WYDAJNOŚĆ ZŁĄCZEŃ I ZAGNIEŻDŻEŃ DLA SCHEMATÓW ZNORMALIZOWANYCH I ZDENORMALIZOWANYCH

Sprawozdanie w oparciu o artykuł o tej samej nazwie autorstwa Łukasza JAJEŚNICY, Adama PIÓRKOWSKIEGO

Kod do sprawozdania umieszczony jest osobno dla postgresql i sql server.

1. Wstęp

W sprawozdaniu został poruszony temat optymalizacji w bazach danych. Poniżej przedstawiono rozważania na temat wydajnościowych aspektów normalizacji schematów baz danych. Badania oparto o przykład jakim jest tabela stratygraficzna. Przeprowadzono eksperymenty sprawdzające wydajność dla złączeń i zagnieżdżeń skorelowanych dla systemów zarządzania bazami PostgreSQL oraz SQL server.

2. Konfiguracja sprzętowa i programowa

CPU: Intel(R) Core(TM) i5-4200M CPU @ 2.50GHz 2.50GHz

Zainstalowana pamięć RAM 16,0 GB

Wersja systemu: Windows 10 Pro 64-bitowy

Dysk Twardy: LITEONIT LCS-256L9S-11 2.5 7mm 256GB

PostgreSQL wersja - 13

SQL server wersja - 15.0.18369.0

4. Kryteria testów

W teście wykonano szereg zapytań sprawdzających wydajność złączeń i zagnieżdżeń z tabelą geochronologiczną w wersji zdenormalizowanej i znormalizowanej. Procedurę testową przeprowadzono w dwóch etapach:

>pierwszy etap obejmował zapytania bez nałożonych indeksów na kolumny danych (jedynymi indeksowanymi danymi były dane w kolumnach będących kluczami głównymi po-szczególnych tabel,),

> w drugim etapie nałożono indeksy na wszystkie kolumny biorące udział w złączeniu. Zasadniczym celem testów była ocena wpływu normalizacji na zapytania złożone –złączenia i zagnieżdżenia (skorelowane) [14]. W tym celu zaproponowano cztery zapytania:

> Zapytanie 1 (1 ZL), którego celem jest złączenie syntetycznej tablicy miliona wyników z 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,68)=(GeoTabela.id_pietro));
```

>Zapytanie 2 (2 ZL), którego celem 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,68)=GeoPietro.id_pietro) NATURAL JOIN GeoEpoka NATURAL JOIN
GeoOkres NATURAL JOIN GeoEra NATURAL JOIN GeoEon;
```

