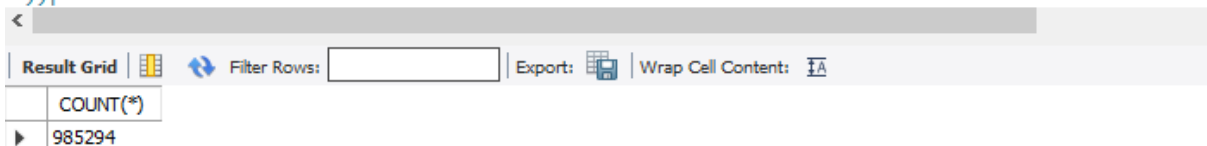
COUNT(*)
985294

Rys. 9. Zapytanie 2 ZL w MySQL

```

217 -- ZAPYTANIE 3 ZG
218 • SELECT COUNT(*) FROM milion WHERE mod(milion.liczba,68)=
219 (SELECT id_pietro FROM zdenormalizowana WHERE mod(milion.liczba,68)=(id_pietro));
220
221

```




COUNT(*)
985294

Rys. 10. Wynik zapytania 3 ZG w MySQL

```

221 -- ZAPYTANIE 4 ZG
222 • SELECT COUNT(*) FROM milion WHERE mod(milion.liczba,68)
223 IN (SELECT geopietro.id_pietro FROM geopietro
224 NATURAL JOIN geoepoka NATURAL JOIN
225 geookres NATURAL JOIN geoera NATURAL JOIN geoeon);
226

```



COUNT(*)
985294

Rys. 11. Wynik zapytania 4 ZG w MySQL

PostgreSQL

```
224 -- 1 ZL
225 SELECT COUNT(*) FROM milion INNER JOIN GeoTabela ON
226 (mod(Milion.liczba,68)=(GeoTabela.id_pietro));
227
228
229
```

Data Output Messages Notifications

	count bigint	
1	911769	

Rys. 12. Zapytanie ZL 1 w PostgreSQL

```
28 -- 2 ZL
29 SELECT COUNT(*) FROM Milion INNER JOIN GeoPietro ON
30 (mod(Milion.liczba,68)=GeoPietro.id_pietro) NATURAL JOIN GeoEpoka NATURAL JOIN
31 GeoOkres NATURAL JOIN GeoEra NATURAL JOIN GeoEon;
32
33
34
```

Data Output Messages Notifications

	count bigint	
	911769	

Rys. 13. Zapytanie 2 ZL w PostgreSQL

```
232
233 -- 3 ZG
234 SELECT COUNT(*) FROM Milion WHERE mod(Milion.liczba,68)=
235 (SELECT id_pietro FROM GeoTabela WHERE mod(Milion.liczba,68)=(id_pietro));
236
237
```

Data Output Messages Notifications

	count bigint	
1	911769	

Rys. 14. Zapytanie 3 ZG w PostgreSQL


```

237 -- 4 ZG
238 SELECT COUNT(*) FROM Milion WHERE mod(Milion.liczba,68)
239 IN (SELECT GeoPietro.id_pietro FROM GeoPietro
240 NATURAL JOIN GeoEpoka NATURAL JOIN
241 GeoOkres NATURAL JOIN GeoEra NATURAL JOIN GeoEon);
242
243

