Porównanie wydajności złączeń i zagnieżdżeń dla schematów znormalizowanych i zdenrmalizowanych

Maciej Bąk

5 czerwca 2021

1 Wstęp

Przedmiotem analizy było zbadanie i porównanie wydajności kwerend bazujących na złączeniach i zagnieżdżeniach dla schematów znormalizowanych i zdenromalizowanych. Analizę przeprowadzono na dwóch różnych systemach bazodanowych opierających się o koncept relacyjnych baz danych. W trakcie badania wzorowano się na artykule *Wydajność złączeń i zagnieżdżeń dla schematów znormalizowanych i zdenormalizowanych*

2 Konfiguracja sprzętowa i programowa

Testy omówione w niniejszym opracowaniu przeprowadzono na komputerze o następującej specyfikacji:

- CPU: Intel Core i5-8250U, 6M Cache, up to 3.40 GHz
- GPU: NVIDIA GeForce MX150
- RAM: 8 GB
- SSD: M.2
- S.O: Ubuntu 20.04.2 LTS
- PostgreSQL 13.2
- MySQL 8.0.25

3 Metodyka testów

W celu zautomatyzowania procedury testowej stworzyłem odpowiednie skrypty uruchamiające wcześniej zdefiniowane kwerendy oraz polecenia. Pozwoliło to na znaczne uproszczenie testowania oraz dobieranie ilości przeprowadzanych serii pomiarowych według potrzeb. Wszystkie niezbędne skrypty zostały umieszczone w repozytorium na GitHubie. Serię testów

wyzwala uruchomienie skryptu startBench.sh (uwaga! skrypty będą działać nieprawidłowo na komputerze innym od tego wykorzystanego do testów). Najpierw testowana jest baza MySQL z wykorzystaniem zapytań 1ZL, 2ZL, 3ZG, 4ZG (opisane w dalszej części opracowania) bez założonych indeksów nieklastrowanych. Kolejno, skrypt tworzy indeksy dla odpowiednich kolumn i ponawia procedurę testowania czterema zapytaniami. Czas wykonywania poszczególnych zapytań jest zapisywany do odpowiednich plików. Pliki te są następnie odpowiednio przetwarzane w celu obliczenia średniej wartości czasu trwania poszczególnych kwerend. Testy zostały przeprowadzone w analogiczny sposób dla MySQL oraz PostgreSQL. Po wykonaniu zadań skrypt prezentuje uzyskane wyniki.

```
################################
MySQL - without indexes
        1 ZL: 576
        2 ZL: 2433
        3 ZG: 40138
        4 ZG: 2488
MySQL - with indexes
        1 ZL: 1789
        2 ZL: 2345
        3 ZG: 2393
        4 ZG: 2264
################################
PostgreSQL - without indexes
        1 ZL: 102
        2 ZL: 176
        3 ZG: 5601
        4 ZG: 100
PostgreSQL - with indexes
        1 ZL: 70
        2 ZL: 154
        3 ZG: 4980
        4 ZG: 77
```

Rys. 1. Przykładowy rezultat działania skryptu testującego

Bazę danych przygotowano wzorując się na tabeli geochronologicznej. Przygotowano ją w wersji znormalizowanej gdzie poszczególne jednostki czasu geologicznego zostały powiązane ze sobą odpowiednimi relacjami oraz w postaci zdenromalizowanej (GeoTabela) - jedna tabela zawierająca wszystkie wykorzystane dane geochronologiczne.

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

4 Zapytania testowe

W celu przeprowadzenia testów stworzono tabelę Milion zawierającą syntetyczne dane o jednorodnym rozkładzie od 0 do 999 999

Jak wspomniano w poprzednim pragrafie, do wykonania testów użyto czterech zapytań oznaczonych jako 1ZL, 2ZL, 3ZG, 4ZG.

4.1 1ZL

Złączenie tabeli Milion z tabelą geochronologiczną GeoTabela w postaci zdenormalizowanej:

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

4.2 2ZL

Złączenie tabeli Milion z tabelą geochronologiczną w postaci znormalizowanej (złączenie pięciu tabel):

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

4.3 3ZG

Złączenie tabeli Milion z tabelą geochronologiczną GeoTabela w postaci zdenormalizowanej poprzez zagnieżdżenie skorelowane:

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

4.4 4ZG

Złączenie tabeli Milion z tabelą geochronologiczną w postaci znormalizowanej poprzez zagnieżdżenie skorelowane:

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

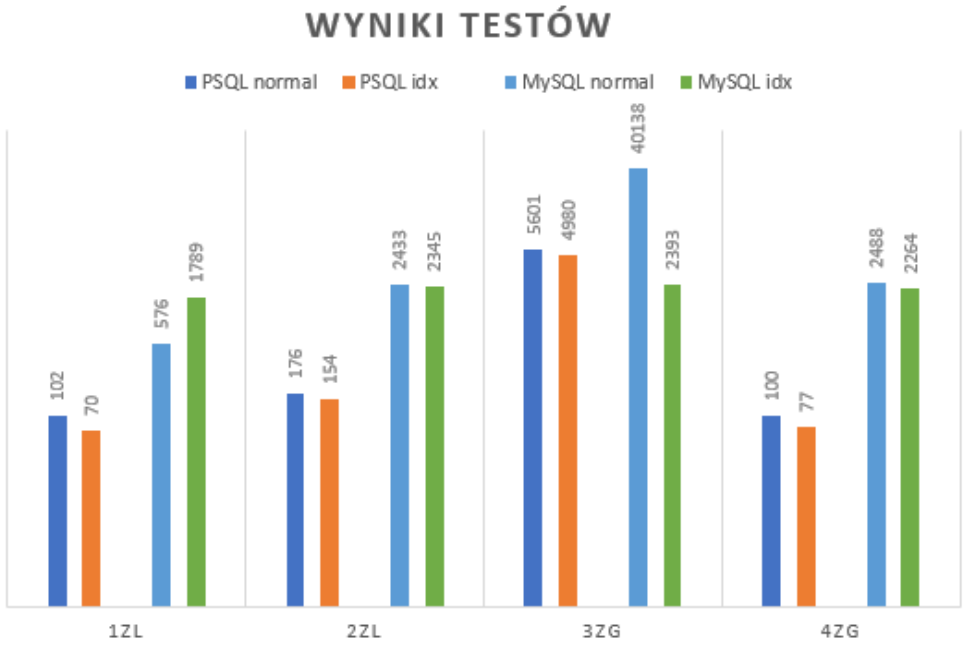
5 Analiza wyników

Przeprowadzono 25 serii testów. Wyniki testów w postaci średnich czasów trwania poszczególnych zapytań przedstawiono w tabeli poniżej. Pełny zestaw wyników z czasami poszczególnych zapytań dostępny jest w załączonym repozytorium.

	1ZL	**2ZL**	**3ZG**	**4ZG**
Bez indeksów				
MySQL	576	2433	40138	2488
PostgreSQL	102	176	5601	100
Z indeskami				
MySQL	1789	2345	2393	2264
PostgreSQL	70	154	4980	77

Tab. 1. Porównanie średnich czasów zapytań [ms]

Wyniki przedstawiono na wykresie poniżej, użyto skali logarytmicznej w celu łatwiejszego wartości odstających.



Rys. 2. Wyniki testów w skali logarytmicznej (wartości w [ms])

Przeanalizowano również plany wykonywania poszczególnych zapytań. Stworzono je dla każdej konfiguracji zapytań (bez i z indeksami) dla dwóch baz danych. Wszystkie rezultaty umieszczono na końcu sprawozdania (pierwsza tabela to zapytania bez indeksacji, druga z indeksami. Odpowiednio dla MySQL i PostgreSQL).

6 Wnioski

W ogólności indeksacja poprawiła wydajność wykonywanych zapytań. Przypadkiem odbiegającym od normy jest zapytanie 1ZL dla MySQL, gdzie to indeksacja nie zmniejszyła czasu wykonywanych zapytań. Być może taka sytuacja nie jest związana bezpośrednio z samym procesem indeksacji a innym, obciążającym procesem który działał w tej chwili na komputerze. Zdecydowanie największą poprawę wydajności widać dla zapytania 3ZG w bazie MySQL po dodaniu indeksów. W PostgreSQL, dla tego zapytania również zaobserwowano poprawę, jednak nie jest ona tak znacząca. Ważnym wniosek jest obserwacja, że zagnieżdżenia skorelowane są z reguły wolniejsze od złączeń. Dodatkową, wartą uwagi obserwacją jest fakt, że w większość przypadków czas wykonywania zapytań był mniejszy w PostgreSQL.

Bibliografia

[1] Ł.Jajeśnica, A.Piórkowski, *Wydajność złączeń ii zagnieżdżeń dla schematów znormalizowanych i zdenormalizowanych*, (Studia Ingormatica, Volume 31, Number 2A (89), 2010).