>Zapytanie 3 (3 ZG), którego celem 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,68)= (SELECT id_pietro FROM
GeoTabela   WHERE mod(Milion.liczba,68)=(id_pietro));
```

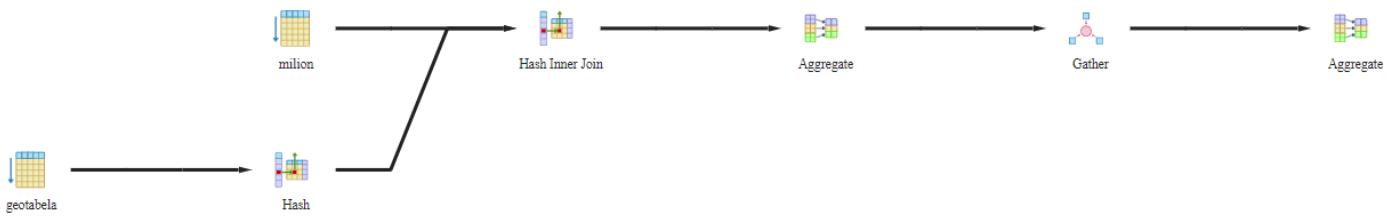
>Zapytanie 4 (4 ZG), którego celem jest złączenie syntetycznej tablicy miliona wyników z tabelą geochronologiczną w postaci znormalizowanej, przy czym złączenie jest wykonywane poprzez zagnieżdżenie skorelowane, a zapytanie wewnętrzne jest złączeniem ta-bel poszczególnych jednostek geochronologicznych:

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

BEZ INDEKSÓW

1 zapytanie:

POSTGRESQL



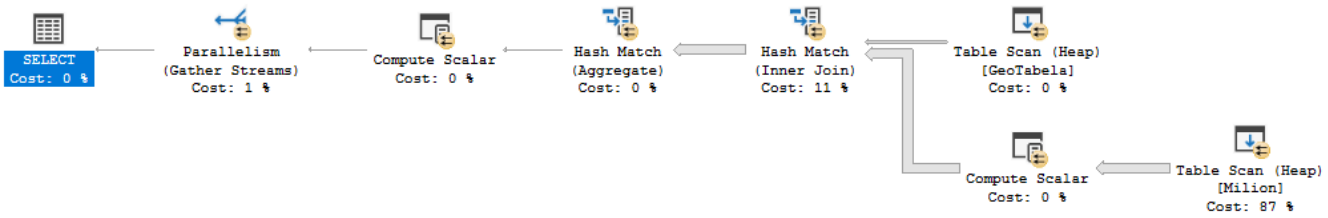
Statistics per Node Type

Node type	Count
Aggregate	2
Gather	1
Hash	1
Hash Inner Join	1
Seq Scan	2

Statistics per Relation

Relation name	Scan count
