WYDZIAŁ GEOLOGII, GEOFIZYKI I OCHRONY ŚRODOWISKA – AGH

SPRAWOZDANIE

Analiza wydajności złączeń i zagnieżdżeń dla schematów znormalizowanych i zdenormalizowanych

ALEKSANDRA PEŁKA

KIERUNEK: GEOINFORMATYKA

GRUPA: 2

KRAKÓW, 03.06.2021

CEL: Analiza wydajności zapytań dla schematów znormalizowanych i zdenormalizowanych dla systemów zarządzania bazami danych SQL Server oraz PostgreSQL z wykorzystaniem indeksów oraz bez ich użycia, na przykładzie tabeli geochronologicznej.

KONFIGURACJA SPRZĘTOWA I PROGRAMOWA

Przeprowadzone testy wydajności zostały wykonane na jednym laptopie o następujących parametrach:

- CPU: Intel(R) Core(TM) i5-10210U CPU @ 1.60GHz 2.11 GHz
- Pamięć RAM: 12 GB
- SSD: 512 GB
- S.O.: Windows 10 Home
-

Wersje wybranych systemów zarządzania bazami danych:

- Microsoft SQL Server Management Studio 18 wersja 15.0.2000.5
- PostgreSQL 4.30

UTWORZENIE BAZY DANYCH

W celu wykonania testów wydajnościowych, przy pomocy poniższych poleceń (**Polecenie 1, Polecenie 2**) została utworzona baza danych Geo, a także schemat geotabela:

```
CREATE DATABASE Geo;                                    Polecenie 1
CREATE SCHEMA geotabela;                                Polecenie 2
```

Następnie utworzono tabele: eon, era, okres, epoka oraz piętro, odpowiadające kolejnym jednostkom geochronologicznym:

```
CREATE TABLE geotabela.era(Id_era VARCHAR(3) PRIMARY KEY,
Id_eon VARCHAR(2) FOREIGN KEY REFERENCES geotabela.eon,
Nazwa_era VARCHAR(15));                                 Polecenie 3
```

Analogiczne komendy do **Polecenia 3**, zostały użyte w przypadku pozostałych tabel.

Po stworzeniu wyżej wspomnianych tabel, wypełniono je odpowiednimi wartościami, wykorzystując analogiczne polecenia do **Polecenia 4**, dla każdego rekordu:

```
INSERT INTO geotabela.era VALUES('1er', '1e', 'Kenozoik');    Polecenie 4
```

Poszczególne, znormalizowane tabele wypełnione rekordami zostały zaprezentowane poniżej (**Tabela 1**, …, **Tabela 5**):

Tabela 1.

EON	
id_eon	nazwa_eon
1e	Fanerozoik

Tabela 2.

ERA		
id_era	id_eon	nazwa_era
1er	1e	Kenozoik
2er	1e	Mezozoik
3er	1e	Paleozoik

Tabela 3.

OKRES		
id_okres	id_era	nazwa_okres
1o	1er	Czwartorzęd
2o	1er	Neogen
3o	1er	Paleogen
4o	1er	Kreda
5o	1er	Jura
6o	1er	Trias
7o	2er	Perm
8o	2er	Karbon
9o	2er	Dewon
10o	3er	Sylur
11o	3er	Ordowik
12o	3er	Kambr

Tabela 4.

EPOKA		
id_epoka	id_okres	nazwa_epoka
1ep	1o	Holocen
2ep	1o	Plejstocen
3ep	2o	Pliocen
4ep	2o	Miocen
5ep	3o	Oligocen
6ep	3o	Eocen
7ep	3o	Paleocen
8ep	4o	Późna kreda
9ep	4o	Wczesna kreda
10ep	5o	Jura późna
32ep	12o	Miaoling
33ep	12o	Oddział 2
34ep	12o	Terenew

Tabela 5.

PIETRO		
id_pietro	id_epoka	nazwa_pietro
1	1ep	Megalaj
2	1ep	Northgrip
3	1ep	Grenland
4	2ep	Późny
5	2ep	Chiban
6	2ep	Kalabr
7	2ep	Gelas
8	3ep	Piacent
9	3ep	Zankl
10	4ep	Messyn
100	33ep	Pietro 3
101	34ep	Piętro 2
102	34ep	Fortun

Tabela zdenormalizowana (**Tabela 6**), umożliwiająca łatwiejszy dostęp do wszystkich danych, powstała z połączenia rekordów zawartych w pozostałych tabelach (**Tabela 1**, …, **Tabela 5**). W tym celu użyto poniższego zapytania (**Polecenie 5.1, Polecenie 5.2**), wykorzystującego złączenie wewnętrzne:

```
SELECT Id_pietro, Nazwa_pietro, geotabela.epoka.Id_epoka , Nazwa_epoka,
geotabela.okres.Id_okres, Nazwa_okres, geotabela.era.Id_era, Nazwa_era,
geotabela.eon.Id_eon, Nazwa_eon
INTO GeoTabela
FROM  geotabela.pietro
INNER JOIN geotabela.epoka
INNER JOIN geotabela.okres
INNER JOIN geotabela.era
INNER JOIN geotabela.eon
ON geotabela.era.Id_eon = geotabela.eon.Id_eon
ON geotabela.okres.Id_era = geotabela.era.Id_era
ON geotabela.epoka.Id_okres = geotabela.okres.Id_okres
ON geotabela.pietro.Id_epoka =geotabela.epoka.Id_epoka;
```