```

Data Output Messages Notifications

count bigint

1	911769
---	--------

Rys. 15. Zapytanie 4 ZG w PostgreSQL

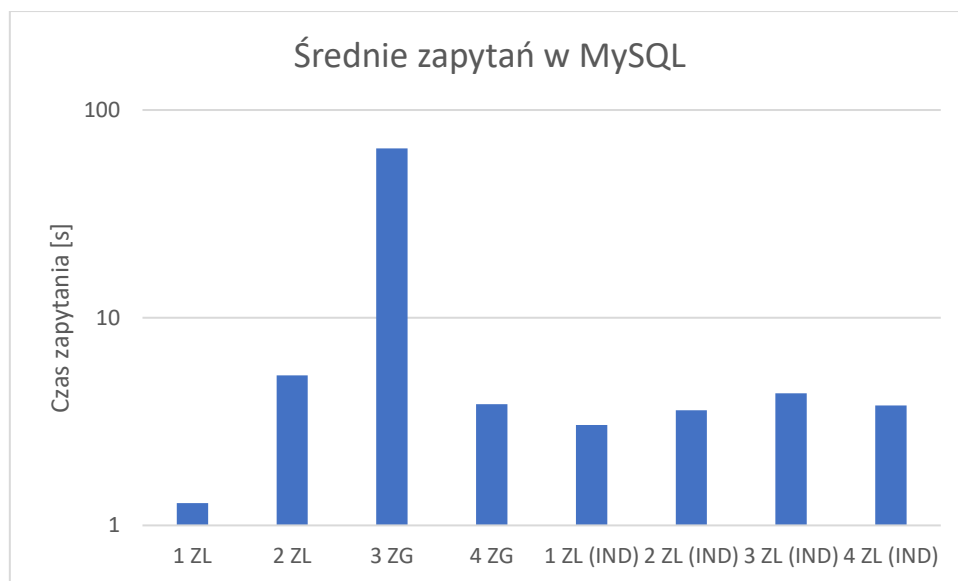
Wyniki i wnioski

W programie Microsoft Excel przetworzono dane związane z czasem zapytań, a następnie utworzono odpowiednie wykresy obrazujące wyniki. Dla każdej kwerendy wykonano cztery zapytania, z których liczono średnią oraz wartość minimalną.

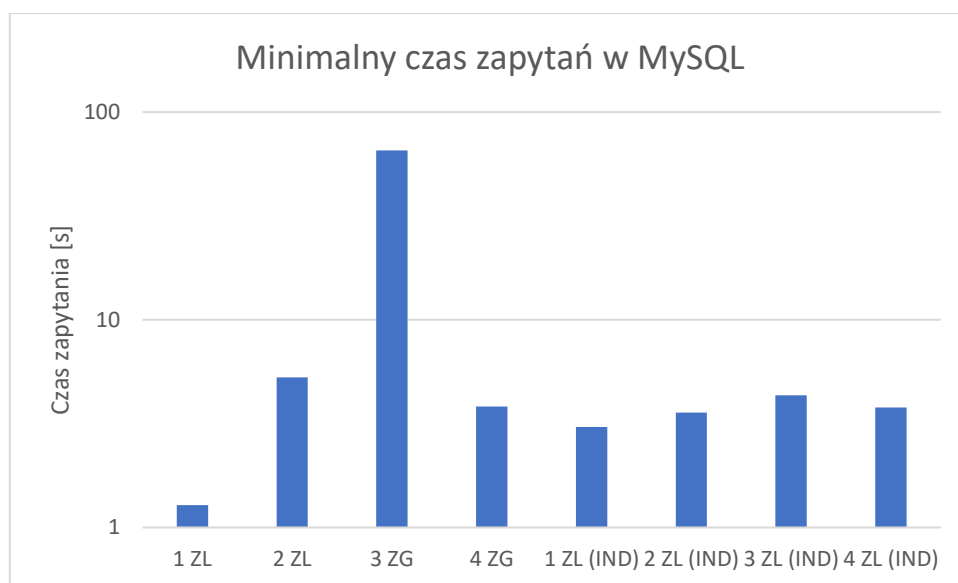
MySQL

	MySQL	Czas kwerendy [s]				Średnia	Min
Bez indeksów	Obszar wykresu	1,282	1,313	1,468	1,297	1,34	1,282
	2 ZL	5,36	7,171	5,712	5,28	5,88075	5,28
	3 ZG	68,047	67,25	65,297	66,813	66,85175	65,297
	4 ZG	3,828	3,906	4,734	4,047	4,12875	3,828
Z indeksami	1 ZL	3,268	3,047	3,047	3,125	3,12175	3,047
	2 ZL	4,125	3,578	3,875	3,625	3,80075	3,578
	3 ZL	4,328	4,859	5,203	4,5	4,7225	4,328
	4 ZL	3,906	3,781	3,906	4,36	3,98825	3,781

Tab. 1. Pomiary czasu wykonywania kwerend w MySQL



Wykres 1. Średnie zapytań w MySQL w skali logarytmicznej. Zapytania oznaczone jako (IND) to zapytania z indeksami



