Analiza wydajności złączeń i zagnieżdżeń dla schematów znormalizowanych i zdenormalizowanych

Jakub Hempel, nr 405756

GEOINFORMATYKA, 2 rok

1. Cel

Analiza wydajności zapytań dla schematów znormalizowanych i zdenormalizowanych dla wybranych, ogólno dostępnych systemów zarządzania bazami danych. Zasadniczym celem testów jest ocena wpływu normalizacji na zapytania złożone.

Badanym obiektem była tabela geochronologiczna obrazująca przebieg historii Ziemi na podstawie następstwa procesów i warstw skalnych. Przeprowadzono na niej dwie analizy: bez indeksów oraz z ich użyciem.

2. Konfiguracja sprzętowa i programowa

Przedstawione testy zostały wykonane na laptopie o następujących parametrach:

- CPU: Intel® Core™ i7-9750H, 2,60 GHz
- RAM: 16 GB (2400 MHz)
- HDD: 199,32 GB
- SSD: 265,29 GB
- S.O.: Microsoft Windows 11 Home

Wybrane systemy zarządzania bazami danych:

- Microsoft SQL Server Management Studio 18 wersja: 15.0.18404.0
- PostgreSQL 6.4
- MySQL 8.0.29
- SQLiteStudio 3.35.4

3. Konstrukcja bazy danych

Pierwszym krokiem było utworzenie znormalizowanego schematu tabeli geochronologicznej. Do tego celu zostały użyte następujące komendy języka SQL:

CREATE DATABASE Geologia – utworzenie bazy danych

CREATE TABLE GeoEon(id_eon INT PRIMARY KEY, nazwa_eon VARCHAR(80)) – utworzenie tabel, np. GeoEon

INSERT INTO GeoEon VALUES (1, 'Fanerozoik') – wprowadzenie rekordów do tabel

Tabele w schemacie znormalizowanym wyglądają następująco:

GeoEon

	id_eon [PK] integer	nazwa_eon character varying (80)
1	1	Fanerozoik

GeoEra

	id_era [PK] integer	id_eon integer	nazwa_era character varying (80)
1	1	1	Kenozoik
2	2	1	Mezozoik
3	3	1	Paleozoik

GeoOkres

	id_okres [PK] integer	id_era integer	nazwa_okres character varying (80)
1	1	1	Czwartorzęd
2	2	1	Neogen
3	3	1	Paleogen
4	4	2	Kreda
5	5	2	Jura
6	6	2	Trias
7	7	3	Perm
8	8	3	Karbon
9	9	3	Dewon
10	10	3	Sylur
11	11	3	Ordowik
12	12	3	Kambr

GeoEpoka

	id_epoka [PK] integer	id_okres integer	nazwa_epoka character varying (80)
1	1	1	Holocen
2	2	1	Plejstocen
3	3	2	Pliocen
4	4	2	Miocen
5	5	3	Oligocen
6	6	3	Eocen
7	7	3	Paleocen
8	8	4	Kreda Górna
9	9	4	Kreda Dolna
10	10	5	Jura Górna

...

GeoPietro

	id_pietro [PK] integer	id_epoka integer	nazwa_pietro character varying (80)
1	1	1	Megalaj
2	2	1	Northgrip
3	3	1	Grenland
4	4	2	Plejstocen górny
5	5	2	Jon
6	6	2	Chiban
7	7	2	Kalabr
8	8	2	Gelas
9	9	3	Piacent
10	10	3	Zankl

...

Tabela zdenormalizowana GeoTabela zawiera wszystkie dane z powyższych tabel. Została utworzona następującym zapytaniem z wykorzystaniem złączeń naturalnych:

```
CREATE TABLE GeoTabela AS
(SELECT * FROM GeoPietro
 NATURAL JOIN GeoEpoka
 NATURAL JOIN GeoOkres
 NATURAL JOIN GeoEra
 NATURAL JOIN GeoEon);
```

- tabela utworzona z tabel znormalizowanych a następnie wypełniona wszystkimi danymi

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

- dodanie klucza głównego do GeoTabela

GeoTabela

	id_eon integer	id_era integer	id_okres integer	id_epoka integer	id_pietro [PK] integer	nazwa_pietro character varying (80)	nazwa_epoka character varying (80)	nazwa_okres character varying (80)	nazwa_era character varying (80)	nazwa_eon character varying (80)
1	1	1	1	1	1	Megalaj	Holocen	Czwartorzęd	Kenozoik	Fanerozoik
2	1	1	1	1	2	Northgrip	Holocen	Czwartorzęd	Kenozoik	Fanerozoik
3	1	1	1	1	3	Grenland	Holocen	Czwartorzęd	Kenozoik	Fanerozoik
4	1	1	1	2	4	Plejstocen górny	Plejstocen	Czwartorzęd	Kenozoik	Fanerozoik
5	1	1	1	2	5	Jon	Plejstocen	Czwartorzęd	Kenozoik	Fanerozoik
6	1	1	1	2	6	Chiban	Plejstocen	Czwartorzęd	Kenozoik	Fanerozoik
7	1	1	1	2	7	Kalabr	Plejstocen	Czwartorzęd	Kenozoik	Fanerozoik
8	1	1	1	2	8	Gelas	Plejstocen	Czwartorzęd	Kenozoik	Fanerozoik
9	1	1	2	3	9	Piacent	Pliocen	Neogen	Kenozoik	Fanerozoik
10	1	1	2	3	10	Zankl	Pliocen	Neogen	Kenozoik	Fanerozoik

…

W tabelach GeoEpoka, GeoPietro oraz GeoTabela przedstawiono jedynie 10 pierwszych rekordów, ze względu na dużą obszerność danych.

