**WYDAJNOŚĆ ZŁĄCZEŃ I ZAGNIEŻDŻEŃ DLA SCHEMATÓW ZNORMALIZOWANYCH I ZDENORMALIZOWANYCH**

**Karol Pastuszka**

**04.06.2021**

## 1) Cel ćwiczenia oraz jego wykonanie.

Utworzyłem bazę danych zawierającą informacje na temat tabeli stratygraficznej oraz jednostek z których się ona składa. Wykonałem to w celu sprawdzenia wydajności dla zapytań oraz zagnieżdżeń dla tabeli znormalizowanych i nieznormalizowanych oraz zbadania wpływu indeksowania jako mechanizmu do optymalizacji baz danych SQL. Utworzyłem tabelę zawierające dane geochronologiczną w postaci znormalizowanej oraz zdenormalizowanej oraz tabelę Milion która została wypełniona liczbami od 0 do 999 999. Zapytania łączyły zawsze jedną z tabel geochronologicznych z tabelą Milion.

**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,88)=(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,88)=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,88)
 = (SELECT id_pietro FROM GeoTabela WHERE mod(Milion.liczba,88)=(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 tabel poszczególnych jednostek**.

```
SELECT COUNT(*) FROM Milion
WHERE mod(Milion.liczba,88) in (SELECT GeoPietro.id_pietro FROM GeoPietro
NATURAL JOIN GeoEpoka NATURAL JOIN GeoOkres NATURAL JOIN GeoEra NATURAL JOIN GeoEon);
```

Następnie przeprowadziłem testy wydajności, na darmowym oprogramowaniu **MySQL oraz SQLServer.**

W pierwszej kolejności wykonałem pomiary szybkości wykonania zapytań, które nie miały nałożonych indeksów na kolumny, następnie powtórzyłem to dla zapytań z indeksami nałożonymi na kolumny. Wykonałem 15 powtórzeń dla każdego zapytania w obu wariantach.

## 2) Konfiguracja sprzętowa komputerów, na którym zostały wykonane testy.

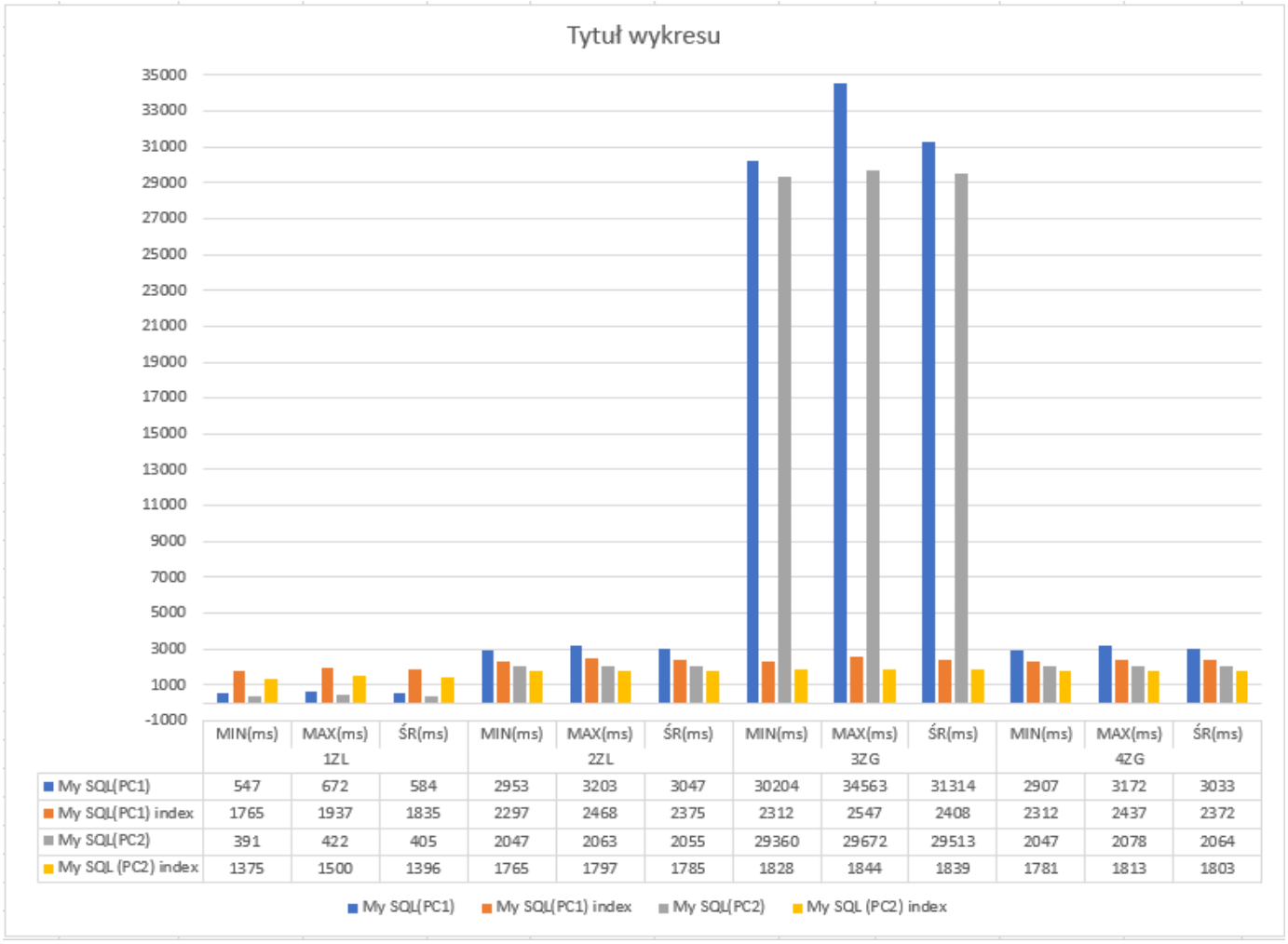
Komputer nr1.