Polecenie 5.1

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

Polecenie 5.2

Tabela 6. *Zdenormalizowana tabela geochronologiczna*

GeoTabela									
Id_pietro	Nazwa_pietro	Id_epoka	Nazwa_epoka	Id_okres	Nazwa_okres	Id_era	Nazwa_era	Id_eon	Nazwa_eon
1	Megalaj	1ep	Holocen	1o	Czwartorzęd	1er	Kenozoik	1e	Fanerozoik
2	Northgrip	1ep	Holocen	1o	Czwartorzęd	1er	Kenozoik	1e	Fanerozoik
3	Grenland	1ep	Holocen	1o	Czwartorzęd	1er	Kenozoik	1e	Fanerozoik
4	Późny	2ep	Plejstocen	1o	Czwartorzęd	1er	Kenozoik	1e	Fanerozoik
5	Chiban	2ep	Plejstocen	1o	Czwartorzęd	1er	Kenozoik	1e	Fanerozoik
6	Kalabr	2ep	Plejstocen	1o	Czwartorzęd	1er	Kenozoik	1e	Fanerozoik
7	Gelas	2ep	Plejstocen	1o	Czwartorzęd	1er	Kenozoik	1e	Fanerozoik
8	Piacent	3ep	Pliocen	2o	Neogen	1er	Kenozoik	1e	Fanerozoik
9	Zankl	3ep	Pliocen	2o	Neogen	1er	Kenozoik	1e	Fanerozoik
10	Messyn	4ep	Miocen	2o	Neogen	1er	Kenozoik	1e	Fanerozoik
100	Pietro 3	33ep	Oddział 2	12o	Kambr	3er	Paleozoik	1e	Fanerozoik
101	Piętro 2	34ep	Terenew	12o	Kambr	3er	Paleozoik	1e	Fanerozoik
102	Fortun	34ep	Terenew	12o	Kambr	3er	Paleozoik	1e	Fanerozoik

W sprawozdaniu, w tabelach: **Tabeli 4**, …, **Tabeli 6,** wyświetlono pierwsze 10 oraz ostatnie 3 rekordy, ze względu na obszerność.

TESTY WYDAJNOŚCI

Przed przystąpieniem do wykonania testów wydajności złączeń oraz zapytań zagnieżdżonych, utworzono dwie dodatkowe tabele, przy użyciu szeregu poniższych poleceń (**Polecenie 6.1**, **Polecenie 6.2**):

```
CREATE TABLE Dziesiec(cyfra INT, bit BIT );
INSERT INTO Dziesiec VALUES(0,8);
…
INSERT INTO Dziesiec VALUES(9,8);
```

Polecenie 6.1

```
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;
```

Polecenie 6.2

Tabela Dziesiec została wypełniona liczbami naturalnymi od 0 do 9, natomiast tabela Milion liczbami od 0 do 999 999.

Test wydajności złączeń oraz zapytań zagnieżdżonych został przeprowadzony na podstawie czterech zapytań (**Zapytanie 1**,... **Zapytanie 4**), w dwóch wersjach:

- bez wykorzystania indeksów
- z wykorzystaniem indeksów nałożonych na wszystkie kolumny biorące udział w zapytaniu

ZAPYTANIE 1:
Złączenie tabeli Milion ze zdenormalizowaną tabelą geochronologiczną GeoTabela z wykorzystaniem operacji modulo, dopasowującej zakresy wartości złączanych kolumn:

```
SELECT COUNT(*) FROM Milion INNER JOIN GeoTabela ON
( (Milion.liczba%68)=(GeoTabela.Id_pietro));
```

Zapytanie 1

ZAPYTANIE 2:
Złączenie tabeli Milion ze znormalizowaną tabelą geochronologiczną GeoTabela z wykorzystaniem operacji modulo:

```
SELECT COUNT(*) FROM Milion
INNER JOIN geotabela.pietro
INNER JOIN geotabela.epoka
INNER JOIN geotabela.okres
INNER JOIN geotabela.era
INNER JOIN geotabela.eon
ON geotabela.era.Id_eon = geotabela.eon.Id_eon
ON geotabela.okres.Id_era = geotabela.era.Id_era
ON geotabela.epoka.Id_okres = geotabela.okres.Id_okres
ON geotabela.pietro.Id_epoka =geotabela.epoka.Id_epoka
ON ((Milion.liczba%68)=geotabela.pietro.Id_pietro);
```

Zapytanie 2

ZAPYTANIE 3:
Złączenie tabeli Milion ze zdenormalizowaną tabelą geochronologiczną GeoTabela poprzez zagnieżdżenie skorelowane:

```
SELECT COUNT(*) FROM Milion WHERE Milion.liczba%68=
(SELECT Id_pietro FROM GeoTabela WHERE Milion.liczba%68 = (Id_pietro));
```

Zapytanie 3

ZAPYTANIE 4:
Złączenie tabeli Milion ze znormalizowaną tabelą geochronologiczną GeoTabela poprzez zagnieżdżenie skorelowane, gdzie zapytanie wewnętrzne jest złączeniem tabel poszczególnych jednostek geochronologicznych:

```
SELECT COUNT(*) FROM Milion WHERE (Milion.liczba%68) IN (SELECT
geotabela.pietro.Id_pietro
FROM geotabela.pietro
INNER JOIN geotabela.epoka
INNER JOIN geotabela.okres
INNER JOIN geotabela.era
INNER JOIN geotabela.eon
ON geotabela.era.Id_eon = geotabela.eon.Id_eon
ON geotabela.okres.Id_era = geotabela.era.Id_era
ON geotabela.epoka.Id_okres = geotabela.okres.Id_okres
ON geotabela.pietro.Id_epoka =geotabela.epoka.Id_epoka)
```

Zapytanie 4

Każdy z testów został przeprowadzony dziesięciokrotnie (wartości skrajne pominięto). Otrzymane wyniki zostały zaprezentowane w poniższych tabelach (**Tabela 7**, **Tabela 8**):