Node type	**Count**
geotabela	1
Seq Scan	1
milion	1
Seq Scan	1

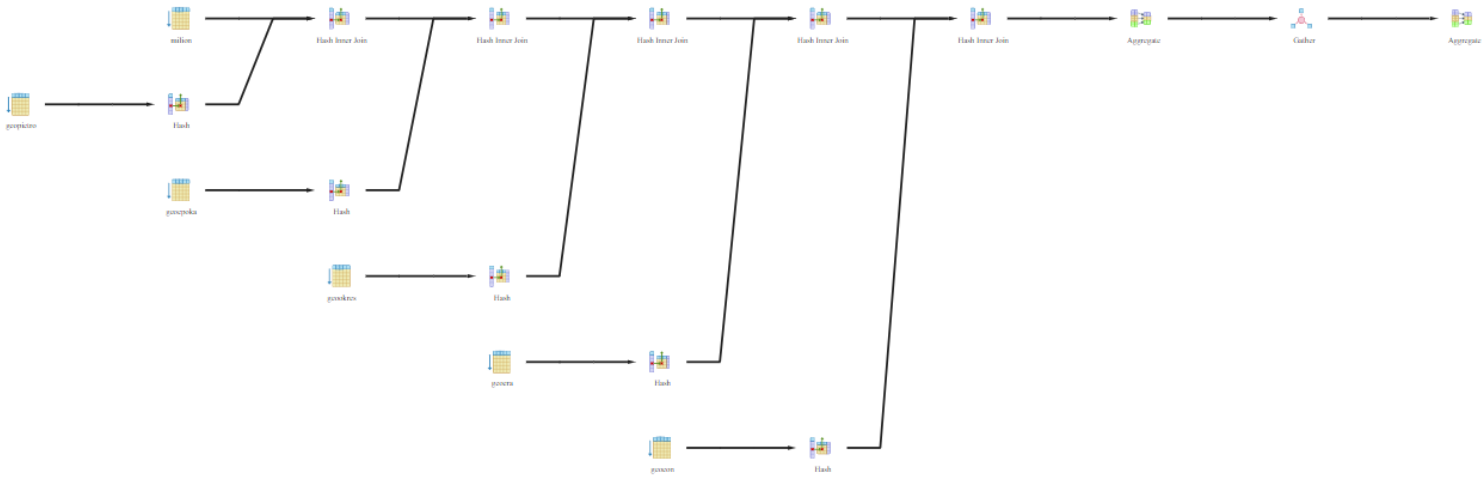
SQL SERVER



Na przedstawionych screnshotach widzimy, że w przypadku Postgresa wykorzystano Seq Scan, natomiast w przypadku SQL server Table Scan. Na tej podstawie możemy stwierdzić że baza wymaga optymalizacji

2 zapytanie:

POSTGRESQL



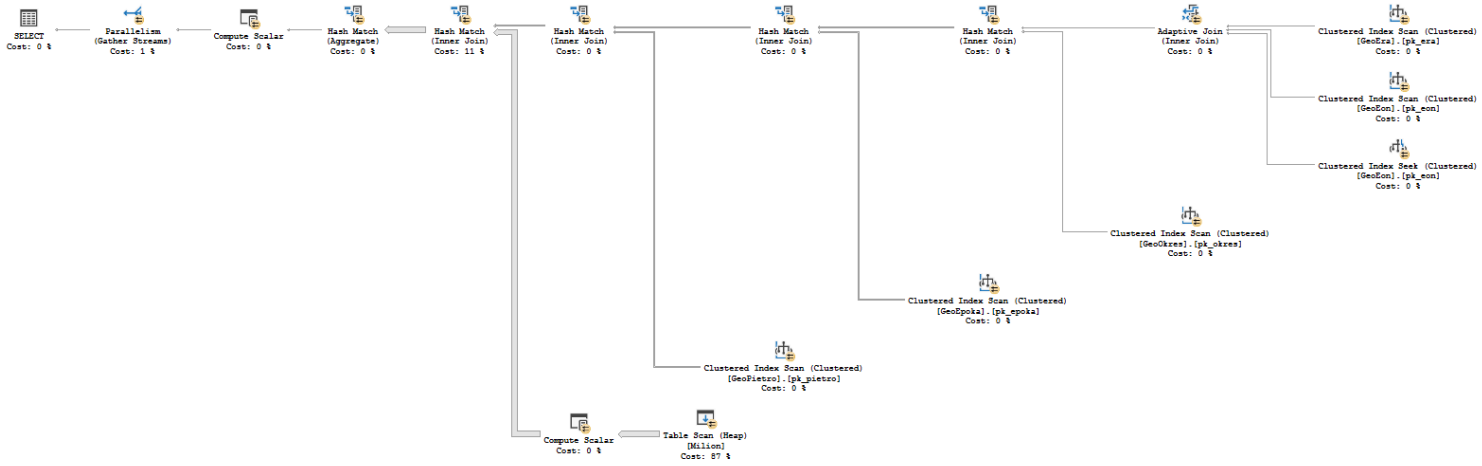
Statistics per Node Type

Node type	Count
Aggregate	2
Gather	1
Hash	5
Hash Inner Join	5
Seq Scan	6

Statistics per Relation

Relation name	Scan count
Node type	Count
geoeon	1
Seq Scan	1
geoepoka	1
Seq Scan	1
geoera	1
Seq Scan	1
geookres	1
Seq Scan	1
geopietro	1
Seq Scan	1
milion	1
Seq Scan	1

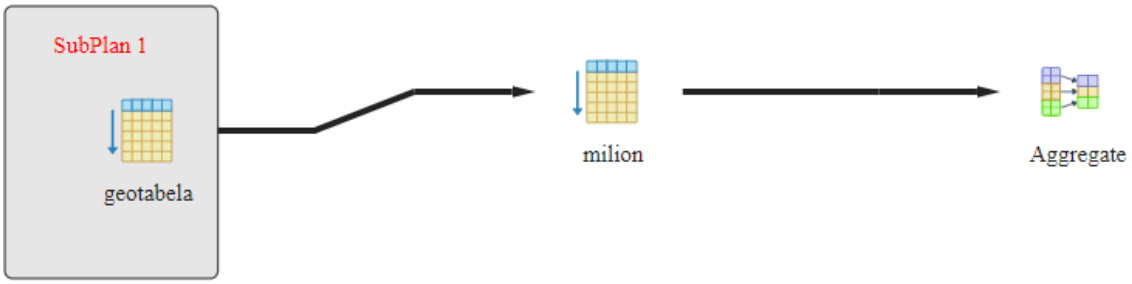
SQL SERVER