- CPU: Intel(R) Core(TM) i7-7700 CPU @ 3.6GHz 4.1Ghz,
- RAM: Pamięć DDR4 2 x 8.00GB (2133/3200 MHZ)
- SSD Kingston SA2000M8500G, 500GB,
- S.O: Windows 10 Pro x86,
- MySQL, wersja Community Server 8.0.24,
- SQL Server 15.0.2000.5

Komputer nr2.

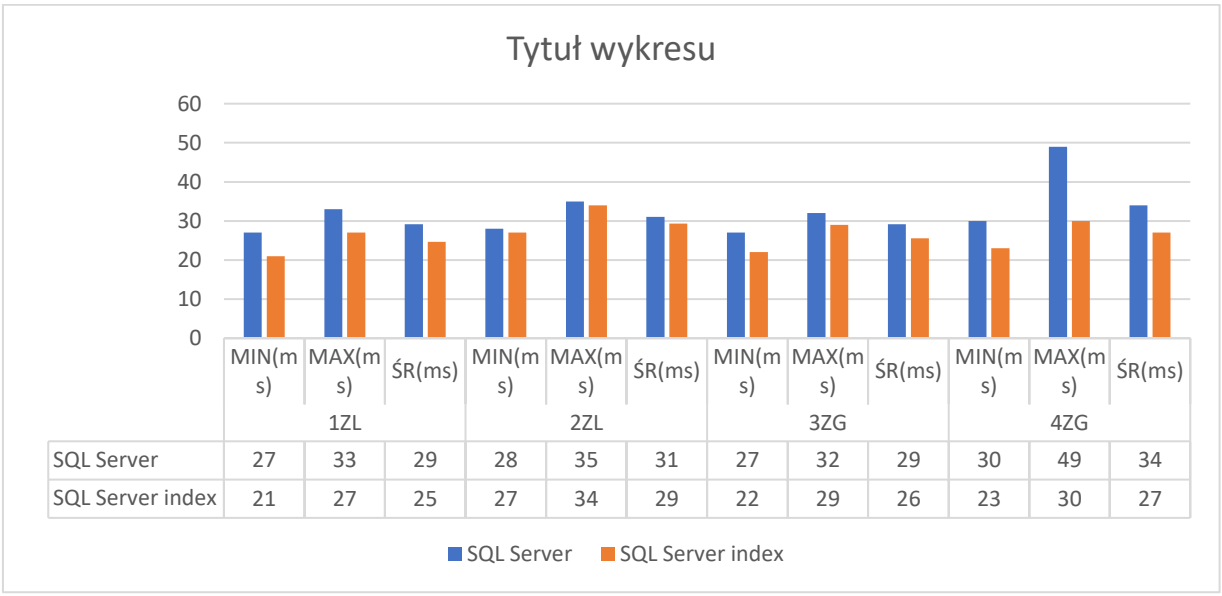
- CPU: Intel(R) Core(TM) i9-10900K CPU @ 3.7GHz 5.1Ghz,
- RAM: Pamięć DDR4 2 x 16.00GB (3600 MHZ)
- SSD Dysk SSD Samsung 970 EVO Plus 1 TB M.2 2280 PCI-E x4 Gen3 NVMe
- S.O: Windows 10 Pro x86,
- MySQL, wersja Community Server 8.0.24,