Tabela 7. *Zapytania wykonane w wersji bez indeksów*

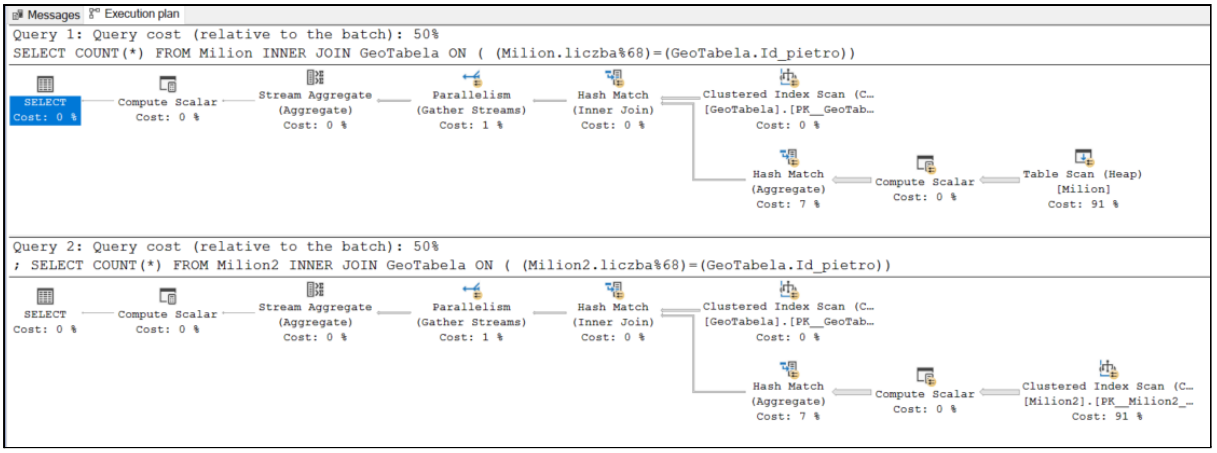
	BEZ INDEKSÓW - SQL Server				BEZ INDEKSÓW - PostgreSQL			
Lp.	Zapytanie 1	Zapytanie 2	Zapytanie 3	Zapytanie 4	Zapytanie 1	Zapytanie 2	Zapytanie 3	Zapytanie 4
1	61	41	56	56	273	493	17129	263
2	40	45	55	50	254	459	15757	315
3	40	45	53	42	254	517	16027	321
4	40	44	58	41	294	572	16462	321
5	66	44	52	38	264	670	15828	313
6	53	43	45	47	390	750	16627	301
7	49	56	42	55	318	525	15678	328
8	38	41	45	51	339	563	15741	261
9	36	45	47	43	297	447	16457	275
10	57	50	44	50	296	516	15870	342
Wartość minimalna:	36	41	42	38	254	447	15678	261
Wartość średnia:	48,0	45,4	49,7	47,3	297,9	551,2	16157,6	304,0

Tabela 8. *Zapytania wykonane w wersji z indeksami*

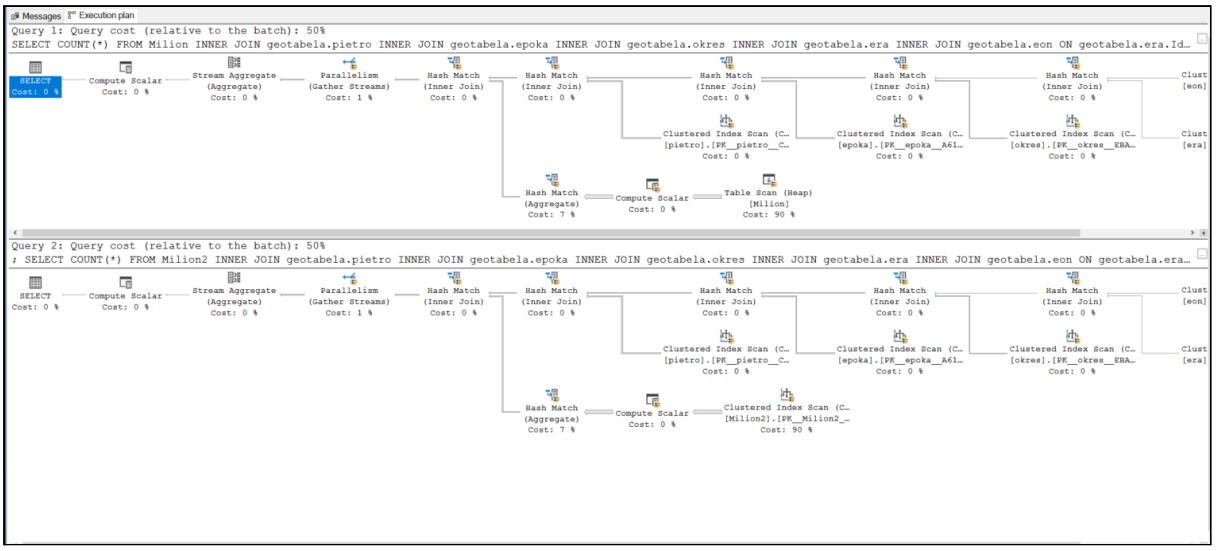
	Z INDEKSAMI - SQL Server				Z INDEKSAMI - PostgreSQL			
Lp.	Zapytanie 1	Zapytanie 2	Zapytanie 3	Zapytanie 4	Zapytanie 1	Zapytanie 2	Zapytanie 3	Zapytanie 4
1	33	40	40	43	196	301	14308	209
2	33	37	30	36	201	307	13504	179
3	37	36	43	39	210	351	13482	184
4	47	47	39	37	212	303	13499	207
5	36	47	32	42	189	290	12986	229
6	39	41	40	33	200	315	12728	225
7	37	44	35	36	180	301	13346	209
8	36	42	35	33	189	304	13551	201
9	33	43	31	43	206	294	13481	222
10	33	44	40	36	192	290	13601	193
Wartość minimalna:	33	36	30	33	180	290	12728	179
Wartość średnia:	36,4	42,1	36,5	37,8	197,5	305,6	13448,6	205,8

Plany wykonania poszczególnych zapytań w wybranych systemach baz danych, w wariantach bez oraz z indeksami, przedstawiające szacunkowy koszt wykonania zapytań, zostały przedstawione na poniższych zrzutach ekranu (**Rys.1**, … **Rys.12**):