W drugim zapytaniu również mymy Seq Scan w Postgresie, w SQL server natomiast pojawiły się Cluster Index Scan wiele razy wraz z Table Scan i Compute Scalar.

3 zapytanie:

POTGRESQL



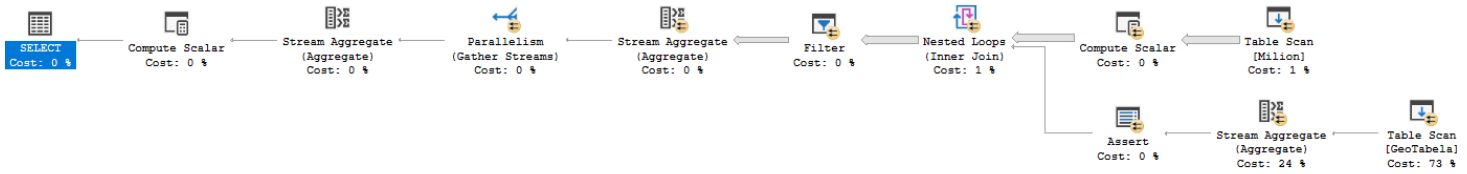
Statistics per Node Type

Node type	Count
Aggregate	1
Seq Scan	2

Statistics per Relation

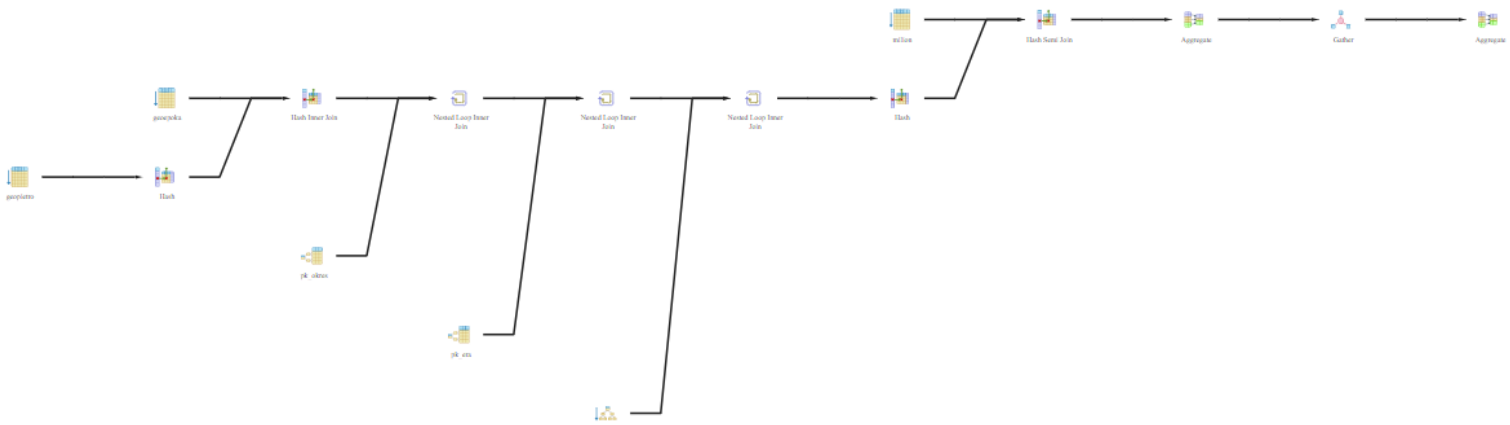
Relation name	Scan count
Node type	Count
geotabela	1
Seq Scan	1
milion	1
Seq Scan	1

SQL SERVER



4 zapytanie:

POSTRGESQL



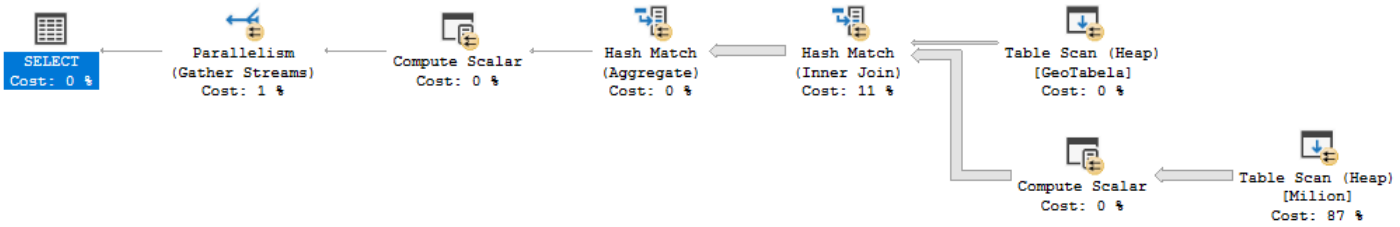
Statistics per Node Type

Node type	Count
Aggregate	2
Gather	1
Hash	2
Hash Inner Join	1
Hash Semi Join	1
Index Only Scan	1
Index Scan	2
Nested Loop Inner Join	3
Seq Scan	3

Statistics per Relation

Relation name	Scan count
Node type	**Count**
geoeon	1
Index Only Scan	1
geoepoka	1
Seq Scan	1
geoera	1
Index Scan	1
geookres	1
Index Scan	1
geopietro	1
Seq Scan	1
