Bazy Danych - Indeksowanie

Kacper Krawczyk

07.06.2022

1 Wprowadzenie.

Artykuł jest przygotowywany na podstawie projektu Łukasza Jajeśnica, który przeprowadził badania pod patronatem Adama Piórowskiego z Akademii Górniczo Hutniczej, Katedry Geoinformatyki i Informatyki Stosowanej.

Projekt ten ma na celu sprawdzenie wydajności zapytań, złączeń i zagnieżdżeń, zindeksowanych lub niezindeksowanych dla schematów znormalizowanych i zdenormalizowanych. Został on zrealizowany w kilku rozwiązaniach bazodanowych - MySQL, PostgreSQL, SQL Server i SQLite. Dane, na których opiera się projekt są związane z tabelą geochronologiczną. Zostały one przedstawione w postaci dwóch schematów bazodanowych - schemat znormalizowany płatka śniegu i schemat zdenormalizowany gwiazdy.

Proces analizy został poparty szeregiem wykresów, które pozwoliły na kompleksowe przedstawienie wyników analizy jak i wyciągnięcie wniosków związanych z odpowiedzią na postawione pytania.

Główne pytania postawione podczas analizy:

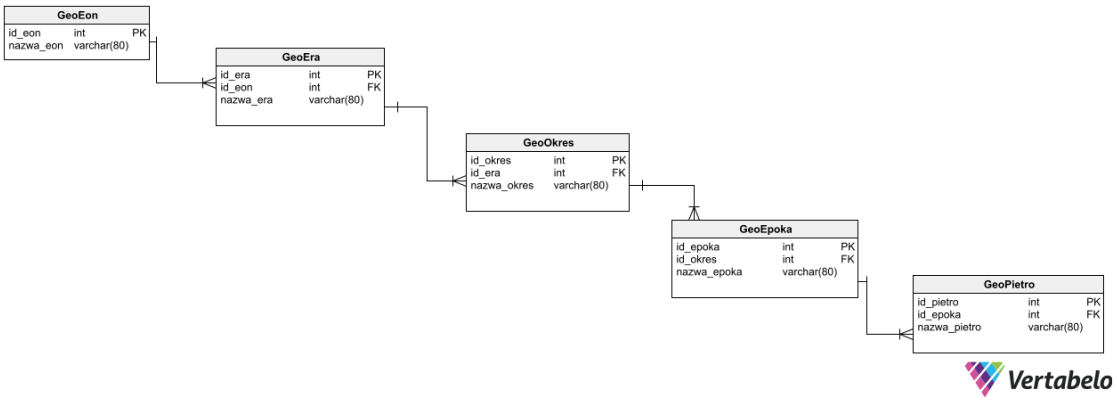
- Która postać jest bardziej wydajna - znormalizowana czy zdenormalizowana?
- Jak wydajne okazuje się użycie indeksów w poszczególnych systemach bazodanowych?
- Czy występują przypadki spowolnienia działania zapytań w przypadku użycia indeksów?
- Który system bazodanowy okażę się najbardziej efektywny w zadanym środowisku danych?

2 Dane.

Głównym źródłem danych była tabela geochronologiczna, zawierająca jednostki geochronologiczne mające wymiar czasowy (eon, era, okres, epoka i wiek), oraz odpowiadające im jednostki stratygraficzne. Wykorzystana tabela geochronologiczna:

Rysunek 1: Tabela Geochronologiczna.

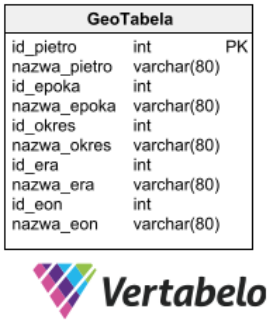
Na początku tabela geochronologiczna została przedstawiona w postaci znormalizowanej - schemat płatka śniegu. Tabela została wypełniona danymi na podstawie tabeli przedstawionej na rysunku 1.



Rysunek 2: Tabela Geochronologiczna - Geotabela - postać znormalizowana, schemat płatka śniegu.

Następnie na podstawie tych tabel, zostało uruchomione złączenie naturalne, które pozwoliło na stworzenie tabeli zdenormalizowanej - schemat gwiazdy, oraz na wypełnienie jej wcześniej przygotowanymi danymi.

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



Rysunek 3: Tabela Geochronologiczna - Geotabela - postać zdenormalizowana, schemat gwiazdy.