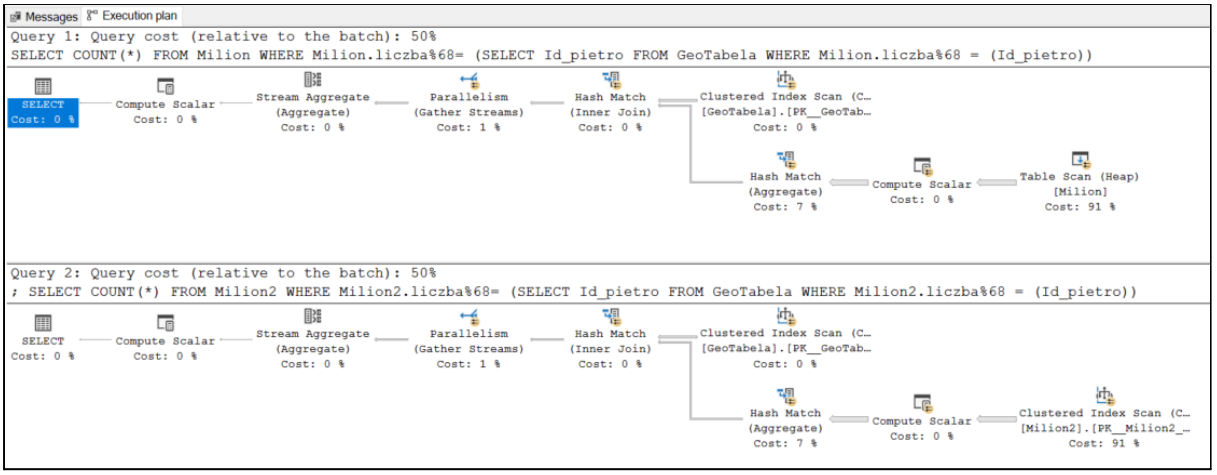
DLA SQL Server:



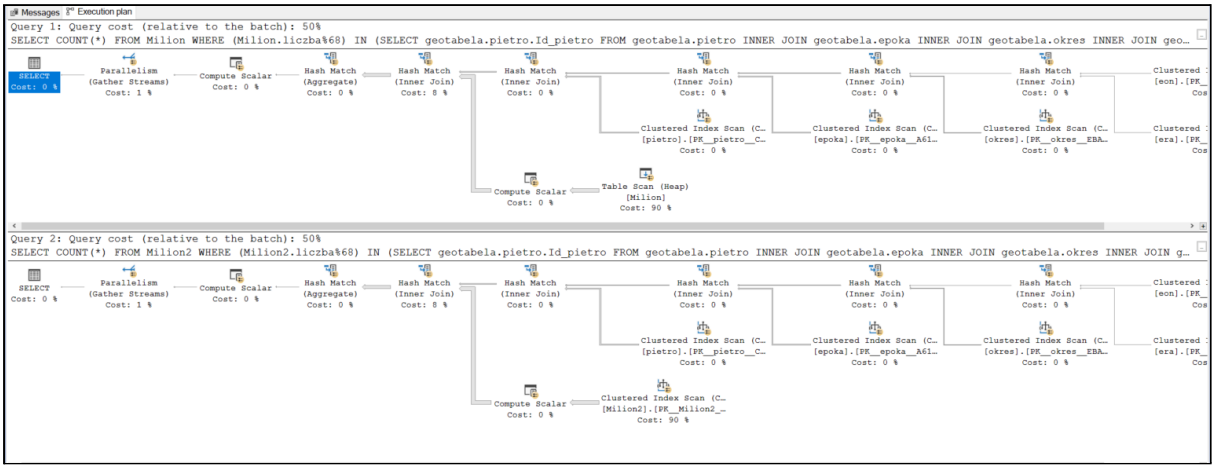
Rys.1. Zapytanie 1 - porównanie wersji z indeksem oraz bez indeksu.



Rys.2. Zapytanie 2 - porównanie wersji z indeksem oraz bez indeksu.



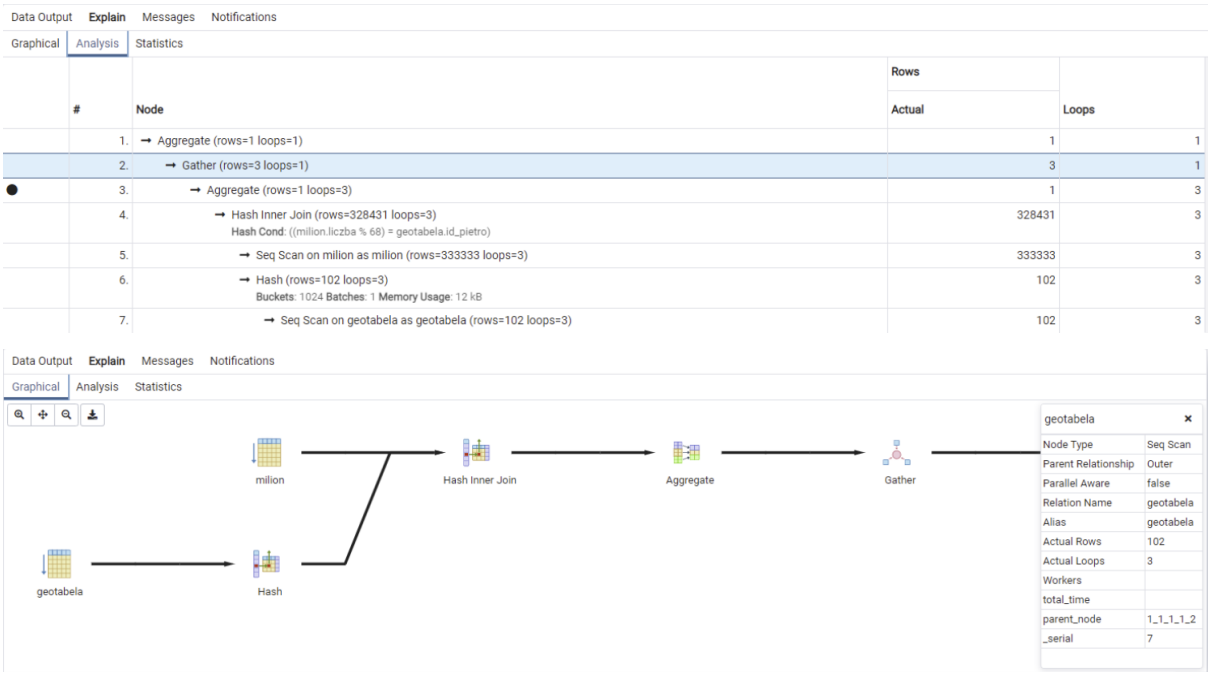
Rys.3. Zapytanie 3 - porównanie wersji z indeksem oraz bez indeksu.