4. Testy wydajności

Testy wykonano na laptopie, którego specyfikacje są podane na pierwszej stronie sprawozdania. Przetestowano najpopularniejsze darmowe środowiska zarządzania bazami danych, tj. SQL Server, PostgresSQL, MySQL oraz SQLite.

Do wykonania testów wydajności utworzone zostały dodatkowe dwie tabele:

1. *Dziesiec*, która została wypełniona kolejnymi liczbami naturalnymi z zakresu <0; 9>
2. *Milion*, wypełnionej kolejnymi liczbami naturalnymi od 0 do 999 999, która została utworzona na podstawie autozłączenia tabeli *Dziesiec*. Do tego celu zostały użyte odpowiednie polecenia języka SQL:

```
INSERT INTO Milion SELECT a1.cyfra +10* a2.cyfra +100*a3.cyfra + 1000*a4.cyfra
+ 10000*a5.cyfra + 10000*a6.cyfra AS liczba , a6.cyfra AS cyfra, a6.bit AS bit
FROM Dziesiec a1, Dziesiec a2, Dziesiec a3, Dziesiec a4, Dziesiec a5, Dziesiec
a6;
```

Testy wydajności złączeń i zagnieżdżeń z tabelą geochronologiczną w wersji zdenormalizowanej i znormalizowanej zostały wykonane na podstawie 4 zapytań. Procedurę tę przeprowadzono w dwóch etapach:

- I etap obejmował zapytania bez nałożonych indeksów na kolumny danych
- II etap obejmował wykonanie zapytań z nałożonymi indeksami na wszystkie kolumny biorące udział w danym zapytaniu.

Zapytanie 1 (1 ZL)

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

Celem zapytania jest złączenie tabeli Milion ze zdenormalizowaną tabelą geochronologiczną GeoTabela, przy czym operator modulo dopasowuje zakresy złączanych kolumn.

Zapytanie 2 (2 ZL)

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

Analogiczne zapytanie do zapytania 1, przy czym w tym przypadku następuje złączenie tabeli Milion ze znormalizowaną postacią tabeli geochronologicznej, jako naturalne złączenie pięciu tabel.

Zapytanie 3 (3 ZG)

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

Celem zapytania jest złączenie tablicy Milion z tabelą geochronologiczną w postaci zdenormalizowanej, przy czym złączenie jest wykonywane poprzez zagnieżdżenie skorelowane.

Zapytanie 4 (4 ZG)

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

W ostatnim testowanym zapytaniu występuje naturalne złączenie tablicy Milion ze znormalizowaną tabelą geochronologiczną, natomiast zapytanie wewnętrzen jest złączeniem tabel poszczególnych jednostek geochronologicznych.

5. Wyniki testów

Wyniki testów zostały przedstawione za pomocą tabeli Excel. Każdy z testów (zapytań) został przeprowadzony dziesięciokrotnie, aby skutecznie uśrednić czas wykonywania. Wartości skrajne zostały pominięte, a wyniki zostały przestawione w milisekundach [ms].

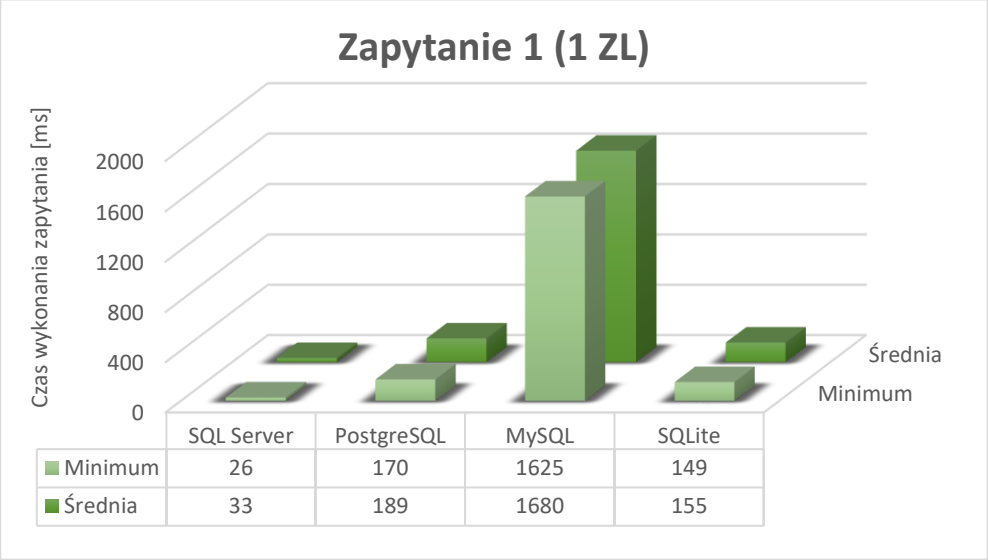
Tabela 1. *Wykonane zapytania bez użycia indeksów.*

