AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE

Sprawozdanie z ćwiczenia IX

Bazy danych

Piotr Krajewski

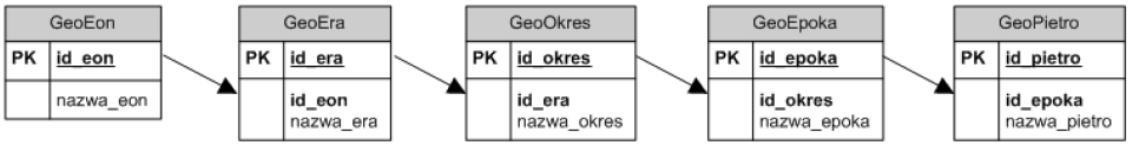
24.05.2023

I Cel i problematyka ćwiczenia.

Ćwiczenie ma na celu sprawdzenie i porównanie wydajności złączeń i zagnieżdżeń skorelowanych dla PostgreSQL oraz MySQL. Do wykonania testów użyto wymiaru czasu na tabeli jednostek geologicznych czyli szablonowej konstrukcji baz danych geologicznych. Tabela została stworzona specjalnie w celach tego zadania w formie znormalizowanej jak i zdenormalizowanej.

II Tabela geochronologiczna.

Baza składa się z tabeli geochronologicznej, gdzie główną jednostką jest eon.
Coraz mniejsze jednostki wchodzące po kolei w siebie to:
eon -> era -> okres -> epoka -> piętro
Tabela (baza) została stworzona w dwóch wariantach – znormalizowanym i zdenormalizowanym, gdzie wariancie znormalizowanym stworzonych zostało pięć tabel, każda w następstwie takim, w jakim występują jednostki geologicznego czasu.



Rys 1. Baza z tabelą znormalizowaną.

Schemat zdenormalizowany to tabela z wszystkimi jednostkami czasu geologicznego na raz i z przynależnościami przejściowymi.

Rys 2. Baza z tabelą zdenormalizowaną.

Finalnie tabela zawierała 77 rekordów, ponieważ wyliczonych zostało 77 pięter geologicznych, poczynając od lochkowa a kończąc na aktualnie trwającym megalaju.

III Tabela pomocnicza „Milion”.

Do zastosowania zapytań testujących czas wykonywania operacji potrzebna była tabela Milion zawierająca liczby od 0 do 999999. Stworzona została za pomocą pomocniczej tabeli dziesięć, symulującej układ dziesiętny. Sześć kolumn tabeli milion zostało wypełnione po kolei każdą cyfrą z tabeli Dziesięć.

```
--stworzenie tabel milion i dziesiec
CREATE TABLE dziesiec(cyfra INT,bit INT);
CREATE TABLE milion(liczba INT,cyfra INT, bit INT);
--wypelnianie dziesiatki
INSERT INTO dziesiec VALUES (0,1);
INSERT INTO dziesiec VALUES (1,1);
INSERT INTO dziesiec VALUES (2,1);
INSERT INTO dziesiec VALUES (3,1);
INSERT INTO dziesiec VALUES (4,1);
INSERT INTO dziesiec VALUES (5,1);
INSERT INTO dziesiec VALUES (6,1);
INSERT INTO dziesiec VALUES (7,1);
INSERT INTO dziesiec VALUES (8,1);
INSERT INTO dziesiec VALUES (9,1);
select * from dziesiec;

--wypelnianie miliona
INSERT INTO milion
select a1.cyfra+10*a2.cyfra+100*a3.cyfra+1000*a4.cyfra
+10000*a5.cyfra+10000*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;
```

Rys 3. Tworzenie i wypełnianie „Miliona” w PostgresSQL.

IV Specyfikacja komputera.

Procesor	AMD Ryzen 7 5800H with Radeon Graphics	3.20 GHz
Zainstalowana pamięć RAM	16,0 GB (dostępne: 15,9 GB)	

Rys 4. Pamięć RAM oraz CPU.

System Windows 11
PostgreSQL w wersji 15.3-1
MySQL w wersji 8.0
Dysk SSD Samsung 980 1TB

V Zastosowane zapytania.