Aby móc przeprowadzić analizę wydajności została wprowadzona dodatkowa tabela *Milion*, wypełniona kolejnymi liczbami naturalnym od 0 do 999 999. Została ona utworzona na podstawie odpowiedniego auto-złączenia tabeli *Dziesiec* wypełnionej liczbami od 0 do 9:

```
CREATE TABLE Dziesiec(cyfra int, bit int);

INSERT INTO Dziesiec VALUES
(0, 0000000),
(1, 0000001),
(2, 0000010),
(3, 0000011),
(4, 0000100),
(5, 0000101),
(6, 0000110),
(7, 0000111),
(8, 0001000),
(9, 0001001);


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 + 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;
```

Rysunek 4: Dane - Tabela Milion i Tabela Dziesiec

Warto również zaznaczyć, że w aktualnym stadium rozwoju środowisk bazodanowych, jeden kod SQL z małymi zmianami był uruchomiony bez żadnych komplikacji w trzech środowiskach bazodanowych - PostgreSQL, MySQL, SQLite. Świadczy to o wysokiej kompatybilności systemów, co jest pożądaną cechą. Inaczej sprawa ma się w przypadku środowiska SQL Server, dla którego kod musiał zostać znacząco zmieniony.

3 Konfiguracja sprzętowa

Parametry Komputera:

- CPU: Intel Core i7-9750H (6 rdzeni, 12 wątków, 2.60-4.50 GHz, 12 MB cache)
- RAM: 16 GB (DDR4, 2666MHz)
- GPU: NVIDIA GeForce GTX 1660 Ti
- Dysk: SSD M.2 PCIe 512GB
- Pamięć karty graficznej: 6 GB GDDR6

Środowiska bazodanowe:

- MySQL 8.0,
- PostgreSQL 14.3
- SQLite 3
- SQL Server 2019

Środowisko analityczne:

- Python 3.9
- Anaconda Distribution 4.12.0
- Jupyter 1.0.0
- Pandas 1.3.4
- Numpy 1.20.3
- Seaborn 0.11.2
- Matplotlib 3.4.3

4 Kryteria testów wydajności.

Głównym celem tego projektu było przeprowadzenie testów wydajności, które miały odpowiedzieć na pytania zawarte w wprowadzeniu.

W teście wykonano cztery zapytania sprawdzające wydajność zapytań, złączeń i zagnieżdżeń zindeksowanych lub niezindeksowanych z tabelą geochronologiczną w wersji zdenormalizowanej lub znormalizowanej. W tym celu wprowadzono dwa etapy analizy. Pierwszy etap obejmował analizę wydajności poniżej przedstawionych zapytań w formie niezindeksowanej, natomiast drugi etap w formie zindeksowanej.

Cztery zapytania miały na celu ocenę wpływu normalizacji jak i indeksowania na zapytania złożone - złączenia i zagnieżdżenia:

- **Zapytanie 1 (ZL)**
  Celem tego zapytania jest złączenie syntetycznej tablic miliona wyników tabelą geochronologiczną w postaci zdenormalizowanej, przy czym do warunku złączenia dodano operację modulo, dopasowującą zakresy wartości złączanych kolumn:

```
SELECT COUNT(*) FROM Milion INNER JOIN GeoTabela ON
(mod(Milion.liczba,74)=(GeoTabela.id_pietro));
```

- **Zapytanie 2 (ZL)**
  Celem tego zapytania jest złączenie syntetycznej tablicy miliona wyników z tabelą geochronologiczną w postaci znormalizowanej, reprezentowaną przez złączenia pięciu tabel:

```
SELECT COUNT(*) FROM Milion INNER JOIN GeoPietro ON
(mod(Milion.liczba,74)=GeoPietcro.id_pietro)
NATURAL JOIN GeoEpoka NATURAL JOIN GeoOkres
NATURAL JOIN GeoEra NATURAL JOIN GeoEon;
```

- **Zapytanie 3 (ZG)**
  Celem tego zapytania jest złączenie syntetycznej tablicy miliona wyników z tabelą geochronologiczną w postaci zdenormalizowanej, przy czym złączenie jest wykonywane poprzez zagnieżdżenie skorelowane:

```
SELECT COUNT(*) FROM Milion WHERE mod(Milion.liczba,74)=
(SELECT id_pietro FROM GeoTabela WHERE mod(Milion.liczba,74)=(id_pietro));
```

- **Zapytanie 4 (ZG)**
  Celem tego zapytania 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 tabel poszczególnych jednostek geochronologicznych:

```
SELECT COUNT(*) FROM Milion WHERE mod(Milion.liczba, 74) IN
(
 SELECT GeoPietro.id_pietro FROM GeoPietro
 NATURAL JOIN GeoEpoka NATURAL JOIN
 GeoOkres NATURAL JOIN GeoEra NATURAL JOIN GeoEon
);
```

5 Wyniki testów.

Każdy pomiar został wykonany dziesięć razy, w celu wyeliminowania możliwości popełnienia błędu pomiarowego, oraz aby dopasować się do wbudowanych systemów *database tuning*. Poniżej została przedstawiona tabela zawierająca zebrane wyniki dla poszczególnych systemów bazodanowych:

PostgreSQL - No Index				
ID	1ZL: [ms]	2ZL: [ms]	3ZG: [ms]	4ZG: [ms]
1	161	229	9977	144
2	155	204	10010	145
3	156	206	9952	142
4	158	232	9873	149
5	136	213	9939	164
6	187	233	9891	144
7	178	228	9918	141
8	149	219	9898	177
9	180	253	9887	130
10	137	232	9932	132
Min	136	204	9873	130
Avg	159,7	224,9	9927,7	146,8

PostgreSQL - Index				
ID	1ZL: [ms]	2ZL: [ms]	3ZG: [ms]	4ZG: [ms]
1	125	209	10712	130
2	124	213	10530	133
3	127	201	10549	141
4	133	202	10714	142
5	125	210	10578	131
6	120	220	10694	134
7	132	207	10598	146
8	123	205	10623	156
9	141	203	10498	135
10	126	210	10458	132
Min	120	201	10458	130
Avg	127,6	208	10595,4	138

Rysunek 5: PostgreSQL - Wyniki Pomiarów

MySQL - No Index				
ID	1ZL: [ms]	2ZL: [ms]	3ZG: [ms]	4ZG: [ms]
1	1594	18782	2282	18547
2	1594	18469	2234	19578
3	1609	18453	2213	18765
4	1594	18437	2253	19450
5	1594	18344	2276	18563
6	1625	18547	2245	19231
7	1593	18621	2295	18312
8	1593	18432	2243	18436
9	1594	18435	2259	18874
10	1578	18457	2276	18932
Min	1578	18344	2213	18312
Avg	1596,8	18497,7	2257,6	18868,8

MySQL - Index				
ID	1ZL: [ms]	2ZL: [ms]	3ZG: [ms]	4ZG: [ms]
1	1704	3625	2296	3593
2	1593	3610	2276	3625
3	1578	3594	2344	3593
4	1594	3609	2281	3594
5	1578	3593	2297	3625
6	1578	3593	2313	3641
7	1593	3640	2281	3593
8	1562	3625	2290	3625
9	1530	3610	2266	3640
10	1532	3641	2250	3640
Min	1530	3593	2250	3593
Avg	1584,2	3614	2289,4	3616,9

Rysunek 6: MySQL - Wyniki Pomiarów

SQL Server - No Index				
ID	1ZL: [ms]	2ZL: [ms]	3ZG: [ms]	4ZG: [ms]
1	25	45	22	25
2	24	40	23	22
3	23	44	25	25
4	27	51	27	31
5	23	37	26	23
6	26	42	27	25
7	29	44	22	26
8	25	36	21	27
9	24	56	24	31
10	25	54	25	32
Min	23	36	21	22
Avg	25,1	44,9	24,2	26,7

SQL Server - Index				
ID	1ZL: [ms]	2ZL: [ms]	3ZG: [ms]	4ZG: [ms]
1	17	38	26	26
2	19	36	21	24
3	20	34	22	23
4	17	39	21	21
5	18	43	23	22
6	24	38	22	23
7	21	37	23	24
8	23	35	21	25
9	22	35	24	26
10	21	34	21	21
Min	17	34	21	21
Avg	20,2	36,9	22,4	23,5

Rysunek 7: SQLServer - Wyniki Pomiarów

SQLite - No Index				
ID	1ZL: [ms]	2ZL: [ms]	3ZG: [ms]	4ZG: [ms]
1	171	629	301	157
2	153	619	292	155
3	146	621	296	144
4	145	626	361	145
5	156	630	303	163
6	150	616	301	152
7	148	621	299	155
8	150	629	302	160
9	148	618	295	153
10	151	629	294	151
Min	145	616	292	144
Avg	151,8	623,8	304,4	153,5