1ZL

-- 1ZL EXPLAIN SELECT COUNT(*) FROM milion INNER JOIN geotabela ON (mod(milion.liczba,68)=(geotabela.id_pietro))											
id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	geotabela		ALL					77	100	
1	SIMPLE	milion		ALL					997,920	100	Using where; Using join buffer (hash join)

2ZL

-- 2ZL EXPLAIN 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											
id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	geoeon		index	PRIMARY	PRIMARY	4		1	100	Using index
1	SIMPLE	milion		ALL					997,920	100	Using join buffer (hash join)
1	SIMPLE	geopietro		eq_ref	PRIMARY,id_epoka	PRIMARY	4	func	1	100	Using where
1	SIMPLE	geoepoka		eq_ref	PRIMARY,id_okres	PRIMARY	4	geomysql.geopietro.id_epoka	1	100	Using where
1	SIMPLE	geookres		eq_ref	PRIMARY,id_era	PRIMARY	4	geomysql.geoepoka.id_okres	1	100	Using where
1	SIMPLE	geoera		eq_ref	PRIMARY,id_eon	PRIMARY	4	geomysql.geookres.id_era	1	100	Using where

3ZG

-- 3ZG EXPLAIN SELECT COUNT(*) FROM milion WHERE mod(milion.liczba,68)= (SELECT id_pietro FROM geotabela WHERE mod(milion.liczba,68)= (id_pietro))											
id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	PRIMARY	milion		ALL					997,920	100	Using where
2	DEPENDENT SUBQUERY	geotabela		ALL					77	10	Using where

4ZG

EXPLAIN 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)											
id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	geoeon		index	PRIMARY	PRIMARY	4		1	100	Using index
1	SIMPLE	milion		ALL					997,920	100	Using join buffer (hash join)
1	SIMPLE	geopietro		eq_ref	PRIMARY,id_epoka	PRIMARY	4	func	1	100	Using where
1	SIMPLE	geoepoka		eq_ref	PRIMARY,id_okres	PRIMARY	4	geomysql.geopietro.id_epoka	1	100	Using where
1	SIMPLE	geookres		eq_ref	PRIMARY,id_era	PRIMARY	4	geomysql.geoepoka.id_okres	1	100	Using where
1	SIMPLE	geoera		eq_ref	PRIMARY,id_eon	PRIMARY	4	geomysql.geookres.id_era	1	100	Using where

1ZL

-- 1ZL EXPLAIN SELECT COUNT(*) FROM milion INNER JOIN geotabela ON (mod(milion.liczba,68)=(geotabela.id_pietro))											
id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	milion		index		liczba_idx	5		997,920	100	Using index
1	SIMPLE	geotabela		eq_ref	id_pietro_idx	id_pietro_idx	4	func	1	100	Using where; Using index

2ZL

-- 2ZL EXPLAIN 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											
id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	geoeon		index	PRIMARY,id_eon_idx	id_eon_idx	4		1	100	Using index
1	SIMPLE	milion		index		liczba_idx	5		997,920	100	Using index; Using join buffer (hash join)
1	SIMPLE	geopietro		eq_ref	PRIMARY,id_pietroG_idx,id_epoka	PRIMARY	4	func	1	100	Using where
1	SIMPLE	geoepoka		eq_ref	PRIMARY,id_epoka_idx,id_okres	PRIMARY	4	geomysql.geopietro.id_epoka	1	100	Using where
1	SIMPLE	geookres		eq_ref	PRIMARY,id_okres_idx,id_era	PRIMARY	4	geomysql.geoepoka.id_okres	1	100	Using where
1	SIMPLE	geoera		eq_ref	PRIMARY,id_era_idx,id_eon	PRIMARY	4	geomysql.geookres.id_era	1	100	Using where

3ZG

-- 3ZG EXPLAIN SELECT COUNT(*) FROM milion WHERE mod(milion.liczba,68)= (SELECT id_pietro FROM geotabela WHERE mod(milion.liczba,68)=(id_pietro))											
id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	PRIMARY	milion		index		liczba_idx	5		997,920	100	Using where; Using index
2	DEPENDENT SUBQUERY	geotabela		eq_ref	id_pietro_idx	id_pietro_idx	4	func	1	100	Using where; Using index

