Wydajność[1]

Dagmara Pasiak

Wprowadzenie

Przeprowadzone poniżej doświadczenie ma na celu porównanie wydajności. Przedstawiony został wpływ rodzaju złączeń i zagnieżdżeń dla schematów znormalizowanych i zdenormalizowanych dla różnych systemów zarządzania bazami danych. Na potrzeby testów została stworzona baza danych licząca ponad milion rekordów, więc dzięki jej rozmiarom różnica czasu wykonania zapytań będzie bardziej widoczna. Na przedmiot badań wybrano tabelę geochronologiczną i tabelę Milion wypełnioną liczbami od 0 do 999 999.

Czas wykonywania zapytania będzie zależeć od: rodzaju złączenia tabel (wewnętrzne/ zapytania zagnieżdżone), tego czy indeksy zostały nałożone, czy tabela jest w wersji znormalizowanej czy z zdenormalizowanej oraz systemu zarządzania bazami danych (PostgreSQL/MySQL).

Parametry komputera

CPU: AMD Ryzen 5 2500U with Radeon Vega Mobile Gfx 2.00 GHz

RAM: 20,0 GB (dostępne 17,9 GB)

Dysk: SSD M.2

S.O.: Windows 10

Wersje systemów zarządzania bazami danych

PostgreSQL, wersja 13.2

MySQL, wersja Community 8.0.25.0

Rodzaje zapytań

1 ZL - Zapytanie łączące syntetyczną tabelę Milion z tabelą geochronologiczną (zdenormalizowaną) przez złączenie warunkowe z operacją modulo

2 ZL - Zapytanie łączące syntetyczną tabelę Milion z tabelą geochronologiczną (znormalizowaną) przez złączenie pięciu tabel

3 ZG - Zapytanie łączące syntetyczną tabelę Milion z tabelą geochronologiczną (zdenormalizowaną) przez zagnieżdżenie skorelowane

4 ZG - Zapytanie łączące syntetyczną tabelę Milion z tabelą geochronologiczną (znormalizowaną) przez zagnieżdżenie skorelowane oraz złączenie tabel jednostek geologicznych dla zapytania wewnętrznego

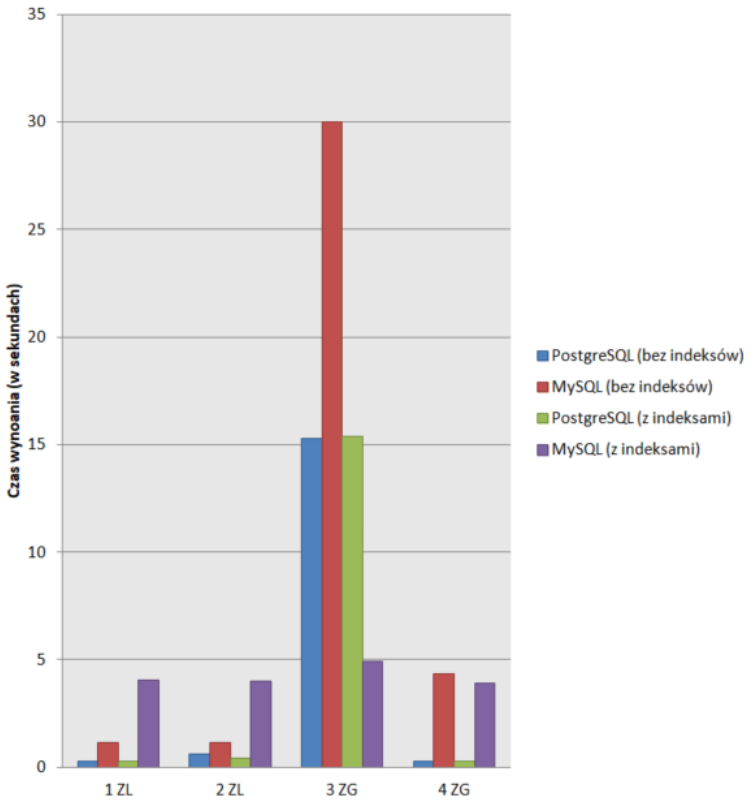
[1] Praca na podstawie artykułu „Wydajność złączeń i zagnieżdżeń dla schematów znormalizowanych i zdeznormalizowanych" Łukasza Jajeśnicy i Adama Piórkowskiego