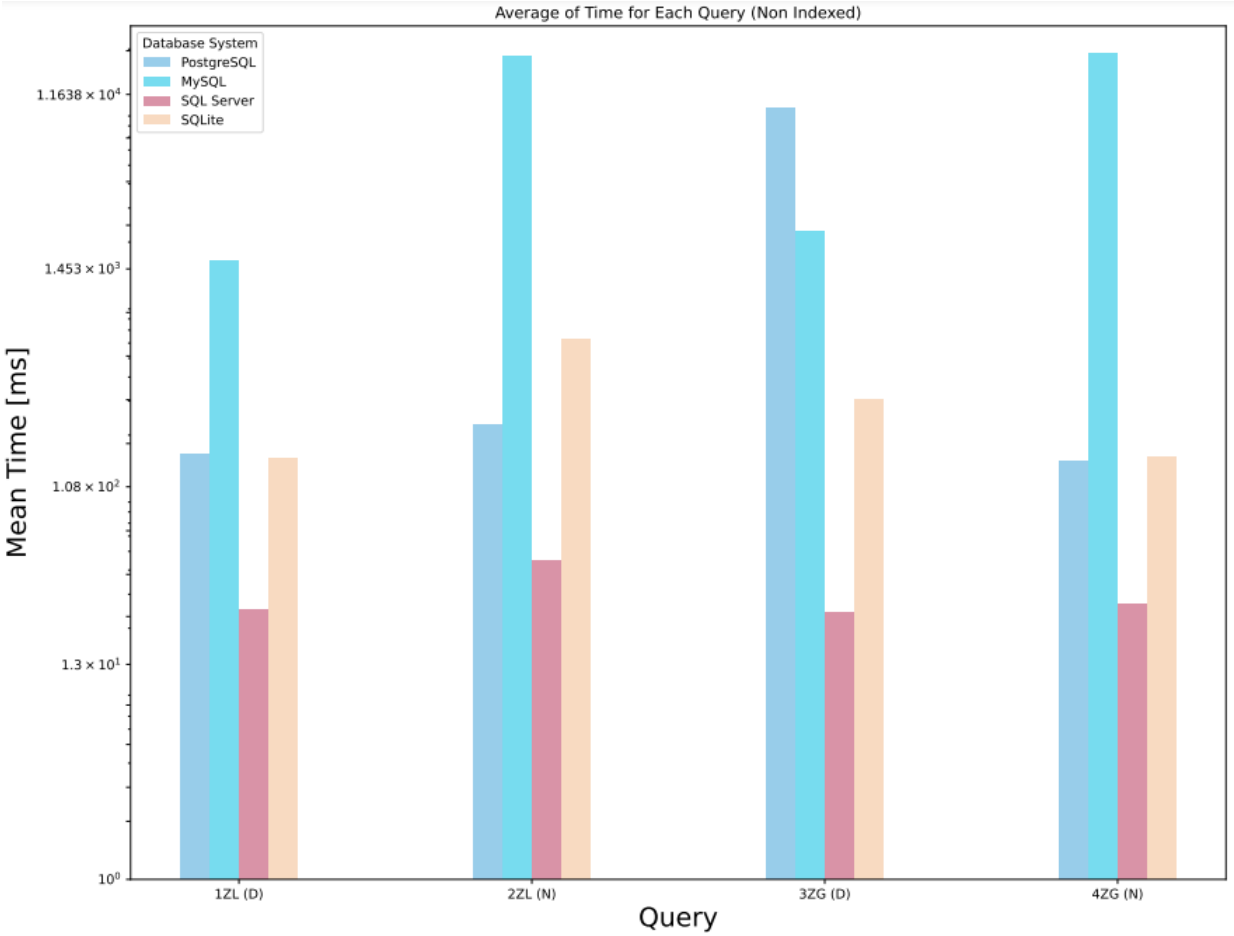
SQLite - Index				
ID	1ZL: [ms]	2ZL: [ms]	3ZG: [ms]	4ZG: [ms]
1	128	535	287	133
2	130	537	270	139
3	130	535	271	134
4	129	541	282	136
5	135	532	269	133
6	130	535	270	133
7	128	541	278	134
8	128	540	269	143
9	127	530	282	139
10	126	537	264	141
Min	126	530	264	133
Avg	129,1	536,3	274,2	136,5