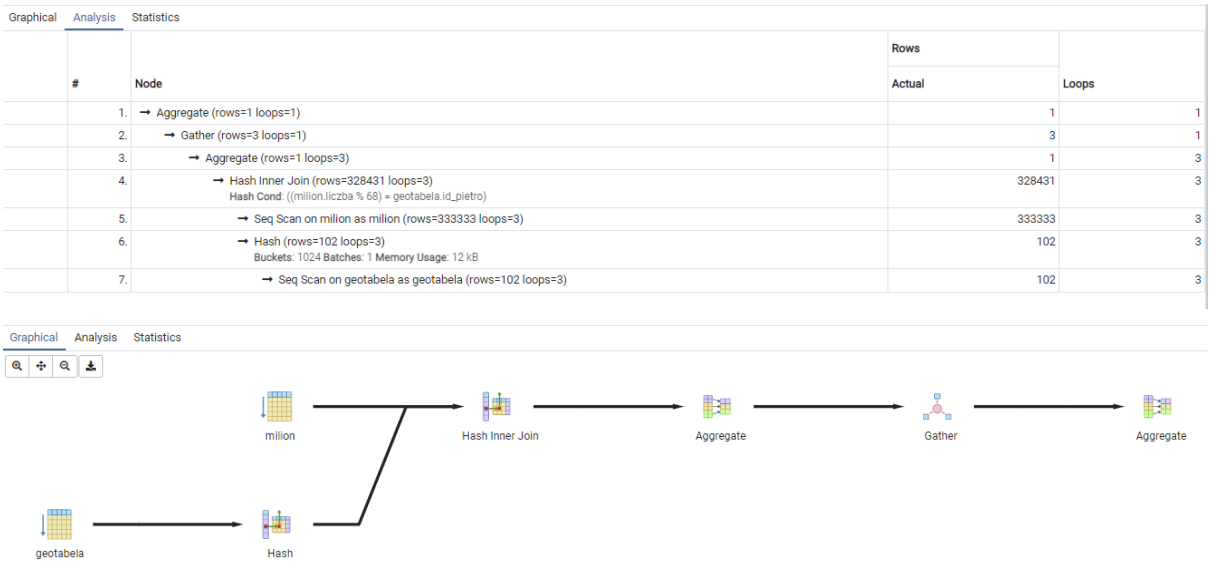
Rys.4. Zapytanie 4 - porównanie wersji z indeksem oraz bez indeksu.

Na powyższych zrzutach ekranu (**Rys.1** ,..., **Rys.4**) w ich górnej części przedstawiono plany realizacji poszczególnych zapytań z wykorzystaniem indeksów, natomiast w części dolnej - bez indeksów.

DLA PostgreSQL:



Rys.5. Zapytanie 1 - wersja z wykorzystaniem indeksu.



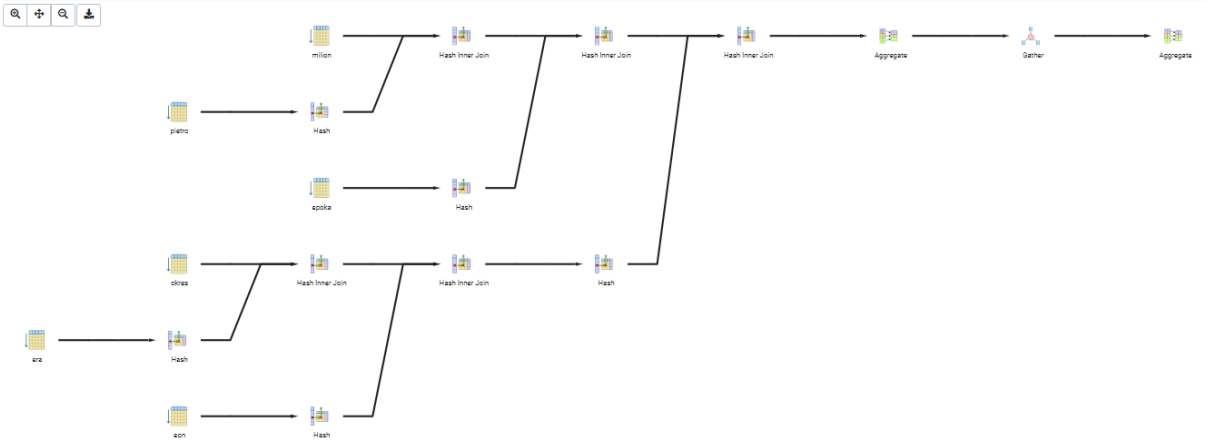
Rys.6. Zapytanie 1 - wersja bez indeksu.