	SQL Server				PostgreSQL			
Lp.	Zapytanie 1	Zapytanie 2	Zapytanie 3	Zapytanie 4	Zapytanie 1	Zapytanie 2	Zapytanie 3	Zapytanie 4
1	26	52	30	33	170	348	12285	187
2	34	58	31	35	182	360	12128	191
3	33	56	36	39	251	374	11958	202
4	35	60	33	35	182	359	11971	243
5	32	59	36	35	173	430	12067	170
6	37	55	31	36	178	346	12162	169
7	32	55	38	30	182	368	12134	190
8	36	50	36	37	197	347	12117	168
9	30	58	30	35	182	364	11979	242
10	33	57	35	39	189	403	12115	184
Minimum	26	50	30	30	170	346	11958	168
Średnia	33	56	34	35	189	370	12092	195

	MySQL				SQLite			
Lp.	Zapytanie 1	Zapytanie 2	Zapytanie 3	Zapytanie 4	Zapytanie 1	Zapytanie 2	Zapytanie 3	Zapytanie 4
1	1687	7750	2406	7985	171	582	303	159
2	1781	7579	2343	7610	149	568	295	144
3	1625	7750	2344	7625	152	566	297	146
4	1641	8010	2391	7610	153	586	299	149
5	1656	8390	2328	7578	154	578	295	148
6	1718	7750	2469	7641	151	572	300	145
7	1641	7625	2390	8172	151	600	294	147
8	1750	7609	2359	7532	153	577	294	151
9	1657	7860	2312	7719	162	577	313	151
10	1641	7541	2453	7797	158	568	323	145
Minimum	1625	7541	2312	7532	149	566	294	144
Średnia	1680	7786	2380	7727	155	577	301	149

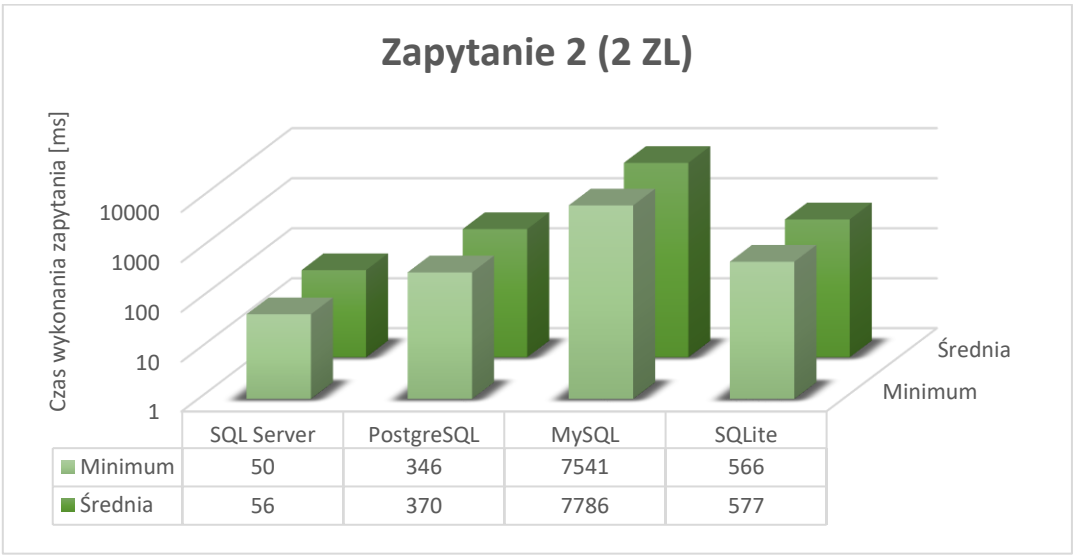
Za pomocą wykresów jesteśmy w stanie dokładniej zanalizować otrzymane wyniki testów.