1ZL

Złączenie tablicy „Milion” z tablicą geochronologiczą zdenormalizowaną wraz z dodaniem opcji modulo, która dopasowuje zakresy łączonych kolumn.

```
--ZAPYTANIE 1
SELECT COUNT(*) FROM milion INNER JOIN geol.chrono ON
(mod(milion.liczba,77)=(geol.chrono.id_pietro));
```

2ZL

Złączenie tablicy „Milion” z tablicą geochronologiczną znormalizowaną.

```
--ZAPYTANIE 2
SELECT COUNT(*) FROM milion INNER JOIN geol.pietro ON
(mod(milion.liczba,77)=geol.pietro.id_pietro) NATURAL JOIN geol.epoka NATURAL JOIN
```

3ZG

Złączenie tablicy „Milion” z tablicą zdenormalizowaną poprzez zagnieżdżenie skorelowane.

```
--ZAPYTANIE 3
SELECT COUNT(*) FROM milion WHERE mod(milion.liczba,77)=
(SELECT id_pietro FROM geol.chrono WHERE mod(milion.liczba,77)=(id_pietro));
```

4ZG

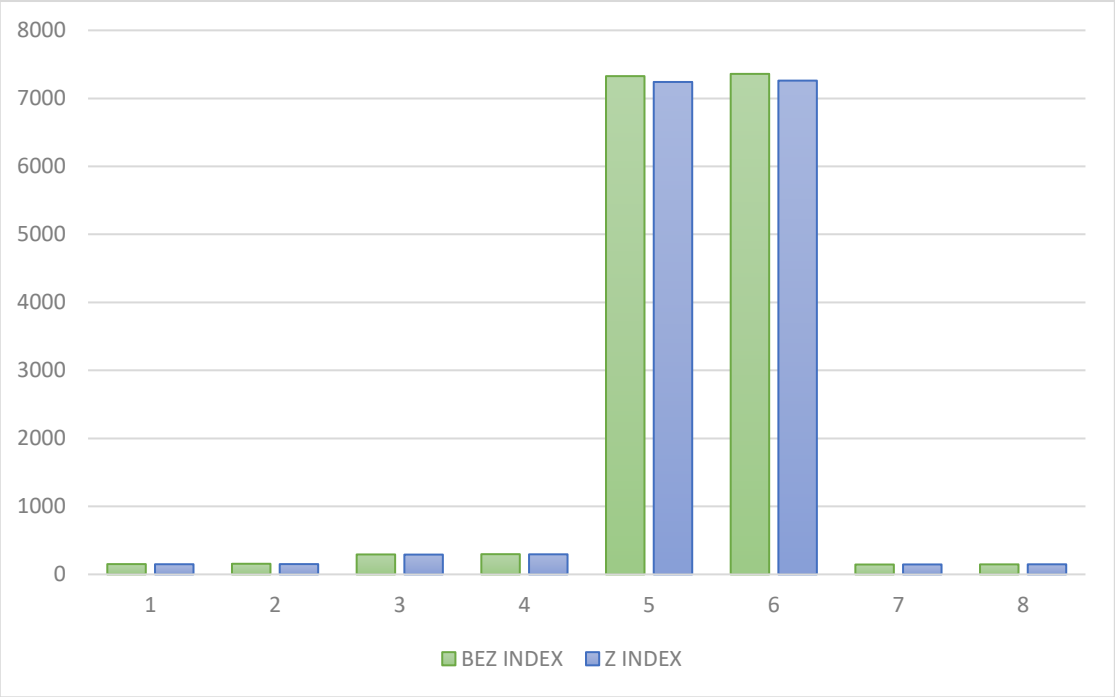
Złączenie podobnie jak w 2ZL, z tym że stosowane jest zagnieżdżenie skorelowane.

```
--ZAPYTANIE 4
SELECT COUNT(*) FROM milion WHERE mod(milion.liczba,77) in
(SELECT geol.pietro.id_pietro FROM geol.pietro NATURAL JOIN geol.epoka NATURAL JOIN geol.okres NATURAL JOIN geol.era NATURAL JOIN geol.eon);
```

VI Wyniki.