milion	1
Seq Scan	1

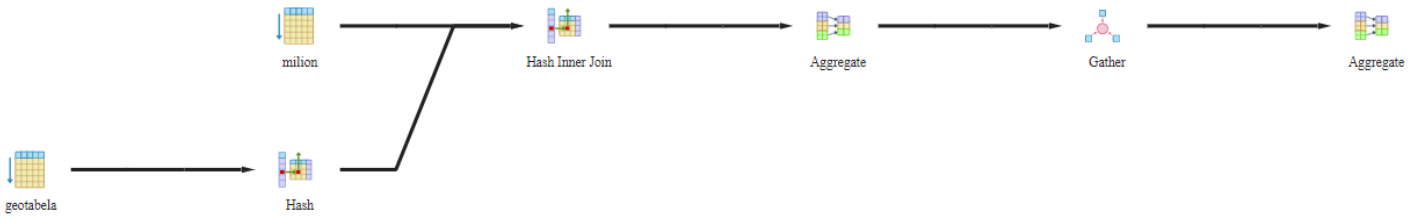
SQL SERVER



Z INDEKSAMI

1 zapytanie:

POSTGRESQL



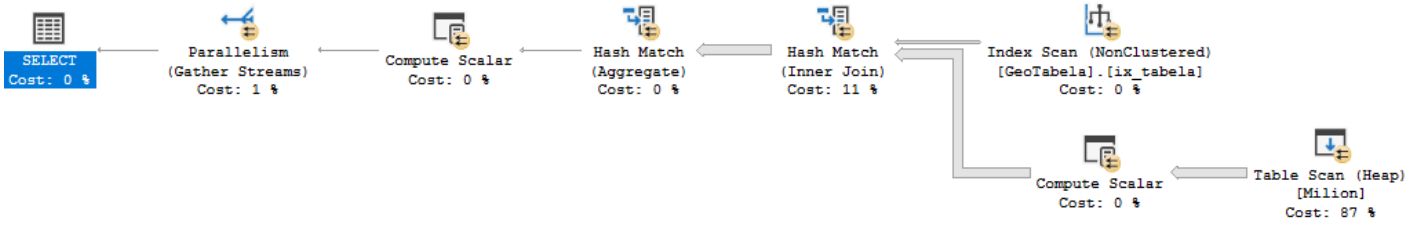
Statistics per Node Type

Node type	Count
Aggregate	2
Gather	1
Hash	1
Hash Inner Join	1
Seq Scan	2

Statistics per Relation

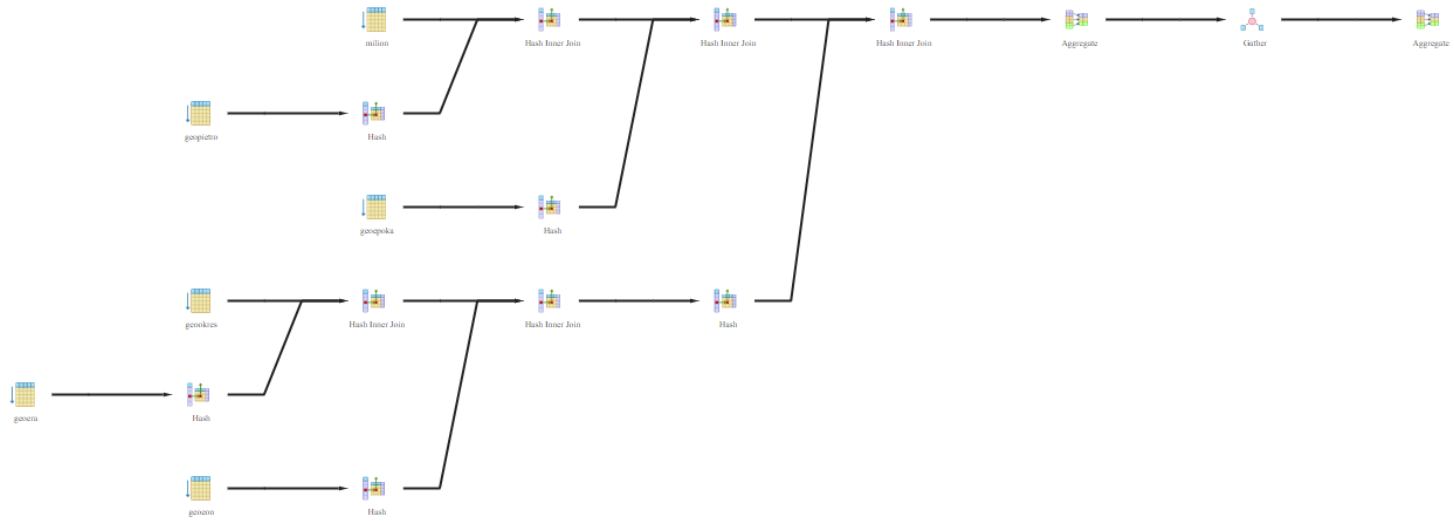
Relation name	Scan count
Node type	Count
geotabela	1
Seq Scan	1
milion	1
Seq Scan	1

SQL SERVER



2 zapytanie:

POSTGRESQL



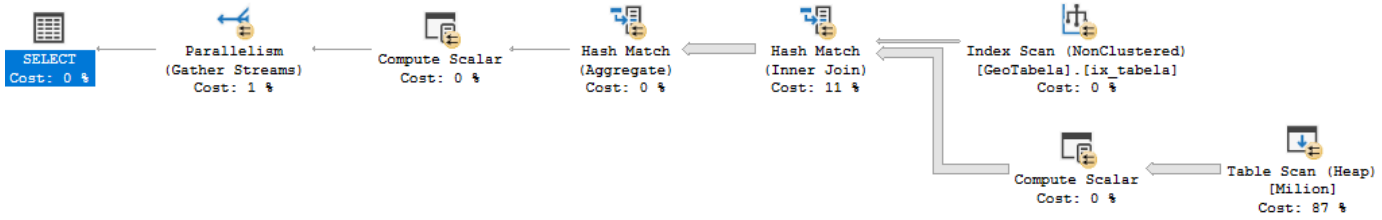
Statistics per Node Type

Node type	Count
Aggregate	2
Gather	1
Hash	5
Hash Inner Join	5
Seq Scan	6

Statistics per Relation

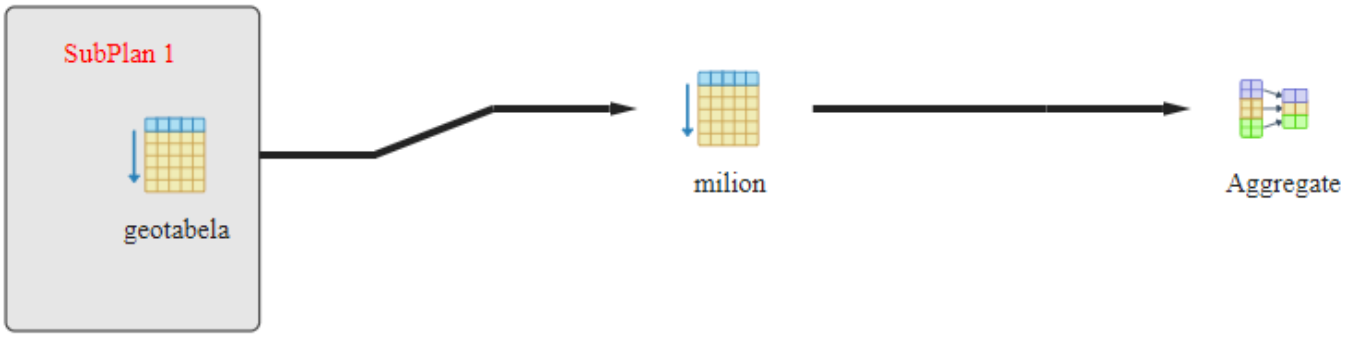
Relation name	Scan count
Node type	Count
geoeon	1
Seq Scan	1
geoepoka	1
Seq Scan	1
geoera	1
Seq Scan	1
geookres	1
Seq Scan	1
geopietro	1
Seq Scan	1
milion	1