Wykres 1. *Minimalna i średnia wartość testów dla zapytania 1 (bez indeksów).*



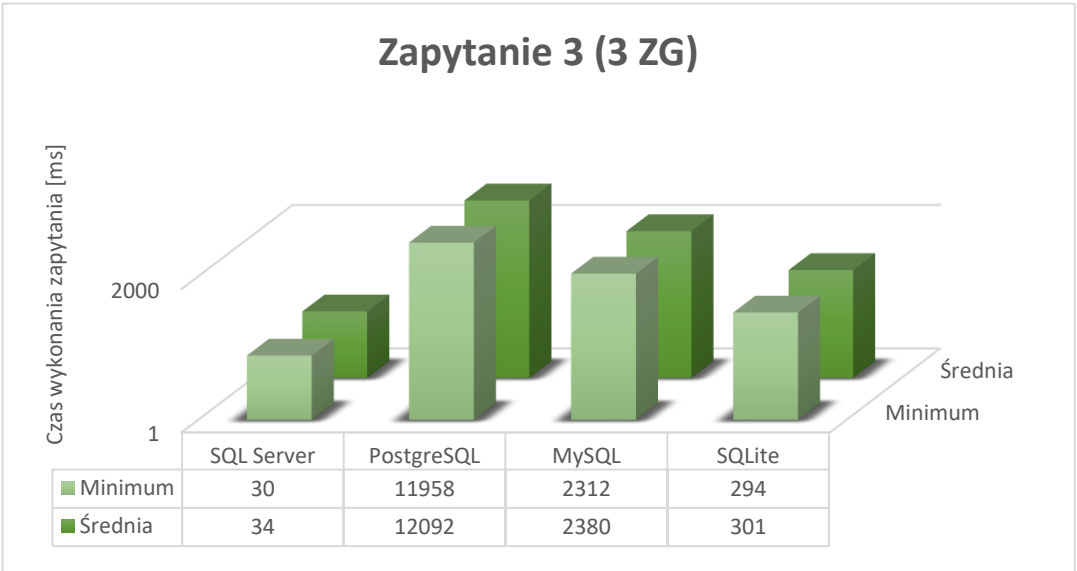
	SQL Server	PostgreSQL	MySQL	SQLite
Minimum	26	170	1625	149
Średnia	33	189	1680	155

Wykres 2. *Minimalna i średnia wartość testów dla zapytania 2 (bez indeksów).*



	SQL Server	PostgreSQL	MySQL	SQLite
Minimum	50	346	7541	566
Średnia	56	370	7786	577

Wykres 3. *Minimalna i średnia wartość testów dla zapytania 3 (bez indeksów).*



	SQL Server	PostgreSQL	MySQL	SQLite
Minimum	30	11958	2312	294
Średnia	34	12092	2380	301

Wykres 4. *Minimalna i średnia wartość testów dla zapytania 4 (bez indeksów).*

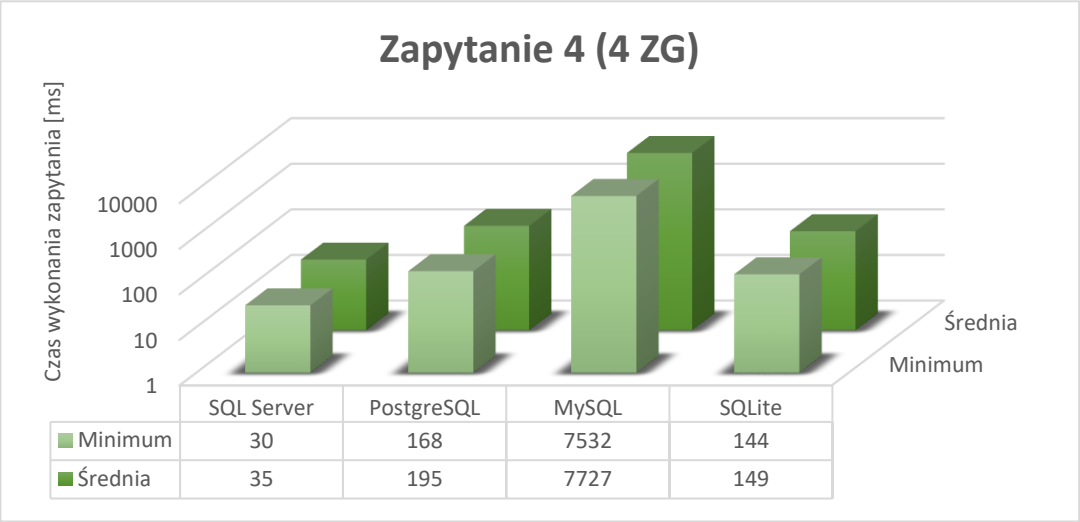
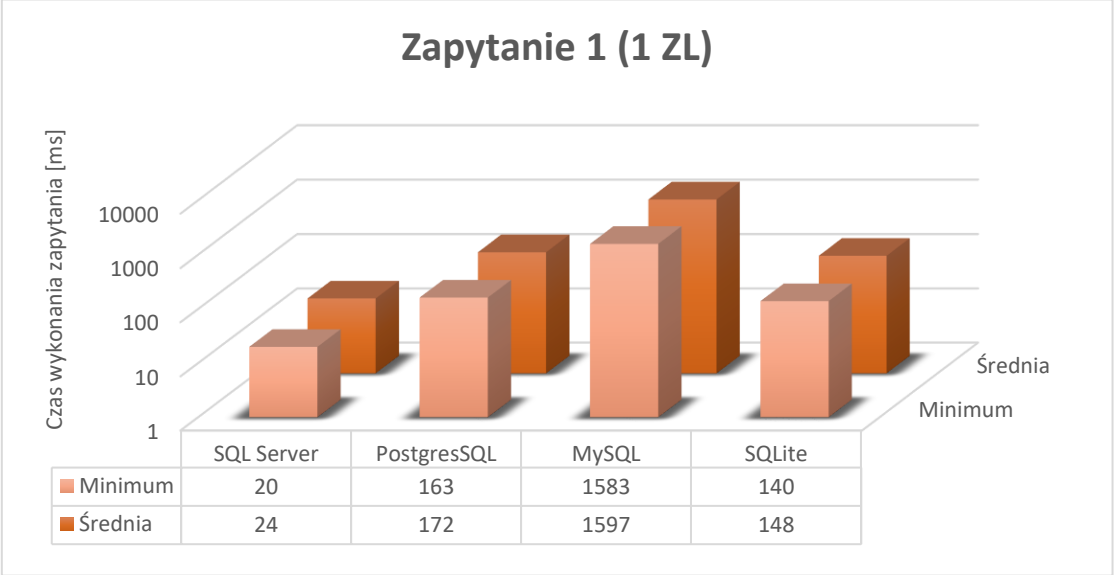