Rysunek 8: SQLite - Wyniki Pomiarów

6 Analiza wyników testu.

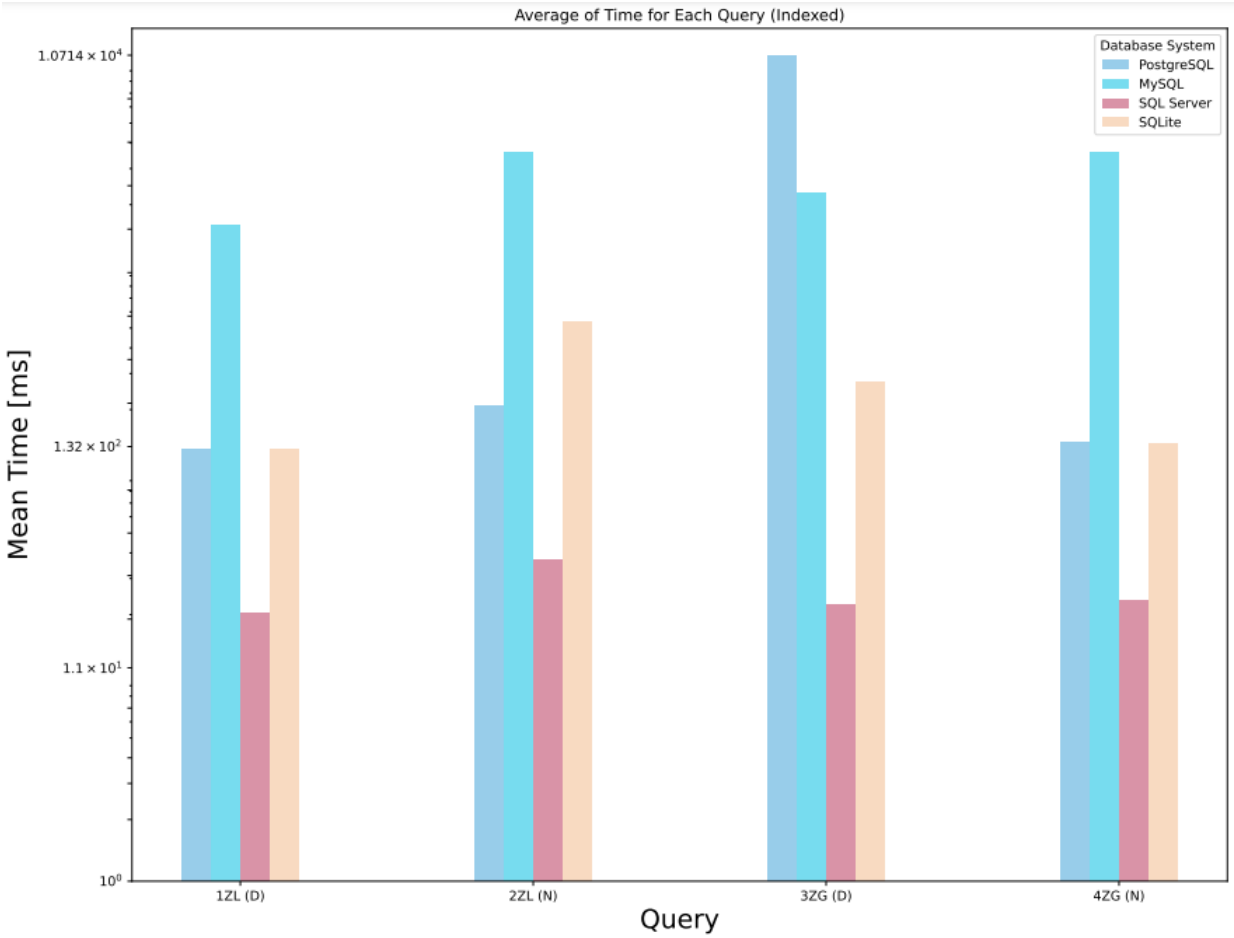
Na podstawie wyników testu została dokonana kompleksowa analiza w środowisku Jupyter Notebook, wykorzystując język programowania Python i bilbioteki Pandas, Numpy, Seaborn i Matplotlib.

6.1 Bar Plot - Non Indexed



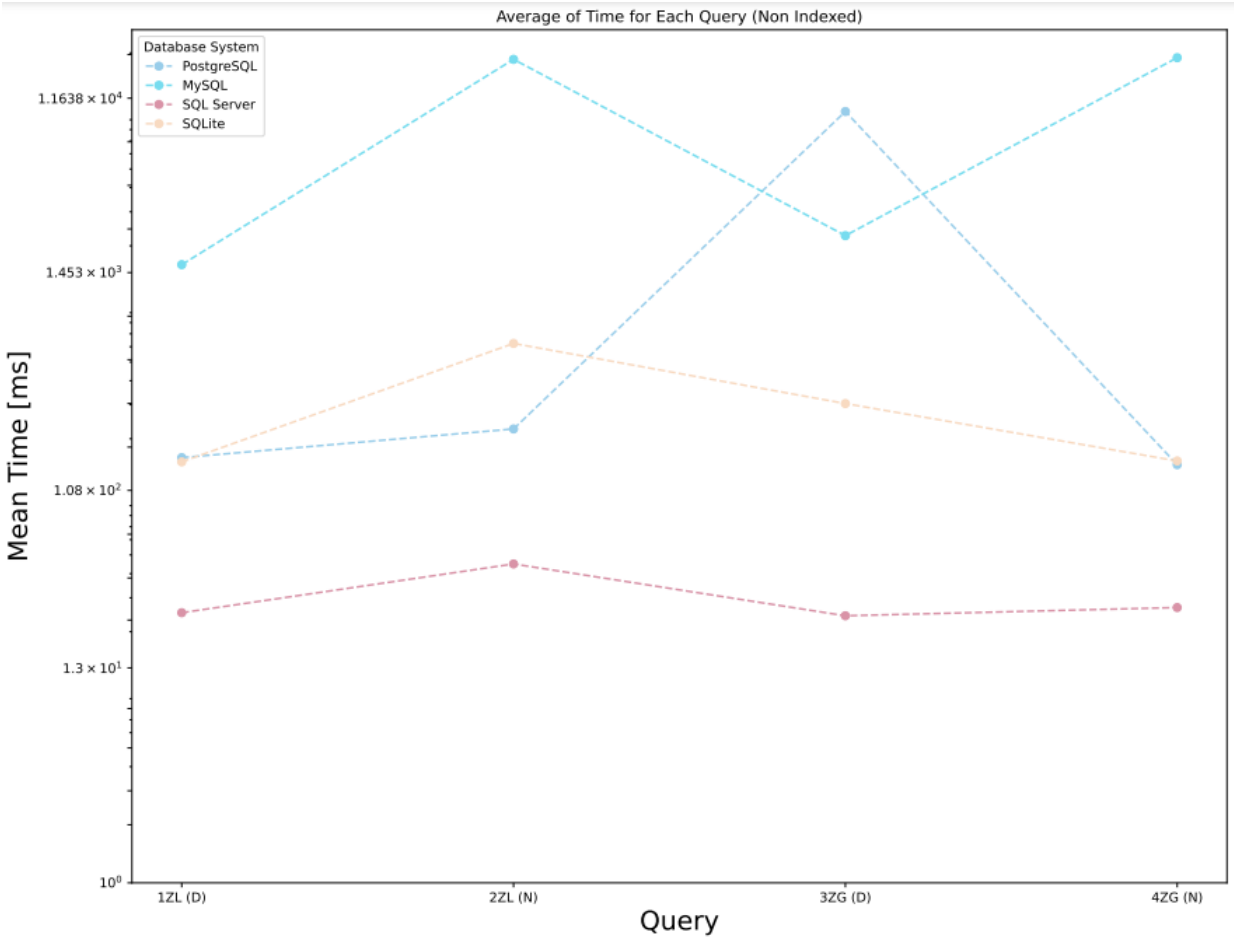
Rysunek 9: Analiza Danych - Bar Plot - Wersja niezindeksowana