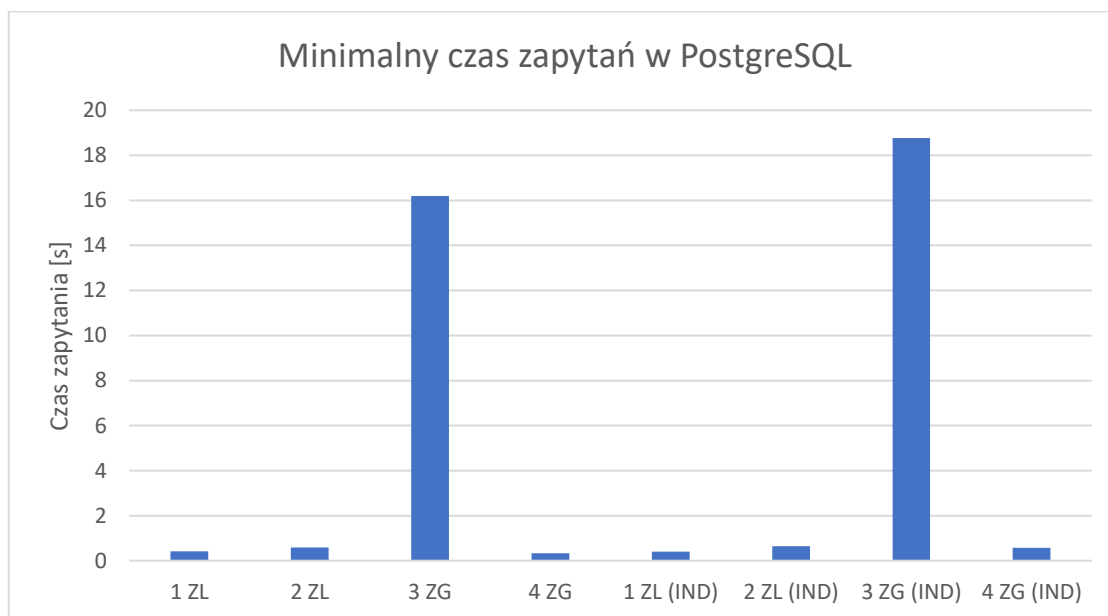
Wykres 2. Minimalny czas zapytań w MySQL w skali logarytmicznej. Oznaczenie (IND) to zapytania z indeksami

Na podstawie powyższych wykresów dotyczących kwerend w MySQL możemy zauważyć, że najbardziej czasochłonną akcją do wykonania jest zapytanie zagnieżdżone nr 3 operujące na tabeli z milionem rekordów. Ciekawą obserwacją jest fakt, że po zindeksowaniu tabel występują mniejsze amplitudy czasu wykonania kwerendy. Najkrótsze zapytania się wydłużają, z kolei najdłuższe skracają.