Tabela 2. *Wykonane zapytania z użyciem indeksów.*

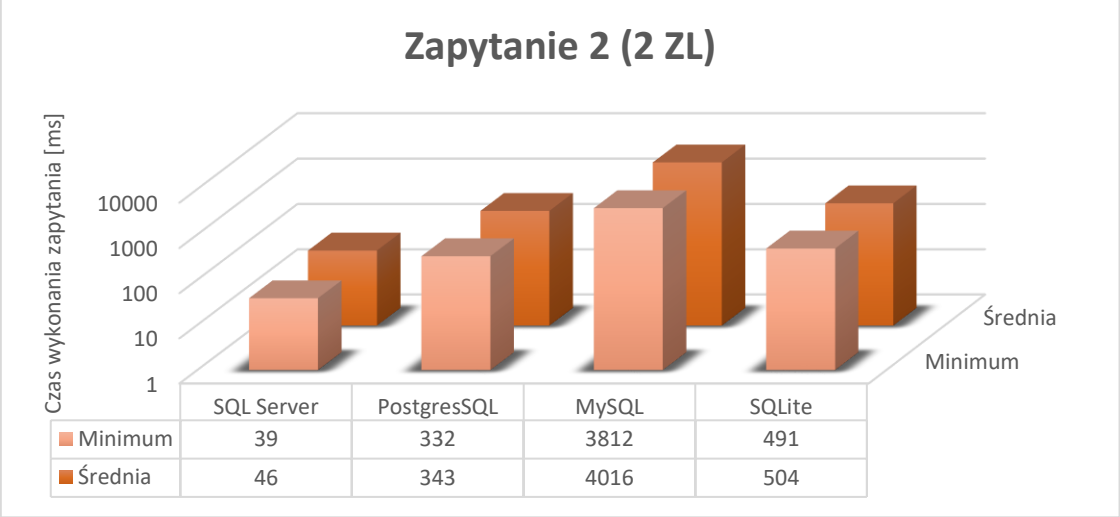
	SQL Server				PostgresSQL			
Lp.	Zapytanie 1	Zapytanie 2	Zapytanie 3	Zapytanie 4	Zapytanie 1	Zapytanie 2	Zapytanie 3	Zapytanie 4
1	21	48	33	34	175	345	11953	158
2	27	48	30	34	176	342	11965	185
3	26	42	31	31	180	349	12076	168
4	20	51	29	36	168	338	12200	165
5	24	43	35	35	170	337	12062	166
6	22	39	34	32	165	345	11976	175
7	22	46	34	32	178	351	12026	171
8	28	47	38	30	171	348	11892	169
9	27	50	32	31	169	341	11995	182
10	25	45	31	33	163	332	11922	159
Minimum	20	39	29	30	163	332	11892	158
Średnia	24	46	33	33	172	343	12007	170

	MySQL				SQLite			
Lp.	Zapytanie 1	Zapytanie 2	Zapytanie 3	Zapytanie 4	Zapytanie 1	Zapytanie 2	Zapytanie 3	Zapytanie 4
1	1596	7601	2278	9411	140	501	275	140
2	1620	7580	2285	9203	145	499	278	142
3	1597	7579	2301	8915	151	498	286	139
4	1596	7550	2296	9210	150	512	275	140
5	1590	7621	2380	9004	148	510	292	138
6	1601	7586	2281	8941	144	508	277	134
7	1600	7690	2278	9240	156	516	281	145
8	1596	7663	2310	9142	152	491	285	137
9	1594	7580	2201	9133	144	499	281	133
10	1583	7610	2299	8994	145	502	273	140
Minimum	1583	7550	2201	8915	140	491	273	133
Średnia	1597	7606	2291	9119	148	504	280	139