6.2 Bar Plot - Indexed



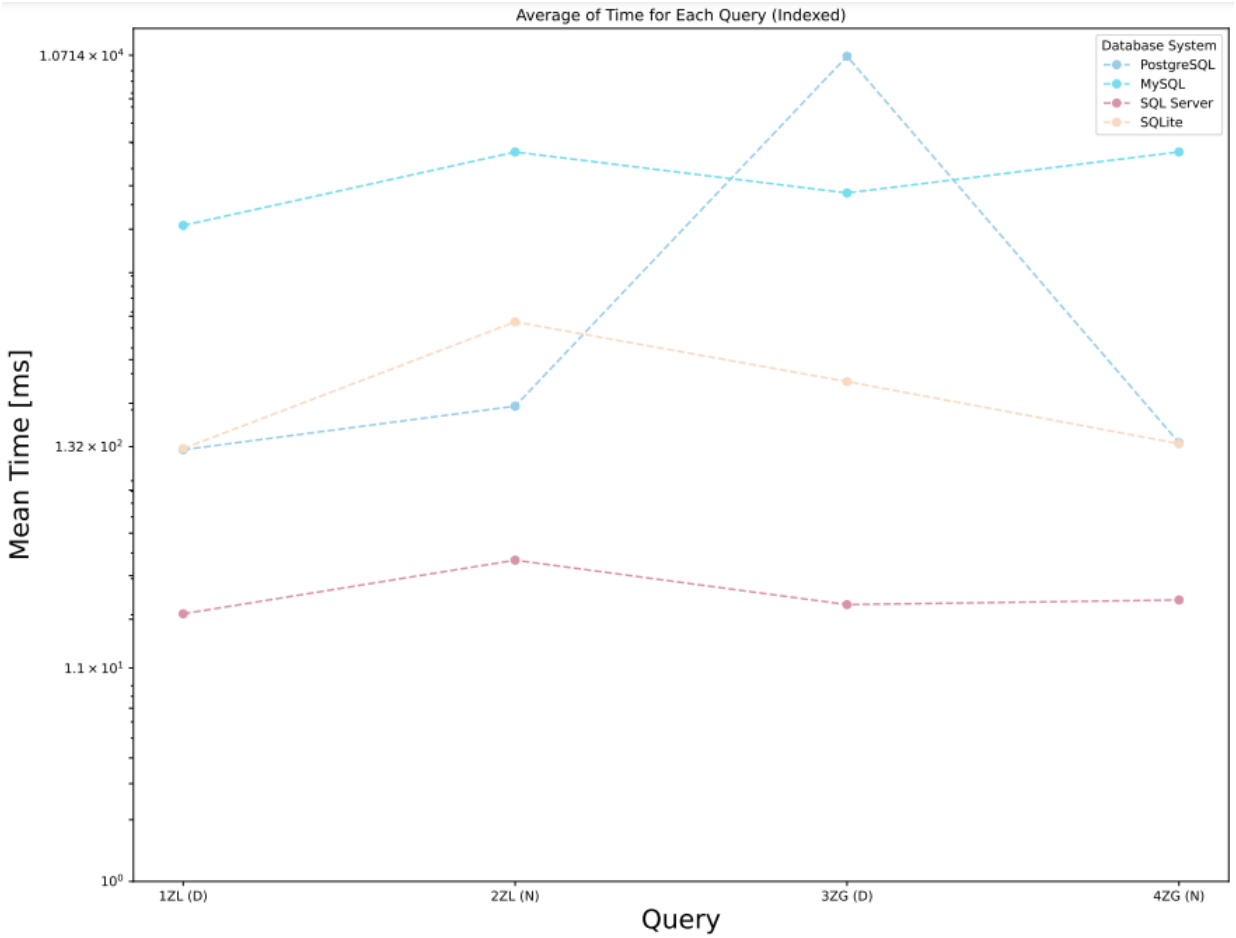
Rysunek 10: Analiza Danych - Bar Plot - Wersja zindeksowana