Graphical Analysis Statistics

#	Node	Actual	Loops
1.	→ Aggregate (rows=1 loops=1)	1	1
2.	→ Gather (rows=3 loops=1)	3	1
3.	→ Aggregate (rows=1 loops=3)	1	3
4.	→ Hash Inner Join (rows=328431 loops=3) **Hash Cond**: ((epoka.id_okres)::text = (okres.id_okres)::text)	328431	3
5.	→ Hash Inner Join (rows=328431 loops=3) **Hash Cond**: ((pietro.id_epoka)::text = (epoka.id_epoka)::text)	328431	3
6.	→ Hash Inner Join (rows=328431 loops=3) **Hash Cond**: ((milion.liczba % 68) = pietro.id_pietro)	328431	3
7.	→ Seq Scan on milion as milion (rows=333333 loops=3)	333333	3
8.	→ Hash (rows=102 loops=3) **Buckets**: 1024 **Batches**: 1 **Memory Usage**: 13 kB	102	3
9.	→ Seq Scan on pietro as pietro (rows=102 loops=3)	102	3
10.	→ Hash (rows=34 loops=3) **Buckets**: 1024 **Batches**: 1 **Memory Usage**: 10 kB	34	3
11.	→ Seq Scan on epoka as epoka (rows=34 loops=3)	34	3
12.	→ Hash (rows=12 loops=3) **Buckets**: 1024 **Batches**: 1 **Memory Usage**: 9 kB	12	3
13.	→ Hash Inner Join (rows=12 loops=3) **Hash Cond**: ((era.id_eon)::text = (eon.id_eon)::text)	12	3
14.	→ Hash Inner Join (rows=12 loops=3) **Hash Cond**: ((okres.id_era)::text = (era.id_era)::text)	12	3
15.	→ Seq Scan on okres as okres (rows=12 loops=3)	12	3
16.	→ Hash (rows=3 loops=3) **Buckets**: 1024 **Batches**: 1 **Memory Usage**: 9 kB	3	3
17.	→ Seq Scan on era as era (rows=3 loops=3)	3	3
18.	→ Hash (rows=1 loops=3) **Buckets**: 1024 **Batches**: 1 **Memory Usage**: 9 kB	1	3
19.	→ Seq Scan on eon as eon (rows=1 loops=3)	1	3

Data Output **Explain** Messages Notifications

Graphical Analysis Statistics

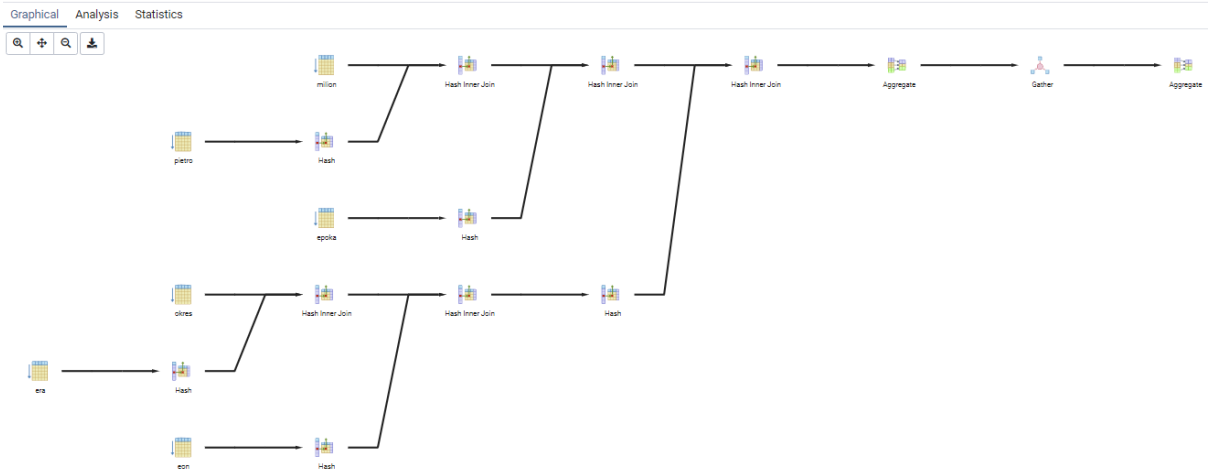


Rys.7. Zapytanie 2 - wersja z wykorzystaniem indeksu.

Graphical | Analysis | Statistics

#	Node	Rows	
		Actual	Loops
1.	→ Aggregate (rows=1 loops=1)	1	1
2.	→ Gather (rows=3 loops=1)	3	1
3.	→ Aggregate (rows=1 loops=3)	1	3
4.	→ Hash Inner Join (rows=328431 loops=3) **Hash Cond**: ((epoka.id_okres)::text = (okres.id_okres)::text)	328431	3
5.	→ Hash Inner Join (rows=328431 loops=3) **Hash Cond**: ((pietro.id_epoka)::text = (epoka.id_epoka)::text)	328431	3
6.	→ Hash Inner Join (rows=328431 loops=3) **Hash Cond**: ((milion.liczba % 68) = pietro.id_pietro)	328431	3
7.	→ Seq Scan on milion as milion (rows=333333 loops=3)	333333	3
8.	→ Hash (rows=102 loops=3) **Buckets**: 1024 **Batches**: 1 **Memory Usage**: 13 kB	102	3
9.	→ Seq Scan on pietro as pietro (rows=102 loops=3)	102	3
10.	→ Hash (rows=34 loops=3) **Buckets**: 1024 **Batches**: 1 **Memory Usage**: 10 kB	34	3
11.	→ Seq Scan on epoka as epoka (rows=34 loops=3)	34	3
12.	→ Hash (rows=12 loops=3) **Buckets**: 1024 **Batches**: 1 **Memory Usage**: 9 kB	12	3
13.	→ Hash Inner Join (rows=12 loops=3) **Hash Cond**: ((era.id_eon)::text = (eon.id_eon)::text)	12	3
14.	→ Hash Inner Join (rows=12 loops=3) **Hash Cond**: ((okres.id_era)::text = (era.id_era)::text)	12	3
15.	→ Seq Scan on okres as okres (rows=12 loops=3)	12	3
16.	→ Hash (rows=3 loops=3) **Buckets**: 1024 **Batches**: 1 **Memory Usage**: 9 kB	3	3
17.	→ Seq Scan on era as era (rows=3 loops=3)	3	3
18.	→ Hash (rows=1 loops=3) **Buckets**: 1024 **Batches**: 1 **Memory Usage**: 9 kB	1	3
19.	→ Seq Scan on eon as eon (rows=1 loops=3)	1	3