Wyniki testów

Testy zostały przeprowadzone 10 krotnie dla każdego z zapytań w poszczególnej konfiguracji. Poniżej przedstawione zostały minimalne i uśrednione wyniki testów.

Tabela 1. Czas wykonania zapytań w sekundach

	1 ZL		2 ZL		3 ZG		4 ZG	
BEZ INDEKSÓW	MIN	ŚR	MIN	ŚR	MIN	ŚR	MIN	ŚR
PostgreSQL	0,277	0,2898	0,6	0,613	15,103	15,295	0,271	0,294
MySQL	1,094	1,131	1,125	1,169	30	30,012	4,078	4,35
Z INDEKSAMI								
PostgreSQL	0,281	0,291	0,414	0,426	15,175	15,362	0,276	0,288
MySQL	3,828	4,031	3,859	4,028	4,797	4,906	3,875	3,928



Wykres 1 Średni czas wykonania zapytań dla poszczególnych zapytań

Query plan

1 ZL

Data Output

	QUERY PLAN text
1	Finalize Aggregate (cost=14664.16..14664.17 rows=1 width=8)
2	-> Gather (cost=14663.95..14664.16 rows=2 width=8)
3	Workers Planned: 2
4	-> Partial Aggregate (cost=13663.95..13663.96 rows=1 width=8)
5	-> Hash Join (cost=2.73..13262.90 rows=160417 width=0)
6	Hash Cond: (mod(milion.liczba, 68) = geotabela.id_pietro)
7	-> Parallel Seq Scan on milion (cost=0.00..9572.67 rows=416667 width=4)
8	-> Hash (cost=1.77..1.77 rows=77 width=4)
9	-> Seq Scan on geotabela (cost=0.00..1.77 rows=77 width=4)

3 ZG

	QUERY PLAN text
1	Aggregate (cost=2175418.50..2175418.51 rows=1 width=8)
2	-> Seq Scan on milion (cost=0.00..2175406.00 rows=5000 width=0)
3	Filter: (mod(liczba, 68) = (SubPlan 1))
4	SubPlan 1
5	-> Seq Scan on geotabela (cost=0.00..2.16 rows=1 width=4)
6	Filter: (mod(milion.liczba, 68) = id_pietro)

2 ZL

	QUERY PLAN text
1	Finalize Aggregate (cost=13912.92..13912.93 rows=1 width=8)
2	-> Gather (cost=13912.70..13912.91 rows=2 width=8)
3	Workers Planned: 2
4	-> Partial Aggregate (cost=12912.70..12912.71 rows=1 width=8)
5	-> Hash Join (cost=7.61..12511.66 rows=160417 width=0)
6	Hash Cond: (geoepoka.id_okres = geookres.id_okres)
7	-> Hash Join (cost=4.23..11250.46 rows=160417 width=4)
8	Hash Cond: (geopietro.id_epoka = geoepoka.id_epoka)
9	-> Hash Join (cost=2.73..10746.75 rows=160417 width=4)
10	Hash Cond: (mod(milion.liczba, 68) = geopietro.id_pietro)
11	-> Parallel Seq Scan on milion (cost=0.00..9572.67 rows=416667 width=4)
12	-> Hash (cost=1.77..1.77 rows=77 width=8)
13	-> Seq Scan on geopietro (cost=0.00..1.77 rows=77 width=8)
14	-> Hash (cost=1.22..1.22 rows=22 width=8)
15	-> Seq Scan on geoepoka (cost=0.00..1.22 rows=22 width=8)
16	-> Hash (cost=3.27..3.27 rows=9 width=4)
17	-> Hash Join (cost=2.09..3.27 rows=9 width=4)
18	Hash Cond: (geoera.id_eon = geoeon.id_eon)
19	-> Hash Join (cost=1.07..2.19 rows=9 width=8)
20	Hash Cond: (geookres.id_era = geoera.id_era)
21	-> Seq Scan on geookres (cost=0.00..1.09 rows=9 width=8)
22	-> Hash (cost=1.03..1.03 rows=3 width=8)
23	-> Seq Scan on geoera (cost=0.00..1.03 rows=3 width=8)
24	-> Hash (cost=1.01..1.01 rows=1 width=4)
25	-> Seq Scan on geoeon (cost=0.00..1.01 rows=1 width=4)