4ZG

EXPLAIN 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)											
id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	geoeon		index	PRIMARY,id_eon_idx	id_eon_idx	4		1	100	Using index
1	SIMPLE	milion		index		liczba_idx	5		997,920	100	Using index; Using join buffer (hash join)
1	SIMPLE	geopietro		eq_ref	PRIMARY,id_pietroG_idx,id_epoka	PRIMARY	4	func	1	100	Using where
1	SIMPLE	geoepoka		eq_ref	PRIMARY,id_epoka_idx,id_okres	PRIMARY	4	geomysql.geopietro.id_epoka	1	100	Using where
1	SIMPLE	geookres		eq_ref	PRIMARY,id_okres_idx,id_era	PRIMARY	4	geomysql.geoepoka.id_okres	1	100	Using where
1	SIMPLE	geoera		eq_ref	PRIMARY,id_era_idx,id_eon	PRIMARY	4	geomysql.geookres.id_era	1	100	Using where

1ZL

EXPLAIN SELECT COUNT(*) FROM geochrono.Milion INNER JOIN geochrono.GeoTabela ON (mod(Milion.liczba,68)= (GeoTabela.id_pietro))
QUERY PLAN
Finalize Aggregate (cost=14664.16..14664.17 rows=1 width=8)
-> Gather (cost=14663.95..14664.16 rows=2 width=8)
Workers Planned: 2
-> Partial Aggregate (cost=13663.95..13663.96 rows=1 width=8)
-> Hash Join (cost=2.73..13262.90 rows=160417 width=0)
Hash Cond: (mod(milion.liczba, 68) = geotabela.id_pietro)
-> Parallel Seq Scan on milion (cost=0.00..9572.67 rows=416667 width=4)
-> Hash (cost=1.77..1.77 rows=77 width=4)
-> Seq Scan on geotabela (cost=0.00..1.77 rows=77 width=4)

2ZL

EXPLAIN SELECT COUNT(*) FROM geochrono.Milion INNER JOIN geochrono.GeoPietro ON (mod(Milion.liczba,68)=geochrono.GeoPietro.id_pietro) NATURAL JOIN geochrono.GeoEpoka NATURAL JOIN geochrono.GeoOkres NATURAL JOIN geochrono.GeoEra NATURAL JOIN geochrono.GeoEon
QUERY PLAN
Finalize Aggregate (cost=13912.92..13912.93 rows=1 width=8)
-> Gather (cost=13912.70..13912.91 rows=2 width=8)
Workers Planned: 2
-> Partial Aggregate (cost=12912.70..12912.71 rows=1 width=8)
-> Hash Join (cost=7.61..12511.66 rows=160417 width=0)
Hash Cond: (geoepoka.id_okres = geookres.id_okres)
-> Hash Join (cost=4.23..11250.46 rows=160417 width=4)
Hash Cond: (geopietro.id_epoka = geoepoka.id_epoka)
-> Hash Join (cost=2.73..10746.75 rows=160417 width=4)
Hash Cond: (mod(milion.liczba, 68) = geopietro.id_pietro)
-> Parallel Seq Scan on milion (cost=0.00..9572.67 rows=416667 width=4)
-> Hash (cost=1.77..1.77 rows=77 width=8)
-> Seq Scan on geopietro (cost=0.00..1.77 rows=77 width=8)
-> Hash (cost=1.22..1.22 rows=22 width=8)
-> Seq Scan on geoepoka (cost=0.00..1.22 rows=22 width=8)
-> Hash (cost=3.27..3.27 rows=9 width=4)
-> Hash Join (cost=2.09..3.27 rows=9 width=4)
Hash Cond: (geoera.id_eon = geoeon.id_eon)
-> Hash Join (cost=1.07..2.19 rows=9 width=8)
Hash Cond: (geookres.id_era = geoera.id_era)
-> Seq Scan on geookres (cost=0.00..1.09 rows=9 width=8)
-> Hash (cost=1.03..1.03 rows=3 width=8)
-> Seq Scan on geoera (cost=0.00..1.03 rows=3 width=8)
-> Hash (cost=1.01..1.01 rows=1 width=4)
-> Seq Scan on geoeon (cost=0.00..1.01 rows=1 width=4)

3ZG

EXPLAIN SELECT COUNT(*) FROM geochrono.Milion WHERE mod(Milion.liczba,68)= (SELECT id_pietro FROM geochrono.GeoTabela WHERE mod(Milion.liczba,68)=(id_pietro))
QUERY PLAN