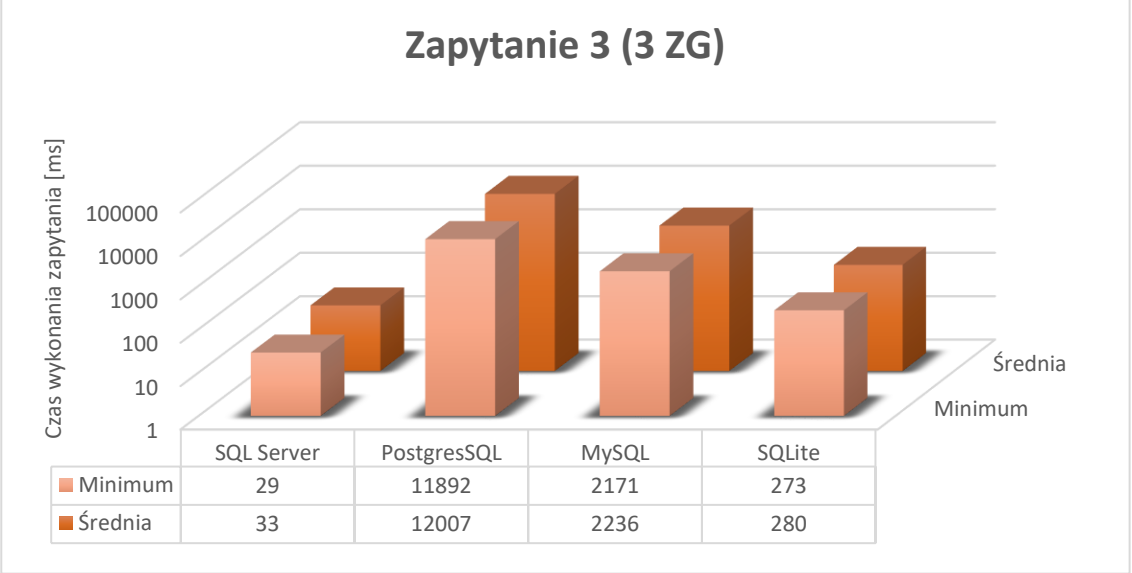
Wykres 5. *Minimalna i średnia wartość testów dla zapytania 1 (z indeksami).*



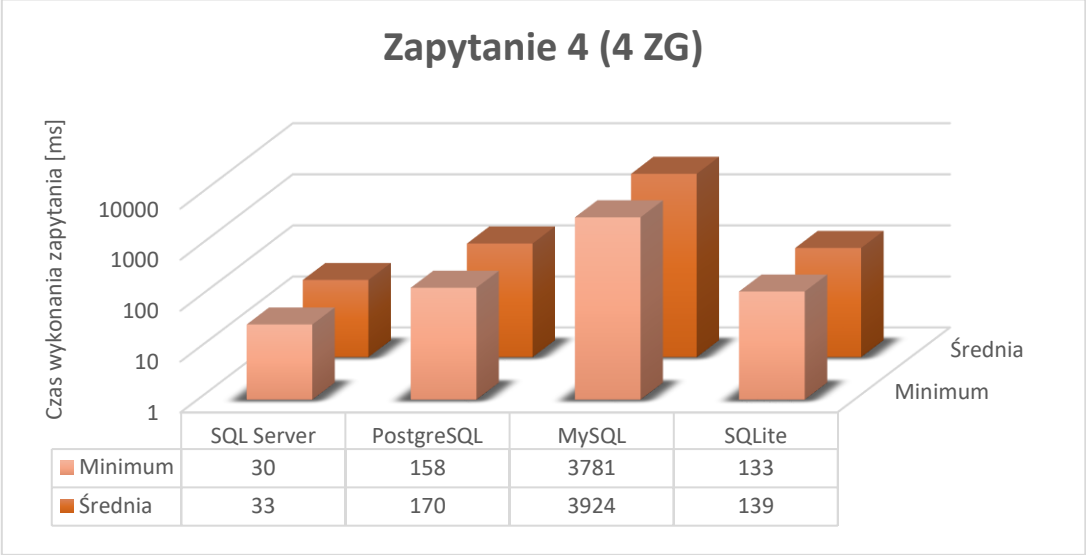
Wykres 6. *Minimalna i średnia wartość testów dla zapytania 2 (z indeksami).*