## 3) Wyniki




| | MIN(ms) | MAX(ms) | ŚR(ms) | MIN(ms) | MAX(ms) | ŚR(ms) | MIN(ms) | MAX(ms) | ŚR(ms) | MIN(ms) | MAX(ms) | ŚR(ms) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 1ZL | | | 2ZL | | | 3ZG | | | 4ZG | | |
| My SQL(PC1) | 547 | 672 | 584 | 2953 | 3203 | 3047 | 30204 | 34563 | 31314 | 2907 | 3172 | 3033 |
| My SQL(PC1) index | 1765 | 1937 | 1835 | 2297 | 2468 | 2375 | 2312 | 2547 | 2408 | 2312 | 2437 | 2372 |
| My SQL(PC2) | 391 | 422 | 405 | 2047 | 2063 | 2055 | 29360 | 29672 | 29513 | 2047 | 2078 | 2064 |
| My SQL (PC2) index | 1375 | 1500 | 1396 | 1765 | 1797 | 1785 | 1828 | 1844 | 1839 | 1781 | 1813 | 1803 |






| | MIN(ms) | MAX(ms) | ŚR(ms) | MIN(ms) | MAX(ms) | ŚR(ms) | MIN(ms) | MAX(ms) | ŚR(ms) | MIN(ms) | MAX(ms) | ŚR(ms) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 1ZL | | | 2ZL | | | 3ZG | | | 4ZG | | |
| SQL Server | 27 | 33 | 29 | 28 | 35 | 31 | 27 | 32 | 29 | 30 | 49 | 34 |
| SQL Server index | 21 | 27 | 25 | 27 | 34 | 29 | 22 | 29 | 26 | 23 | 30 | 27 |

## 4) Wnioski

Analizując otrzymane wyniki możemy dojść do wniosku, że nałożenie indeksów przyczynia się do zwiększenie prędkości oraz zmniejszenia czasy wykonywania się kwerend.

- Zagnieżdżenia skorelowane są wolniejsze w wykonaniu niż złączenia w przypadku używania MySQL
- Jedynym wypadkiem, w którym postać zdenormalizowana jest wolniejsza jest zapytanie 1 dla MySQL, średni czas wykonania po indeksowaniu znacznie wzrósł. (ok 330%).
- Dla zapytania 3 (MySQL) występuje olbrzymi spadek o ok 1450 %.
- W SQL Serwerze zmiany te nie są aż tak drastyczne i widoczny jest spadek średniego czasu wraz z wprowadzeniem indeksowania.
- SQL Serwer poradził sobie zdecydowanie lepiej z tym zadaniem niż MySQL.

Analiza została wykonana w MySQL dla dwóch różnych komputerów w celu potwierdzenia swoich spostrzeżeń oraz sprawdzenia poprawności wyliczeń. Natomiast w SQL Server tylko dla jednego, gdyż wyniki były do siebie zbliżone.