Seq Scan	1

SQL SERVER



3 zapytanie:

POSTGRESQL



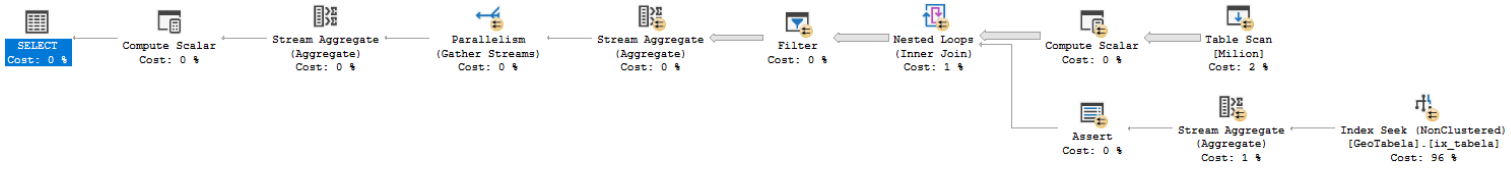
Statistics per Node Type

Node type	Count
Aggregate	1
Seq Scan	2

Statistics per Relation

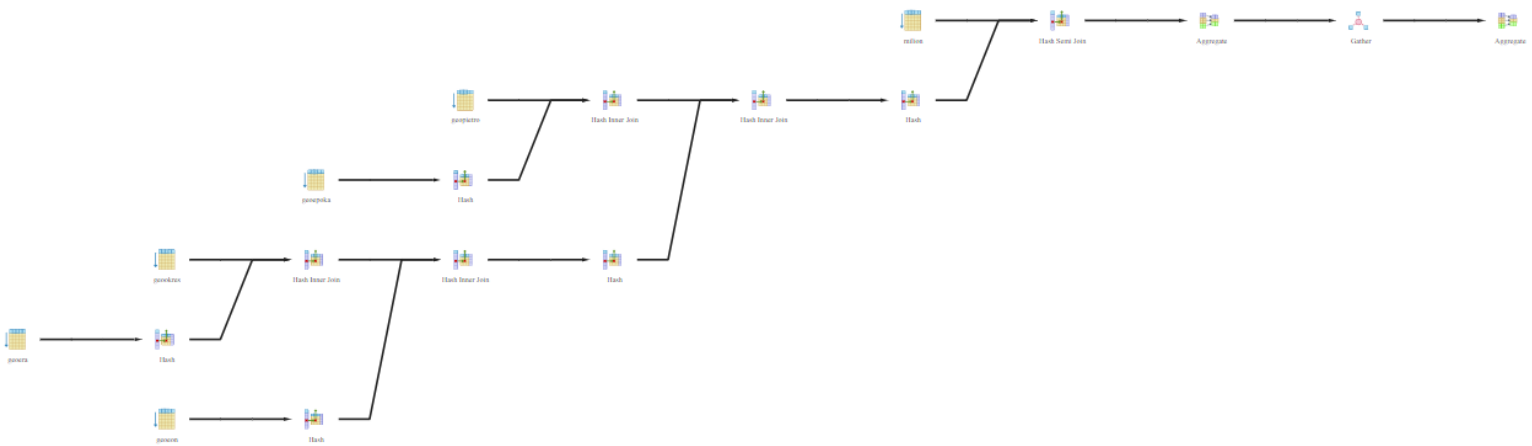
Relation name	Scan count
Node type	Count
geotabela	1
Seq Scan	1
milion	1
Seq Scan	1

SQL SERVER



4 zapytanie:

POSTGRESQL



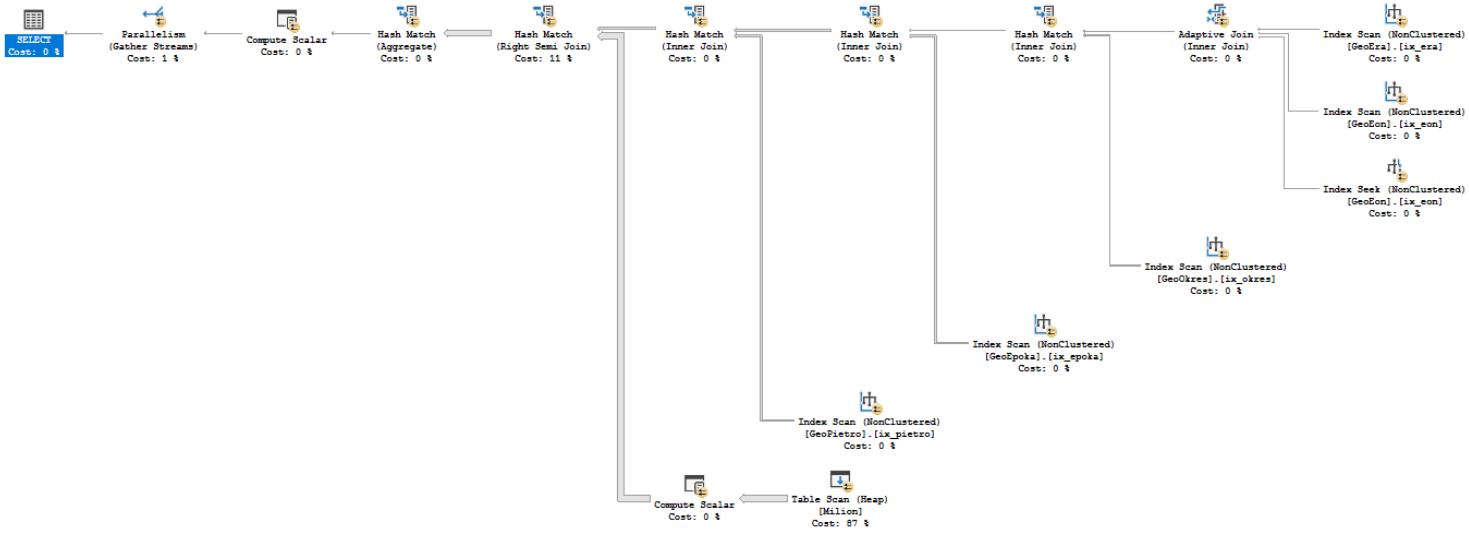
Statistics per Node Type

Node type	Count
Aggregate	2
Gather	1
Hash	5
Hash Inner Join	4
Hash Semi Join	1
Seq Scan	6

Statistics per Relation

Relation name	Scan count
Node type	Count
geoeon	1
Seq Scan	1
geoepoka	1
Seq Scan	1
geoera	1
Seq Scan	1
geookres	1
Seq Scan	1
geopietro	1
Seq Scan	1
milion	1
Seq Scan	1

SQL SERVER



		BEZ INDEKSÓW			
	Próby	zap1	zap2	zap3	zap4
postgres	1.	366	734	15753	324
	2.	349	765	16400	328