Czas w ms	1ZL		2ZL		3ZG		4ZG	
	Min	Śr	Min	Śr	Min	Śr	Min	Śr
	Bez indeksów							
PostgreSQL	151	156	293	297	7327	7360	145	147
	Z indeksami							
PostgreSQL	149	151	291	295	7241	7261	146	148

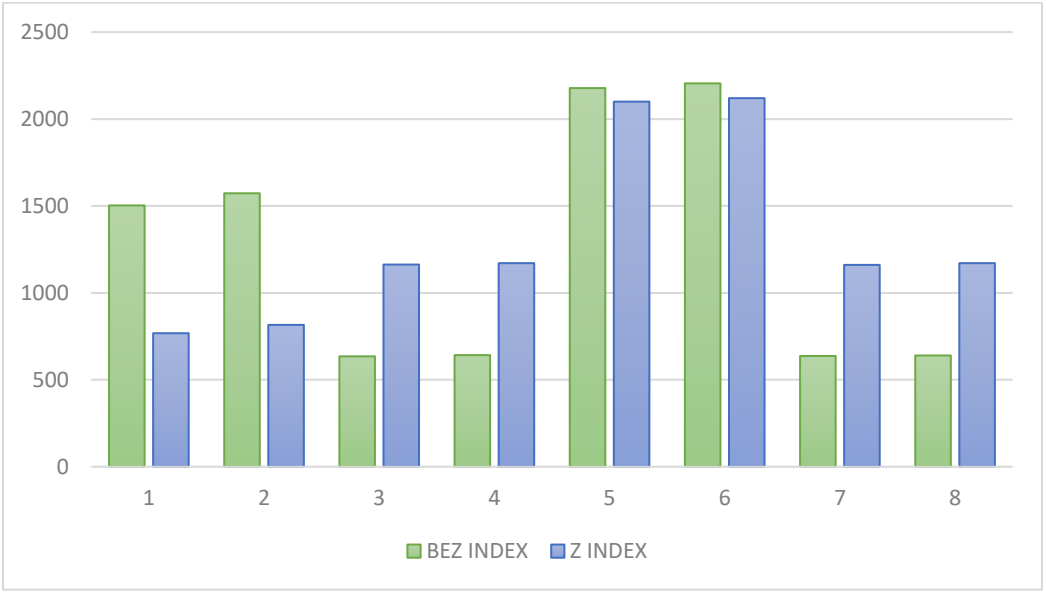
Tab 1. Wyniki testów dla PostgreSQL.



Wykres 1. Wyniki w poniższej kolejności dla PostgreSQL.
1-1ZL MIN, 2-1ZL ŚR, 3-2ZL MIN, 4-2ZL ŚR, 5-3ZG MIN, 6-3ZG ŚR, 7-4ZG MIN, 8-4ZG ŚR

Czas w ms	1ZL		2ZL		3ZG		4ZG	
	Min	Śr	Min	Śr	Min	Śr	Min	Śr
	Bez indeksów							
MySQL	1503	1573	635	642	2178	2205	637	640
	Z indeksami							
MySQL	768	816	1163	1171	2100	2120	1161	1171

Tab 2. Wyniki testów dla MySQL.



Wykres 1. Wyniki w poniższej kolejności dla MySQL.
1-1ZL MIN, 2-1ZL ŚR, 3-2ZL MIN, 4-2ZL ŚR, 5-3ZG MIN, 6-3ZG ŚR, 7-4ZG MIN, 8-4ZG ŚR

VII. Wnioski.

Po przejrzeniu wyników można wysnuć wnioski, że indeksowanie wpływa tylko nieznacznie na poprawę lub pogorszenie czasu operacji w PostgreSQL. Najbardziej widoczne jest to w wypadku zagadnienia trzeciego, ponieważ również czas potrzebny na wykonanie operacji jest dużo wyższy. W wypadku MySQL sprawa ma się zgoła inaczej, gdzie w przypadkach 1ZL czas z indexem skraca się o około połowę, natomiast dla 2ZL i 4ZG jest odwrotnie. Sytuacja z 3ZG wygląda podobnie jak w wypadku Postgre. Zauważyć również można jak duże znaczenie ma normalizacja bazy danych tak jak w przypadku 3ZG dla PostgreSQL, gdzie czas to około 7,36 sekundy, będące wielokrotnie większą wartością niż około 0,15 sekundy. W wypadku MySQL wyraźnie zaobserwować można o wiele dłuższe czasy niż w przypadku PostgreSQL, za wyjątkiem trzeciego zadania, które wykazało się jedynie około trzykrotnie większym czasem niż inne odpalane w MySQL. Można wywnioskować że tabela zdenormalizowana nie stanowi aż tak dużego problemu jak w wypadku „konkurencji”.