Wykres 7. *Minimalna i średnia wartość testów dla zapytania 3 (z indeksami).*

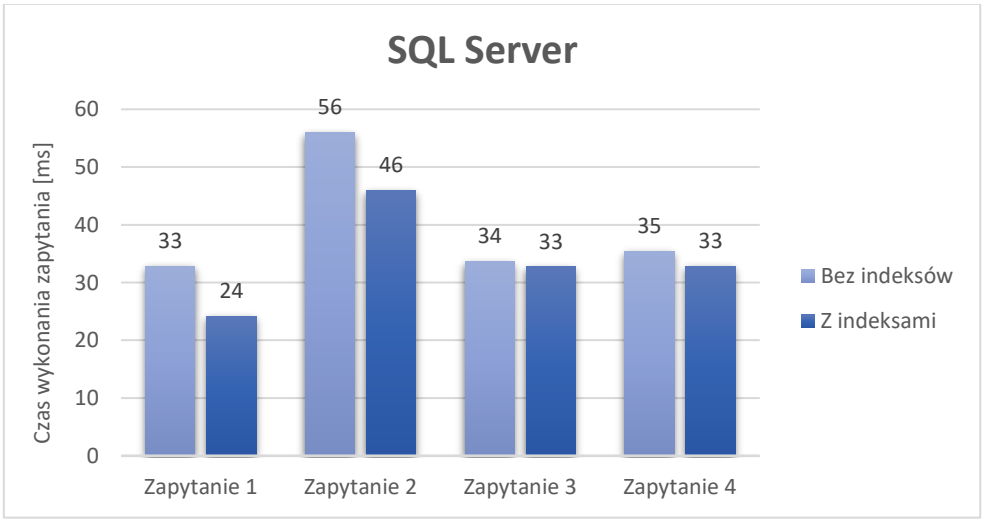


Wykres 8. *Minimalna i średnia wartość testów dla zapytania 4 (z indeksami).*

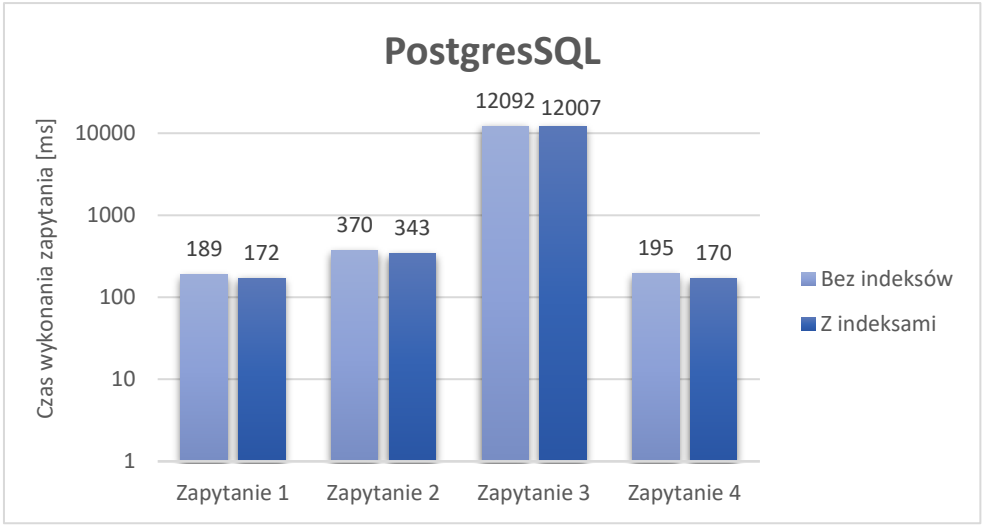


Kolejne wykresy pomogą w analizie czasu wykonania zapytań, dzięki przedstawieniu czasu średniego bez indeksów oraz z ich użyciem.

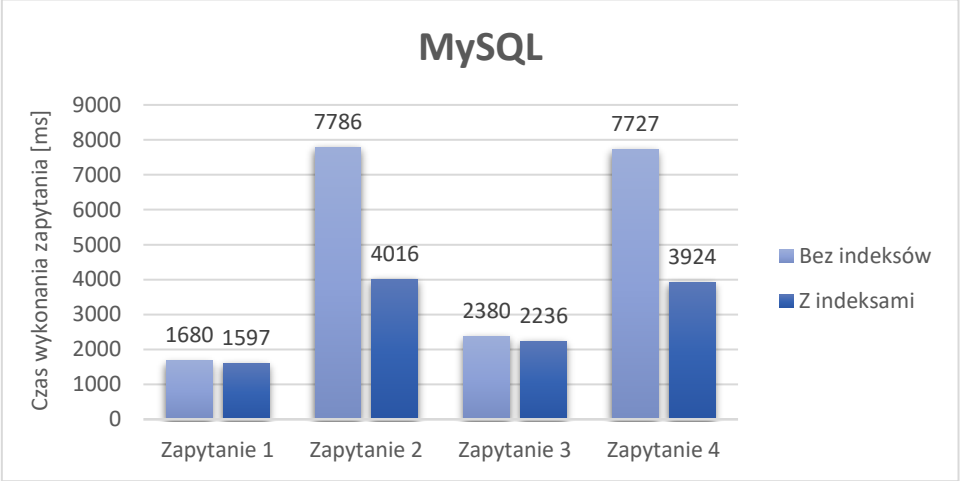
Wykres 9. *Średni czas wykonania zapytań w środowisku SQL Server.*



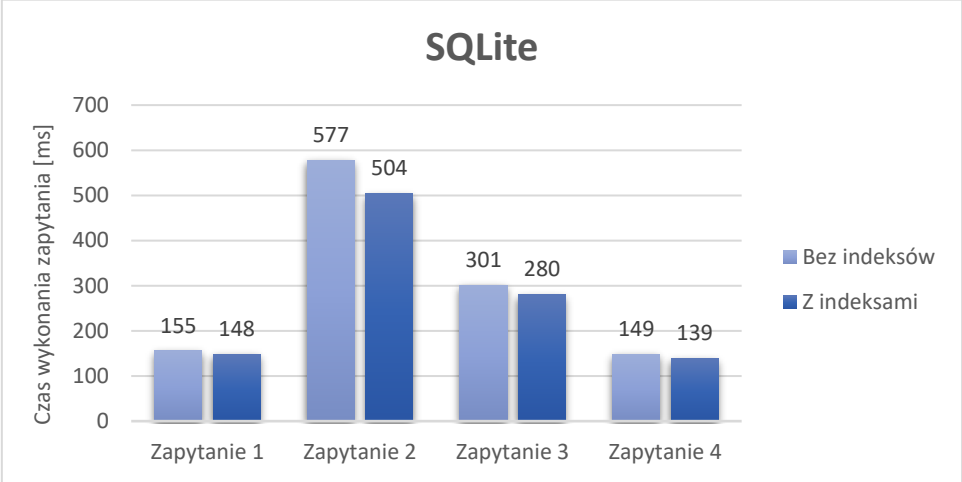
Wykres 10. *Średni czas wykonania zapytań w środowisku PostgresSQL.*