4 ZG

	QUERY PLAN text
1	Finalize Aggregate (cost=12668.93..12668.94 rows=1 width=8)
2	InitPlan 1 (returns $0)
3	-> Hash Join (cost=4.88..7.50 rows=77 width=4)
4	Hash Cond: (geoepoka.id_okres = geookres.id_okres)
5	-> Hash Join (cost=1.50..3.51 rows=77 width=8)
6	Hash Cond: (geopietro.id_epoka = geoepoka.id_epoka)
7	-> Seq Scan on geopietro (cost=0.00..1.77 rows=77 width=8)
8	-> Hash (cost=1.22..1.22 rows=22 width=8)
9	-> Seq Scan on geoepoka (cost=0.00..1.22 rows=22 width=8)
10	-> Hash (cost=3.27..3.27 rows=9 width=4)
11	-> Hash Join (cost=2.09..3.27 rows=9 width=4)
12	Hash Cond: (geoera.id_eon = geoeon.id_eon)
13	-> Hash Join (cost=1.07..2.19 rows=9 width=8)
14	Hash Cond: (geookres.id_era = geoera.id_era)
15	-> Seq Scan on geookres (cost=0.00..1.09 rows=9 width=8)
16	-> Hash (cost=1.03..1.03 rows=3 width=8)
17	-> Seq Scan on geoera (cost=0.00..1.03 rows=3 width=8)
18	-> Hash (cost=1.01..1.01 rows=1 width=4)
19	-> Seq Scan on geoeon (cost=0.00..1.01 rows=1 width=4)
20	-> Gather (cost=12661.21..12661.42 rows=2 width=8)
21	Workers Planned: 2
22	Params Evaluated: $0
23	-> Partial Aggregate (cost=11661.21..11661.22 rows=1 width=8)
24	-> Parallel Seq Scan on milion (cost=0.00..11656.00 rows=2083 width=0)
25	Filter: (mod(liczba, 68) = $0)

Wnioski

Analizując czasy wykonania zapytań można zauważyć kilka zależności. Użycie indeksów w PostgreSQL praktycznie nie zmieniło czasu wykonania zapytań, za to w MySQL sprawiło, że niezależnie od rodzaju zapytania, wyniki miały bardzo podobną wartość. W drugim przypadku oznacza to zwiększenie czasu dla trzech zapytań i znaczne zmniejszenie dla zapytania 3ZG. Lepszą wydajnością charakteryzuje się też postać zdenormalizowana, choć przy optymalizacji należy pamiętać o tym by nie stracić przy niej na przejrzystości bazy, którą zapewnia trzecia postać normalna. MySQL okazał się wydajniejszy od PostgreSQL jedynie w przypadku zapytania 3ZG po założeniu indeksów.

Przedstawione wyniki wskazują na to, że proces optymalizacji zależy od wielu czynników i wykonanie jej w efektywny sposób wymaga znajomości działania danego systemu zarządzania bazami danych oraz wiedzy teoretycznej na ten temat.