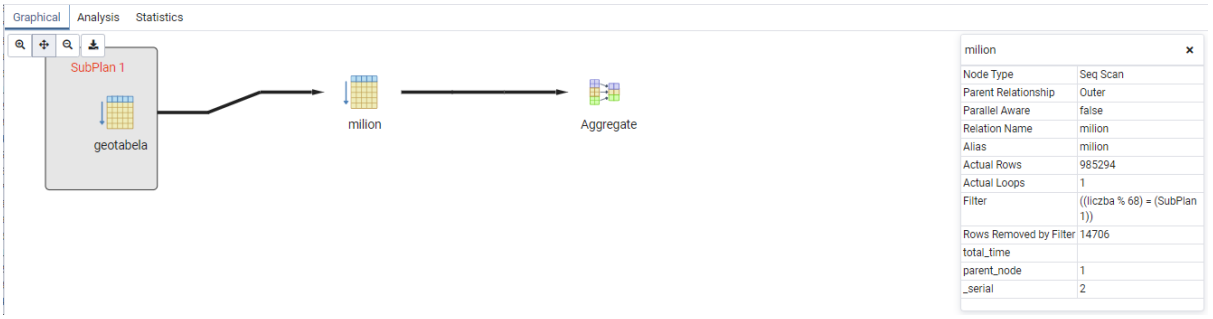
Graphical | Analysis | Statistics



Rys.8. Zapytanie 2 - wersja bez indeksu.

Graphical | Analysis | Statistics

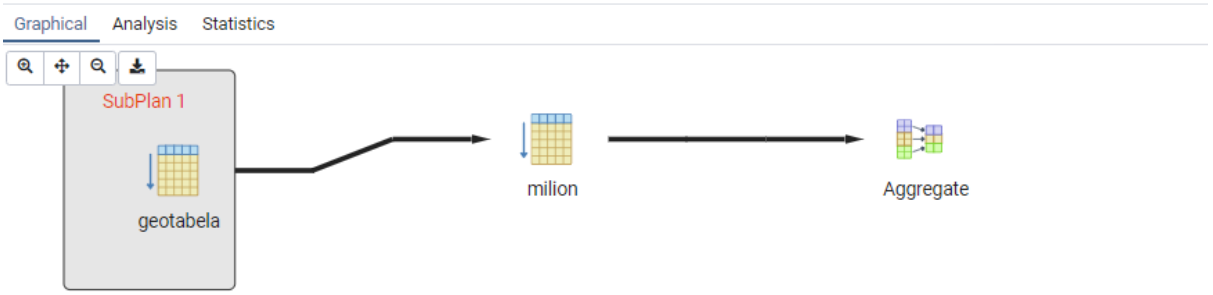
#	Node	Rows Actual	Loops
1.	→ Aggregate (rows=1 loops=1)	1	1
2.	→ Seq Scan on milion as milion (rows=985294 loops=1) **Filter**: ((liczba % 68) = (SubPlan 1)) **Rows Removed by Filter**: 14706	985294	1
3.	→ Seq Scan on geotabela as geotabela (rows=1 loops=1000000) **Filter**: ((milion.liczba % 68) = id_pietro) **Rows Removed by Filter**: 101	1	1000000



Rys.9. Zapytanie 3 - wersja z wykorzystaniem indeksu.

Graphical | Analysis | Statistics

#	Node	Rows Actual	Loops
1.	→ Aggregate (rows=1 loops=1)	1	1
2.	→ Seq Scan on milion as milion (rows=985294 loops=1) **Filter**: ((liczba % 68) = (SubPlan 1)) **Rows Removed by Filter**: 14706	985294	1
3.	→ Seq Scan on geotabela as geotabela (rows=1 loops=1000000) **Filter**: ((milion.liczba % 68) = id_pietro) **Rows Removed by Filter**: 101	1	1000000

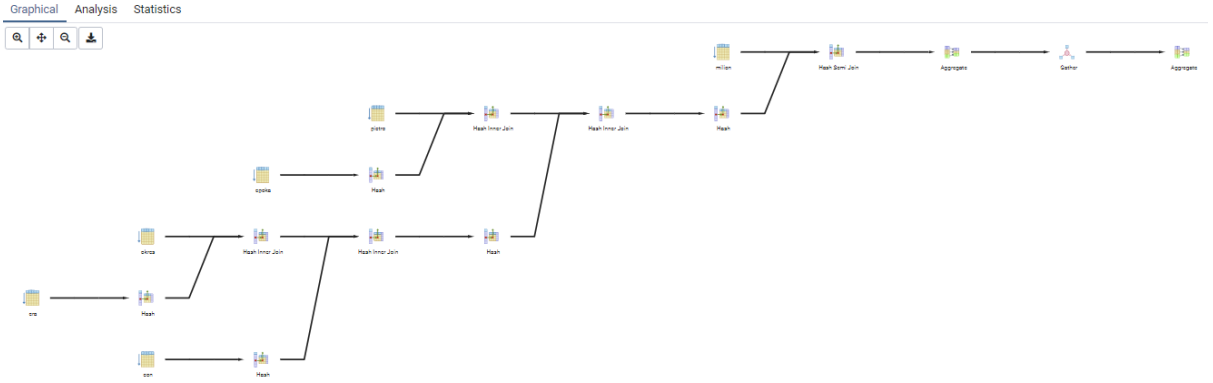


Rys.10. Zapytanie 3 - wersja bez indeksu.