Aggregate (cost=2175418.50..2175418.51 rows=1 width=8)
-> Seq Scan on milion (cost=0.00..2175406.00 rows=5000 width=0)
Filter: (mod(liczba, 68) = (SubPlan 1))
SubPlan 1
-> Seq Scan on geotabela (cost=0.00..2.16 rows=1 width=4)
Filter: (mod(milion.liczba, 68) = id_pietro)
JIT:
Functions: 10
Options: Inlining true, Optimization true, Expressions true, Deforming true

4ZG

-- 4ZG EXPLAIN SELECT COUNT(*) FROM geochrono.Milion WHERE mod(Milion.liczba,68) IN (SELECT GeoPietro.id_pietro FROM geochrono.GeoPietro NATURAL JOIN geochrono.GeoEpoka NATURAL JOIN geochrono.GeoOkres NATURAL JOIN geochrono.GeoEra NATURAL JOIN geochrono.GeoEon)
QUERY PLAN
Finalize Aggregate (cost=14093.33..14093.34 rows=1 width=8)
-> Gather (cost=14093.12..14093.33 rows=2 width=8)
Workers Planned: 2
-> Partial Aggregate (cost=13093.12..13093.13 rows=1 width=8)
-> Hash Semi Join (cost=8.47..12692.08 rows=160417 width=0)
Hash Cond: (mod(milion.liczba, 68) = geopietro.id_pietro)
-> Parallel Seq Scan on milion (cost=0.00..9572.67 rows=416667 width=4)
-> Hash (cost=7.50..7.50 rows=77 width=4)
-> Hash Join (cost=4.88..7.50 rows=77 width=4)
Hash Cond: (geoepoka.id_okres = geookres.id_okres)
-> Hash Join (cost=1.50..3.51 rows=77 width=8)
Hash Cond: (geopietro.id_epoka = geoepoka.id_epoka)
-> Seq Scan on geopietro (cost=0.00..1.77 rows=77 width=8)
-> Hash (cost=1.22..1.22 rows=22 width=8)
-> Seq Scan on geoepoka (cost=0.00..1.22 rows=22 width=8)
-> Hash (cost=3.27..3.27 rows=9 width=4)
-> Hash Join (cost=2.09..3.27 rows=9 width=4)
Hash Cond: (geoera.id_eon = geoeon.id_eon)
-> Hash Join (cost=1.07..2.19 rows=9 width=8)
Hash Cond: (geookres.id_era = geoera.id_era)
-> Seq Scan on geookres (cost=0.00..1.09 rows=9 width=8)
-> Hash (cost=1.03..1.03 rows=3 width=8)
-> Seq Scan on geoera (cost=0.00..1.03 rows=3 width=8)
-> Hash (cost=1.01..1.01 rows=1 width=4)
-> Seq Scan on geoeon (cost=0.00..1.01 rows=1 width=4)

1ZL

EXPLAIN SELECT COUNT(*) FROM geochrono.Milion INNER JOIN geochrono.GeoTabela ON (mod(Milion.liczba,68)= (GeoTabela.id_pietro))
QUERY PLAN
Finalize Aggregate (cost=12148.01..12148.02 rows=1 width=8)
-> Gather (cost=12147.79..12148.00 rows=2 width=8)
Workers Planned: 2
-> Partial Aggregate (cost=11147.79..11147.80 rows=1 width=8)
-> Hash Join (cost=2.73..10746.75 rows=160417 width=0)
Hash Cond: (mod(milion.liczba, 68) = geotabela.id_pietro)
-> Parallel Seq Scan on milion (cost=0.00..9572.67 rows=416667 width=4)
-> Hash (cost=1.77..1.77 rows=77 width=4)
-> Seq Scan on geotabela (cost=0.00..1.77 rows=77 width=4)