PostgreSQL

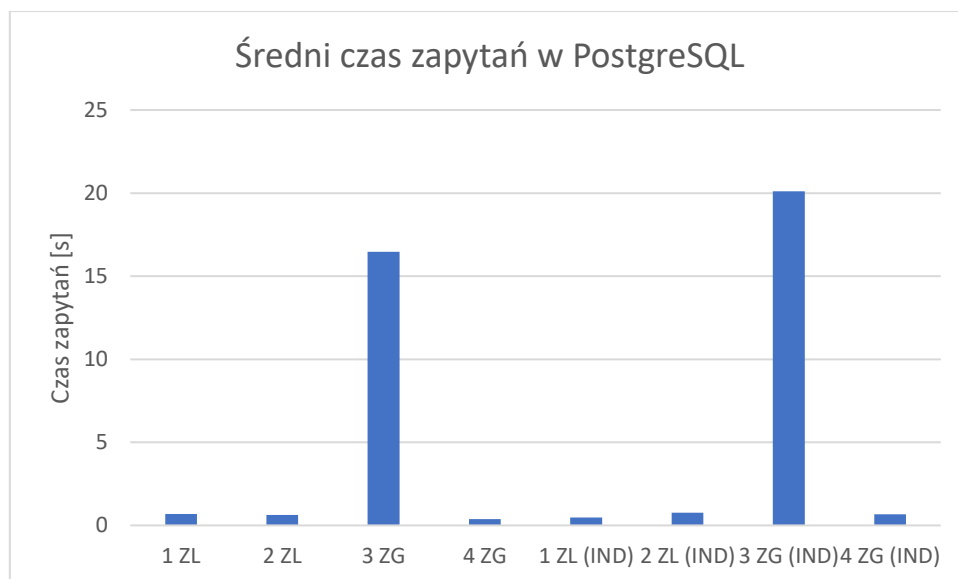
	PostgreSQL	Czas kwerendy [s]				Średnia	Min
Bez indeksów	1 ZL	0,412	0,588	0,43	1,348	0,6945	0,412
	2 ZL	0,587	0,582	0,659	0,702	0,6325	0,582
	3 ZG	16,195	16,797	16,347	16,543	16,4705	16,195
	4 ZG	0,429	0,328	0,369	0,385	0,37775	0,328
Z indeksami	1 ZL (IND)	0,563	0,404	0,464	0,439	0,4675	0,404
	2 ZL (IND)	0,824	0,825	0,745	0,642	0,759	0,642
	3 ZG (IND)	22,957	19,574	19,11	18,767	20,102	18,767
	4 ZG (IND)	0,73	0,806	0,571	0,57	0,66925	0,57

Tab. 2. Pomiar czasu wykonywania kwerendy w PostgreSQL



Wykres 3. Minimalny czas zapytań w PostgreSQL (dopisek IND oznacza obecność indeksów)

Na podstawie powyższego wykresu możliwe jest dostrzeżenie ogromnej dysproporcji w najmniejszych wartościach czasu wykonania kwerendy.

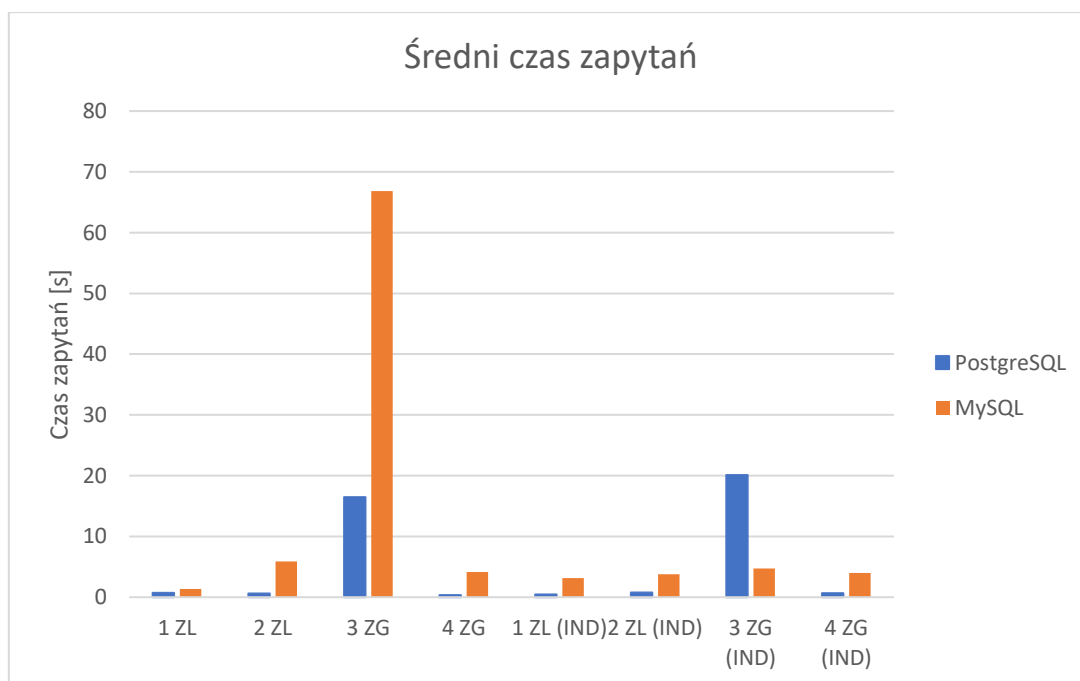


Wykres 4. Średni czas zapytań w PostgreSQL

Porównując średni czas zapytań w PostgreSQL możliwe jest zauważenie, że w przypadku zapytań łączonych indeksy skracają czas wykonywania kwerendy. Przeciwnie zjawisko można dostrzec w przypadku zapytań zagnieżdżonych. Indeksy w ich przypadku wydłużają czas zapytania.