6.3 Scatter Plot - Non Indexed



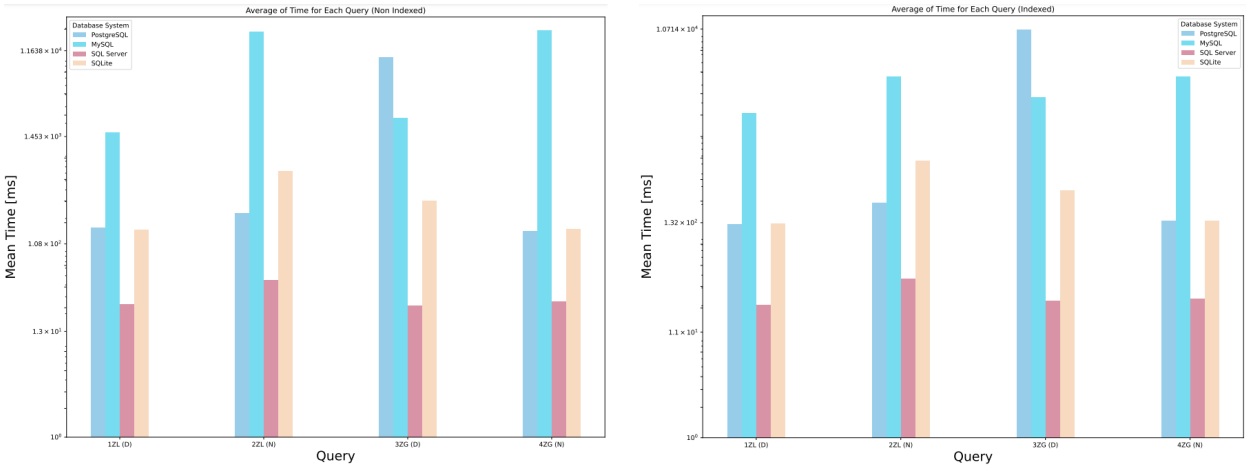
Rysunek 11: Analiza Danych - Scatter Plot - Wersja niezindeksowana

6.4 Scatter Plot - Indexed



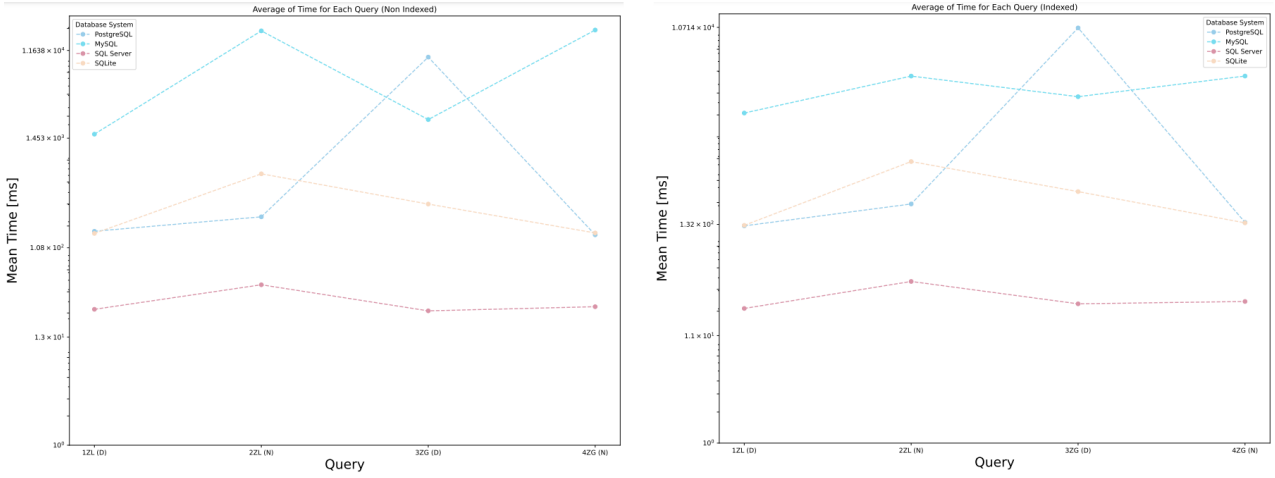
Rysunek 12: Analiza Danych - Scatter Plot - Wersja zindeksowana

6.5 Comparison - Bar Plot



Rysunek 13: Analiza Danych - Bar Plot - Porównanie wersji zindeksowanej i niezindeksowanej

6.6 Comparison - Scatter Plot



Rysunek 14: Analiza Danych - Scatter Plot - Porównanie wersji zindeksowanej i niezindeksowanej

7 Wnioski.

Kompleksowa analiza pozwoliła na wyciągnięcie wniosków związanych z czasem działania zapytań w różnych konfiguracjach. Wnioski będą zawierały odpowiedzi na pytania zawarte we wprowadzeniu jak i kolejne spostrzeżenia.