2ZL

EXPLAIN SELECT COUNT(*) FROM geochrono.Milion INNER JOIN geochrono.GeoPietro ON (mod(Milion.liczba,68)=geochrono.GeoPietro.id_pietro) NATURAL JOIN geochrono.GeoEpoka NATURAL JOIN geochrono.GeoOkres NATURAL JOIN geochrono.GeoEra NATURAL JOIN geochrono.GeoEon
QUERY PLAN
Finalize Aggregate (cost=13912.92..13912.93 rows=1 width=8)
-> Gather (cost=13912.70..13912.91 rows=2 width=8)
Workers Planned: 2
-> Partial Aggregate (cost=12912.70..12912.71 rows=1 width=8)
-> Hash Join (cost=7.61..12511.66 rows=160417 width=0)
Hash Cond: (geoepoka.id_okres = geookres.id_okres)
-> Hash Join (cost=4.23..11250.46 rows=160417 width=4)
Hash Cond: (geopietro.id_epoka = geoepoka.id_epoka)
-> Hash Join (cost=2.73..10746.75 rows=160417 width=4)
Hash Cond: (mod(milion.liczba, 68) = geopietro.id_pietro)
-> Parallel Seq Scan on milion (cost=0.00..9572.67 rows=416667 width=4)
-> Hash (cost=1.77..1.77 rows=77 width=8)
-> Seq Scan on geopietro (cost=0.00..1.77 rows=77 width=8)
-> Hash (cost=1.22..1.22 rows=22 width=8)
-> Seq Scan on geoepoka (cost=0.00..1.22 rows=22 width=8)
-> Hash (cost=3.27..3.27 rows=9 width=4)
-> Hash Join (cost=2.09..3.27 rows=9 width=4)
Hash Cond: (geoera.id_eon = geoeon.id_eon)
-> Hash Join (cost=1.07..2.19 rows=9 width=8)
Hash Cond: (geookres.id_era = geoera.id_era)
-> Seq Scan on geookres (cost=0.00..1.09 rows=9 width=8)
-> Hash (cost=1.03..1.03 rows=3 width=8)
-> Seq Scan on geoera (cost=0.00..1.03 rows=3 width=8)
-> Hash (cost=1.01..1.01 rows=1 width=4)
-> Seq Scan on geoeon (cost=0.00..1.01 rows=1 width=4)

3ZG

EXPLAIN SELECT COUNT(*) FROM geochrono.Milion WHERE mod(Milion.liczba,68)= (SELECT id_pietro FROM geochrono.GeoTabela WHERE mod(Milion.liczba,68)=(id_pietro))
QUERY PLAN

Aggregate (cost=2175418.50..2175418.51 rows=1 width=8)
-> Seq Scan on milion (cost=0.00..2175406.00 rows=5000 width=0)
Filter: (mod(liczba, 68) = (SubPlan 1))
SubPlan 1
-> Seq Scan on geotabela (cost=0.00..2.16 rows=1 width=4)
Filter: (mod(milion.liczba, 68) = id_pietro)
JIT:
Functions: 10
Options: Inlining true, Optimization true, Expressions true, Deforming true

4ZG

EXPLAIN SELECT COUNT(*) FROM geochrono.Milion WHERE mod(Milion.liczba,68) IN (SELECT GeoPietro.id_pietro FROM geochrono.GeoPietro NATURAL JOIN geochrono.GeoEpoka NATURAL JOIN geochrono.GeoOkres NATURAL JOIN geochrono.GeoEra NATURAL JOIN geochrono.GeoEon)
QUERY PLAN
Finalize Aggregate (cost=14093.33..14093.34 rows=1 width=8)
-> Gather (cost=14093.12..14093.33 rows=2 width=8)
Workers Planned: 2
-> Partial Aggregate (cost=13093.12..13093.13 rows=1 width=8)
-> Hash Semi Join (cost=8.47..12692.08 rows=160417 width=0)
Hash Cond: (mod(milion.liczba, 68) = geopietro.id_pietro)
-> Parallel Seq Scan on milion (cost=0.00..9572.67 rows=416667 width=4)
-> Hash (cost=7.50..7.50 rows=77 width=4)
-> Hash Join (cost=4.88..7.50 rows=77 width=4)
Hash Cond: (geoepoka.id_okres = geookres.id_okres)
-> Hash Join (cost=1.50..3.51 rows=77 width=8)
Hash Cond: (geopietro.id_epoka = geoepoka.id_epoka)
-> Seq Scan on geopietro (cost=0.00..1.77 rows=77 width=8)
-> Hash (cost=1.22..1.22 rows=22 width=8)
-> Seq Scan on geoepoka (cost=0.00..1.22 rows=22 width=8)
-> Hash (cost=3.27..3.27 rows=9 width=4)
-> Hash Join (cost=2.09..3.27 rows=9 width=4)
Hash Cond: (geoera.id_eon = geoeon.id_eon)
-> Hash Join (cost=1.07..2.19 rows=9 width=8)
Hash Cond: (geookres.id_era = geoera.id_era)
-> Seq Scan on geookres (cost=0.00..1.09 rows=9 width=8)
-> Hash (cost=1.03..1.03 rows=3 width=8)
-> Seq Scan on geoera (cost=0.00..1.03 rows=3 width=8)
-> Hash (cost=1.01..1.01 rows=1 width=4)
-> Seq Scan on geoeon (cost=0.00..1.01 rows=1 width=4)