Wykres 11. *Średni czas wykonania zapytań w środowisku MySQL.*



Wykres 12. *Średni czas wykonania zapytań w środowisku SQLite.*



6. Wnioski

Powyższa analiza pozwoliła na szczegółowe zbadanie wydajności złączeń i zagnieżdżeń dla schematów znormalizowanych i zdenormalizowanych. Dzięki wykonaniu testów na czterech systemach zarządzania bazami danych można wyciągnąć wnikliwe wnioski.

Otrzymane wyniki skłaniają do nastepujacych wniosków:
Postać zdenormalizowana okazała się wydajniejsza od postaci znormalizowanej. Wiąże się to z brakiem konieczności wielokrotnego złączenia tabel biorących udział w zapytaniu. Wyjątek jednak stanowi zapytanie trzecie (3 ZG), gdzie w środowisku PostgresSQL czas wykonania jest widocznie dłuższy (rzędu kilkunastu sekund) oraz w zapytaniu czwartym (4 ZG), gdzie dla środowiska PostgresSQL oraz SQLite czas jest znacznie krótszy niż w środowisku MySQL. Z najlepszej strony pokazał się SQLServer, w którym wyniki wszystkich zapytań, z indeksami i

bez nich, okazały się najlepszymi czasami. Może to być spowodowane faktem, iż w środowisku jest wbudowana funkcjonalność multiprocessing, która wykorzystuje wszystkie dostępne rdzenie procesora podczas wykonywania zapytań. Natomiast, w MySQLu wykonanie wszystkich testów trwało najdłużej. Wyniki w tym środowisku nie spadały poniżej 1 sekundy. W każdym ze środowisk zaobserowano przyspieszenie wykonania zapytań po użyciu indeksowania *(Wykres 9. – Wykres 12.)*. Były przypadki, w których indeksy spowodowały prawie 2 krotnie mniejszy czas realizacji zapytania, jak na przykład zapytanie 2 i 4 dla środowiska MySQL *(Wykres 11.)*. Natomiast w znacznej ilości przyspieszenie nie przekraczało 10% czasu wykonania zapytań bez użycia indeksów. W przeprowadzonych testach nie zaobserwowano spadku wydajności między zapytaniami bez użycia indeksów a z ich użyciem. Powyższe rozważania prowadzą do końcowych tez: Złączenia są wydajniejsze czasowo od zagnieżdżeń, które spowalniają wykonanie zapytań oraz Postać zdenormalizowana jest bardziej wydajna od postaci znormalizowanej. Jednakże, złożoność postaci zdenormalizowanej jest ciężka w interpretacji i „obsłudze" przez analityków, dlatego podczas tworzenia bazy danych należy postawić na szereg kompromisów, które składają się na to w jakich przypadkach lepiej będzie użyć danej postaci tabel.

7. Podsumowanie

Postać zdenormalizowana jest znacznie wydajniejsza w zapytaniach niż postać znormalizowana. Trzeba jednak pamiętać, że postać zdenormalizowana może prowadzić do takich problemów jak: redundancja danych, anomalia wprowadzania, anomalia usuwania oraz anomalia modyfikowana.
Najlepszym systemem zarządzania bazami danych okazał się SQL Server, natomiast najgorszym MySQL, którego wydajność jest znacznie mniejsza od wydajności reszty środowisk *(Tabela 2. i Tabela 3.)*.
Po wykonaniu pracy w czterech różnych środowiskach dochodzę do wniosku, iż systemy bazodanowe PostgresSQL, MySQL oraz SQLite są kompatybilne ze sobą. Kod między środowiskami nie różnił się znacząco. Natomiast podczas przygotowania do analizy w SQL Serverze, szczególnie na etapie tworzenie zapytań, kod musiał zostać dostosowany do składni funkcjonującej w tym środowisku.

8. Literatura

- Ł. Jajeśnica, A. Piórkowski; Studia Informatica: „Wydajność złączeń i zagnieżdżeń dla schematów znormalizowanych i zdenormalizowanych"
- dr inż. Michał Lupa, dr hab. inż. Adam Piórkowski, Bazy danych 2022 I
- Tabela stratygraficzna rekomendowana przez Międzynarodową Komisję Stratygraficzną, dostępna pod linkiem: http://geogrfiz.geo.uni.lodz.pl/dyd/Tabela_strat_PIG2007-09.pdf jako źródło danych
- dr inż. Sebastian Ernst, Indeksowanie w bazach danych