Na samym wstępie trzeba zaznaczyć, że porównywanie ze sobą kilku środowisk jest zależne od wielu czynników, w tym celu wnioski te nie są uniwersalne.

- Która postać jest bardziej wydajna - znormalizowana czy zdenormalizowana?

  Z reguły postać zdenormalizowana okazała się wydajniejsza od postaci znormalizowanej. Wyjątek występuje w zapytaniu trzecim 3ZG (D), gdzie w środowisku PostgreSQL czas jest znacznie wydłużony, oraz w zapytaniu czwartym 4ZG (N), dla środowisk PostgreSQL i SQLite, gdzie czas jest krótszy.

- Jak wydajne okazuje się użycie indeksów w poszczególnych systemach bazodanowych?

  Jednoznacznie można powiedzieć, że w prawie w każdym przypadku użycie indeksów poprawiło wydajność zapytań zagnieżdżonych jak i złączeń. Jedyny wyjątek stanowi zapytanie zagnieżdżone w postaci zdenormalizowanej 3ZG dla środowiska PostgreSQL, gdzie zauważalny jest spadek wydajności o 6%.

  Największą różnicę można zaobserwować w przypadku środowiska MySQL, gdzie w przypadku zapytań znormalizowanych 2ZL i 4ZG następuję zwiększenie wydajności o około 500%.

- Czy występują przypadki spowolnienia działania zapytań w przypadku użycia indeksów?

  Tak jak zostało to wspomniane w poprzednim podpunkcie. Jedynym wyjątkiem, w którym występuję spadek wydajności, jest zapytanie zagnieżdżone w postaci zdenormalizowanej 3ZG dla środowiska PostgreSQL, gdzie zauważalny jest spadek wydajności o 6%.

- Który system bazodanowy okażę się najbardziej efektywny w zadanym środowisku danych?

  Tak jak zostało to wcześniej wspomniane, porównywanie systemów bazodanowych może zależeć od wielu czynników. W związku z tym, odpowiedź na to pytanie nie będzie uniwersalna, dotyczy ona tylko i wyłącznie przypadku tych danych przedstawionych powyżej.

Najbardziej wydajnym systemem bazodanowym pod każdym względem okazał się SQL Server, którego wydajność jest o kilkaset procent większa w porównaniu do innych systemów. Może to być spowodowane wbudowaną funkcjonalnością *multithreading/multiprocessing*, która do wykonania zapytań wykorzystała wszystkie możliwe rdzenie procesora.

PostgreSQL okazał się również bardzo wydajnym systemem za wyjątkiem zapytania zagnieżdżonego w postaci znormalizowanej 3ZG, którego czas jest o jeden rząd większy niż czas reszty zapytań.

SQLite to kolejny wydajny system, który nie posiada wartości odstających. Najgorzej radzi on sobie ze złączeniem w postaci znormalizowanej.

Najmniej wydajnym systemem okazał się system MySQL, którego wydajność jest o wiele mniejsza niż wydajność reszty systemów. W szczególności w przypadku zapytań znormalizowanych 2ZL i 4ZG.

Dodatkowe wnioski

- Można wysunąć wniosek na temat tego, że postać zdenormalizowana jest bardziej wydajna od postaci znormalizowanej. Jednakże, trzeba pamiętać o innych zaletach postaci znormalizowanej, takich jak uniknięciu szeregu potencjalnych problemów, między innymi anomalie odczytu, anomalie zapisu, nadmiarowość danych i nadmiarowość danych.
- Zostało to wcześniej wspomniane, jednak jest to ważna cecha. Systemy bazodanowe MySQL, PostgreSQL i SQLite są ze sobą bardzo kompatybilne. Jeden kod SQL z małą liczbą zmian został użyty do trzech systemów bazodanowych. Jest to cecha bardzo porządna. W przypadku SQL Server, taka cecha nie miała miejsca. Kod musiał zostać dostosowany do klauzul i funkcjonalności tego wyłącznie systemu.

8 Bibliografia

1. mgr inż. Łukasz JAJEŚNICA, prof. nadzw. dr hab. inż. Adam PIÓRKOWSKI - Wydajność Złączeń i Zagnieżdżeń dla Schematów Znormalizowanych I Zdenormalizowanych
2. Wydawnictwo Naukowe PWN S.A. - Historia Ziemii
3. dr inż. Michał LUPA - Bazy Danych 2022
4. dr. inż Sebastian ERNST - Indeksowanie w bazach danych
5. PostgreSQL Manual
6. MySQL Manual
7. SQLite Manual
8. SQL Server Manual
9. IBM SQL Docs