Graphical | Analysis | Statistics

#	Node	Rows Actual	Loops
1.	→ Aggregate (rows=1 loops=1)	1	1
2.	→ Gather (rows=3 loops=1)	3	1
3.	→ Aggregate (rows=1 loops=3)	1	3
4.	→ Hash Semi Join (rows=328431 loops=3) **Hash Cond**: ((milion.liczba % 68) = pietro.id_pietro)	328431	3
5.	→ Seq Scan on milion as milion (rows=333333 loops=3)	333333	3
6.	→ Hash (rows=102 loops=3) **Buckets**: 1024 **Batches**: 1 **Memory Usage**: 12 kB	102	3
7.	→ Hash Inner Join (rows=102 loops=3) **Hash Cond**: ((epoka.id_okres)::text = (okres.id_okres)::text)	102	3
8.	→ Hash Inner Join (rows=102 loops=3) **Hash Cond**: ((pietro.id_epoka)::text = (epoka.id_epoka)::text)	102	3
9.	→ Seq Scan on pietro as pietro (rows=102 loops=3)	102	3
10.	→ Hash (rows=34 loops=3) **Buckets**: 1024 **Batches**: 1 **Memory Usage**: 10 kB	34	3
11.	→ Seq Scan on epoka as epoka (rows=34 loops=3)	34	3
12.	→ Hash (rows=12 loops=3) **Buckets**: 1024 **Batches**: 1 **Memory Usage**: 9 kB	12	3
13.	→ Hash Inner Join (rows=12 loops=3) **Hash Cond**: ((era.id_eon)::text = (eon.id_eon)::text)	12	3
14.	→ Hash Inner Join (rows=12 loops=3) **Hash Cond**: ((okres.id_era)::text = (era.id_era)::text)	12	3
15.	→ Seq Scan on okres as okres (rows=12 loops=3)	12	3
16.	→ Hash (rows=3 loops=3) **Buckets**: 1024 **Batches**: 1 **Memory Usage**: 9 kB	3	3
17.	→ Seq Scan on era as era (rows=3 loops=3)	3	3
18.	→ Hash (rows=1 loops=3) **Buckets**: 1024 **Batches**: 1 **Memory Usage**: 9 kB	1	3
19.	→ Seq Scan on eon as eon (rows=1 loops=3)	1	3

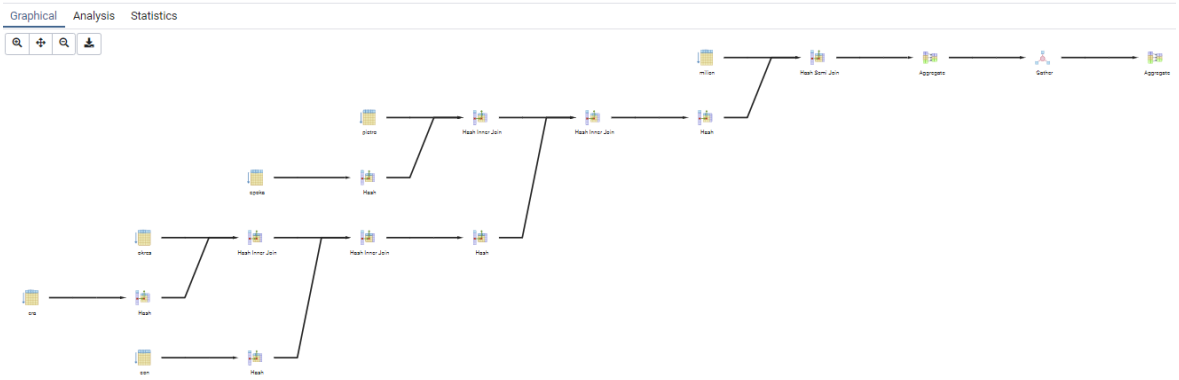
Graphical | Analysis | Statistics



Rys.11. Zapytanie 4 - wersja z wykorzystaniem indeksu.

Graphical | Analysis | Statistics

#	Node	Rows	
		Actual	Loops
1.	→ Aggregate (rows=1 loops=1)	1	1
2.	→ Gather (rows=3 loops=1)	3	1
3.	→ Aggregate (rows=1 loops=3)	1	3
4.	→ Hash Semi Join (rows=328431 loops=3) **Hash Cond**: ((milion.liczba % 68) = pietro.id_pietro)	328431	3
5.	→ Seq Scan on milion as milion (rows=333333 loops=3)	333333	3
6.	→ Hash (rows=102 loops=3) **Buckets**: 1024 **Batches**: 1 **Memory Usage**: 12 kB	102	3
7.	→ Hash Inner Join (rows=102 loops=3) **Hash Cond**: ((epoka.id_okres)::text = (okres.id_okres)::text)	102	3
8.	→ Hash Inner Join (rows=102 loops=3) **Hash Cond**: ((pietro.id_epoka)::text = (epoka.id_epoka)::text)	102	3
9.	→ Seq Scan on pietro as pietro (rows=102 loops=3)	102	3
10.	→ Hash (rows=34 loops=3) **Buckets**: 1024 **Batches**: 1 **Memory Usage**: 10 kB	34	3
11.	→ Seq Scan on epoka as epoka (rows=34 loops=3)	34	3
12.	→ Hash (rows=12 loops=3) **Buckets**: 1024 **Batches**: 1 **Memory Usage**: 9 kB	12	3
13.	→ Hash Inner Join (rows=12 loops=3) **Hash Cond**: ((era.id_eon)::text = (eon.id_eon)::text)	12	3
14.	→ Hash Inner Join (rows=12 loops=3) **Hash Cond**: ((okres.id_era)::text = (era.id_era)::text)	12	3
15.	→ Seq Scan on okres as okres (rows=12 loops=3)	12	3
16.	→ Hash (rows=3 loops=3) **Buckets**: 1024 **Batches**: 1 **Memory Usage**: 9 kB	3	3
17.	→ Seq Scan on era as era (rows=3 loops=3)	3	3
18.	→ Hash (rows=1 loops=3) **Buckets**: 1024 **Batches**: 1 **Memory Usage**: 9 kB	1	3
19.	→ Seq Scan on eon as eon (rows=1 loops=3)	1	3



Rys.12. Zapytanie 4 - wersja bez indeksu.

WNIOSKI

Na podstawie otrzymanych wyników, zaprezentowanych w **Tabeli 7** oraz **Tabeli 8**, można stwierdzić, że nałożenie indeksów na wszystkie kolumny biorące udział w danym zapytaniu, poprawia wydajność jego działania. Porównując wybrane systemy baz danych okazało się, że czas wykonania zapytań jest krótszy w przypadku SQL Server. Analizując plany wykonania zapytań - szczególnie w SQL Server (**Rys.1**, …, **Rys.4**) - nie zauważymy widocznej różnicy między wariantami wykonania zapytań z indeksami lub bez, ponieważ w obu przypadkach jest wykorzystywana operacja Scan, przeszukująca całą tabelę. Różnice możemy dostrzec natomiast porównując wspomniane wcześniej tabele: **Tabela 7** oraz **Tabela 8** przedstawiające czasy wykonania zapytań.