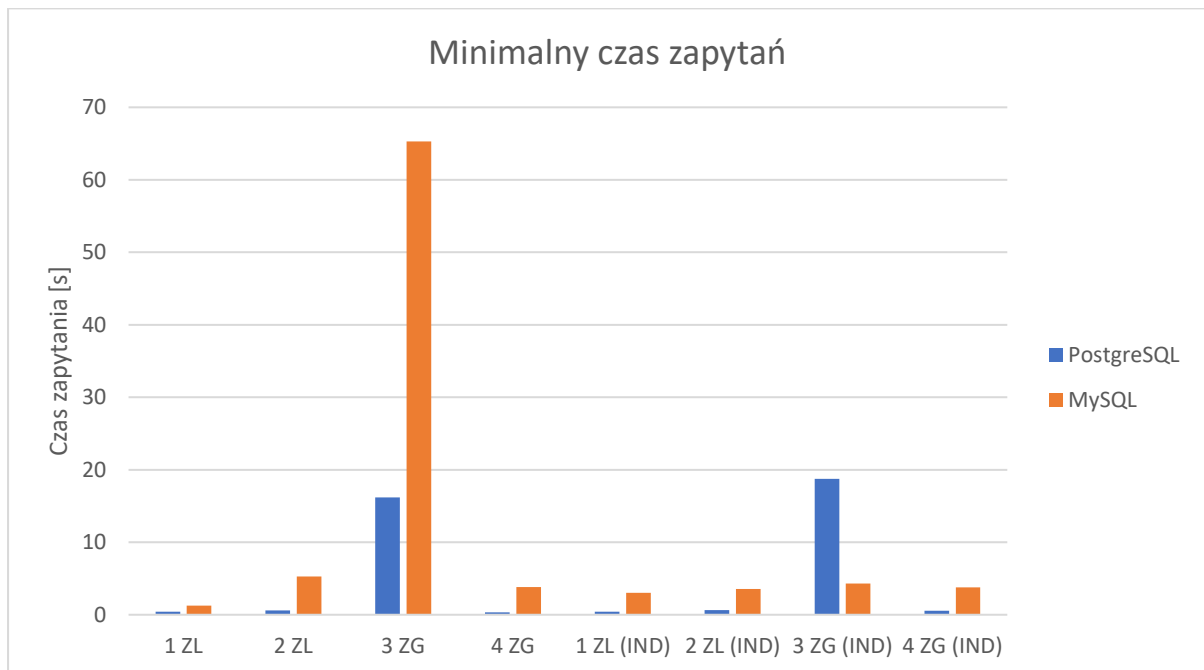
Porównanie MySQL i PostgreSQL

Na wykresach poniżej umieszczono porównanie czasu wykonywania kwerend w dwóch bazach danych: MySQL i PostgreSQL.



Wykres 5. Średni czas zapytań dla MySQL i PostgreSQL

Analizując powyższe dane można dojść do wniosku, że czas wykonania kwerend dla MySQL jest większy niż dla PostgreSQL. Wyjątkiem od tej reguły jest trzecie, indeksowane zapytanie zagnieżdżone. W tym przypadku PostgreSQL prezentuje się marniej od MySQL.



Wykres 6. Minimalny czas zapytań dla MySQL i PostgreSQL

Powyższy wykres przedstawia minimalne czasy zapytań dla obu baz, toteż obserwacje są analogiczne do powyższych. We wszystkich przypadkach PostgreSQL jest szybszy od MySQL za wyjątkiem trzeciego zapytania zagnieżdżonego z indeksem.

Analizując tylko trzecie zapytanie zagnieżdżone można dojść do wniosku, że indeksowanie tabel pozwala na przyspieszenie wykonania kwerendy wyłącznie w przypadku MySQL. W PostgreSQL sytuacja czasowa ulega pogorszeniu.