	3.	355	854	15139	314
	4.	358	844	15124	333
	średnia	357	799,25	15604	324,75
sql server cpu	1.	188	234	22327	218
	2.	172	235	22500	188
	3.	202	202	23140	282
	4.	218	220	22404	234
	średnia	195	222,75	22592,75	230,5
sql server elapsed	1.	93	82	6948	83
	2.	75	59	7174	75
	3.	75	80	7089	77
	4.	74	80	7141	76
	średnia	79,25	75,25	7088	77,75

		Z INDEKSAMI			
	Próby	zap1	zap2	zap3	zap4
postgres	1.	330	614	14921	337
	2.	359	594	15150	336
	3.	368	606	15031	319
	4.	360	559	15169	343
	średnia	354,25	593,25	15067,75	333,75
sql server cpu	1.	219	217	6733	233
	2.	188	235	6812	250
	3.	155	187	6828	204
	4.	156	204	6640	187
	średnia	179,5	210,75	6753,25	218,5
sql server elapsed	1.	79	72	2258	68
	2.	76	82	2297	88
	3.	87	85	2425	91
	4.	77	84	2347	109
	średnia	79,75	80,75	2331,75	89

OSTATECZNE PORÓWNANIE

Tabele przedstawiają czas wykoniania poszczególnych zapytań w milisekundach w czterech próbach. Po lewej stornie widnieją wyniki bez uwzględnienia indeksów, natomiast z prawej z ich uwzględnieniem. W przypadku sql server czas wyniki zostały podzielone na czas procesora oraz ‘elapsed time’ – czyli predkość pamięci masowej.

WNIOSKI

Patrząc na tabele dużą różnice możemy zaobserwować jedynie w przypadku trzeciego zapytania dla bazy obsługiwanej przes sql server. W pozostałych zapytaniach indeksy niestety nie sprawdziły się. Po screnach zamieszczonych powyżej tabeli możemy zaobserwować użycie indeksów klastrowych w niektórych przypadkach , jednakże jedynie w sql server.

Dochodzimy więc do konkluzji iż używanie indeksów sprawdza się jedynie w określonych zapytaniach i nie zawsze jest pomcne w pracy na bazach.