Sprawozdanie z ćwiczenia 9

*Krzysztof Woźniak*

## 1.Wprowadzenie

Ćwiczenie zostało wykonane na podstawie artykułu Łukasza Jaješnicy i Adama Piórkowskiego ,, Sprawozdanie Analiza wydajności złączeń i zagnieżdżeń dla schematów znormalizowanych i zdenormalizowanych, w różnych bazach danych”

Celem zadania było przeanalizowanie wyników zapytań opierających się na złączeniach wielu tabel. Wykonałem je na dwa sposoby – złączenia tabel i zapytaniach zagnieżdżonych. Do wykonania zadania posłużyłem się danymi geologicznymi – fragmencie tabeli stratygraficznej dla Eonu Fanerozoika. Dla nich przeprowadziłem testy.

## 2. Wykonane czynności

Najpierw musiałem utworzyć tabelę dla poszczególnych składowych- eonu, ery, okresu, epoki i pięter:

```
CREATE SCHEMA geo;

CREATE TABLE geo.GeoEon (
    id_eon INT PRIMARY KEY NOT NULL,
    nazwa_eon VARCHAR(32) NOT NULL

);

CREATE TABLE geo.GeoEra (
    id_era INT PRIMARY KEY NOT NULL,
    id_eon INT  NOT NULL,
    nazwa_era VARCHAR(32) NOT NULL

);
```

Następnie wprowadziłem klucze obce :

```
ALTER TABLE geo.GeoEra ADD CONSTRAINT klucz_obcy FOREIGN KEY (id_eon) REFERENCES geo.GeoEon(id_eon);
ALTER TABLE geo.GeoOkres ADD CONSTRAINT klucz_obcy_2 FOREIGN KEY (id_era) REFERENCES geo.GeoEra(id_era);
ALTER TABLE geo.GeoEpoka ADD CONSTRAINT klucz_obcy_3 FOREIGN KEY (id_okres ) REFERENCES geo.GeoOkres(id_okres );
ALTER TABLE geo.GeoPietro ADD CONSTRAINT klucz_obcy_4 FOREIGN KEY (id_epoka) REFERENCES geo.GeoEpoka(id_epoka);
```

Wypełnienie tabeli danymi:

```
INSERT INTO geo.GeoEon VALUES (1, 'Fanerozoik');


INSERT INTO geo.GeoEra VALUES (1, 1, 'Paleozoik');
INSERT INTO geo.GeoEra VALUES (2, 1, 'Mezozoik');
INSERT INTO geo.GeoEra VALUES (3, 1, 'Kenozoik');


INSERT INTO geo.GeoOkres VALUES (1, 1, 'Dewon');
INSERT INTO geo.GeoOkres VALUES (2, 1, 'Karbon');
INSERT INTO geo.GeoOkres VALUES (3, 1, 'Perm');
```

Następnie stworzyłem zdeformalizowaną formę tabeli geochronoligicznej w postaci tabeli GeoTabela, osiągniętej przy pomocy polecenia:

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

Utworzenie tabel pomocniczych Milion i Dziesiec:

```
CREATE TABLE Milion(
    liczba INT,
    cyfra INT,
    bit INT);

    CREATE TABLE Dziesiec(
        cyfra INT,
        bit INT);
```

I wypełnienie ich  wartościami:

```
    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
```

## 3. Przeprowadzenie testów:

W pierwszym etapie zapytania były przeprowadzane bez użycia indeksów na kolumnach danych( z wyjątkiem kluczy głównych poszczególnych tabel).W drugim były już nakładane indeksy na kolumny bieżące udział w złączeniu.

Zapytanie 1 (1 ZL), obejmowało złączenie syntetycznej tablicy miliona wyników z tabelą geochronologiczną w postaci zdenormalizowanej, z dodatkowym warunkiem złączenia operacją modulo, która dopasowuje zakresy wartości złączanych kolumn:

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

Celem Zapytania 2 (2 ZL), było złączenie syntetycznej tablicy miliona wyników z tabelą geochronologiczną w postaci znormalizowanej, reprezentowaną przez złączenia pięciu tabel:

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

Zapytanie 3 (3 ZG), obejmowało 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 geo.GeoTabela WHERE MOD(Milion.liczba,68)=(id_pietro));
```

Zapytanie 4 (4 ZG) było złączeniem syntetycznej tablicy wypełnionej milionem wyników z tabelą geochronologiczną w postaci znormalizowanej. Złączenie wykonywane zostało poprzez zagnieżdżenie skorelowane, a zapytanie wewnętrzne złączeniem tabel poszczególnych jednostek.

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

## 4. Konfiguracja sprzętowa i programowa

Testy omówione w artykule przeprowadziłem na komputerze o następujących parametrach:

- CPU: Intel Core i5-9300H 2.4 - 4.1 GHz,
- RAM: Pamięć DDR4 16 GB (2666 MHz),
- SSD: 512 GB
- System operacyjny Windows 10
- Systemy zarządzania bazami danych:
    - MySQL 8.0 CE
    - PostgreSQL, wersja 14.2
- Testy były przeprowadzane 5-krotnie dla każdego przypadku

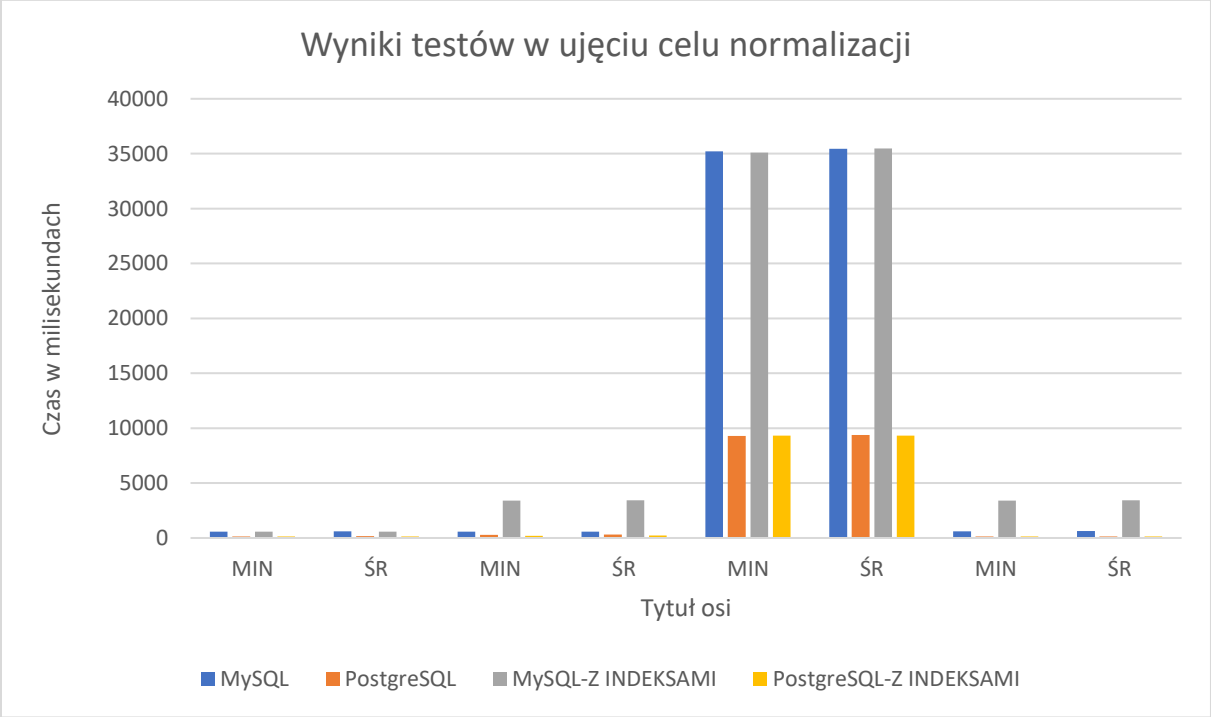
## 5. Wyniki testów i wnioski

Wyniki przeprowadzonych testów w milisekundach przedstawiłem w poniższej tabeli:

| POSTGRES | Próba 1 | Próba 2 | Próba 3 | Próba 4 | Próba 5 | min | sr |
|---|---|---|---|---|---|---|---|
| Zl1- Bez indeksów | 191 | 142 | 138 | 157 | 150 | 138 | 155,6 |
| Zl2-Bez indeksów | 304 | 313 | 294 | 310 | 290 | 290 | 302,2 |
| Zg3-Bez indeksów | 9569 | 9323 | 9410 | 9318 | 9280 | 9280 | 9380 |
| Zg4-Bez indeksów | 133 | 126 | 158 | 160 | 144 | 126 | 144,2 |
| Zl1- z indeksami | 124 | 130 | 155 | 146 | 125 | 124 | 136 |
| Zl2-z indeksami | 211 | 270 | 224 | 251 | 235 | 211 | 238,2 |
| Zg3-z indeksami | 9335 | 9312 | 9329 | 9344 | 9308 | 9308 | 9325,6 |
| Zg4-z indeksami | 149 | 134 | 131 | 136 | 137 | 131 | 137,4 |

| MySQL | Próba 1 | Próba 2 | Próba 3 | Próba 4 | Próba 5 | min | śr |
|---|---|---|---|---|---|---|---|
| Zl1- Bez indeksów | 594 | 578 | 593 | 579 | 609 | 578 | 590,6 |
| Zl2-Bez indeksów | 640 | 625 | 593 | 594 | 610 | 593 | 612,4 |
| Zg3-Bez indeksów | 35468 | 35578 | 35688 | 35250 | 35203 | 35203 | 35437,4 |
| Zg4-Bez indeksów | 625 | 609 | 610 | 641 | 625 | 609 | 622 |
| Zl1- z indeksami | 578 | 579 | 578 | 593 | 579 | 578 | 581,4 |
| Zl2-z indeksami | 3469 | 3437 | 3407 | 3406 | 3422 | 3406 | 3428,2 |
| Zg3-z indeksami | 35110 | 35250 | 35171 | 36188 | 35594 | 35110 | 35462,6 |
| Zg4-z indeksami | 3421 | 3469 | 3438 | 3391 | 3407 | 3391 | 3425,2 |

| | 1 ZL | | 2 ZL | | 3 ZG | | 4 ZG | |
|---|---|---|---|---|---|---|---|---|
| BEZ INDEKSÓW | MIN | ŚR | MIN | ŚR | MIN | ŚR | MIN | ŚR |
| MySQL | 578 | 591 | 578 | 584 | 35203 | 35437 | 609 | 622 |
| PostgreSQL | 138 | 155,6 | 290 | 302,2 | 9280 | 9380 | 126 | 144,2 |
| Z INDEKSAMI | | | | | | | | |
| MySQL | 578 | 581,4 | 3406 | 3428,2 | 35110 | 35462,6 | 3391 | 3425,2 |
| PostgreSQL | 124 | 136 | 211 | 238,2 | 9308 | 9325,6 | 131 | 137,4 |




Wnioski:

- Wprowadzanie indeksów w większości przypadków skraca czas wykonywania zapytania – jedynie zapytanie 2 i 4 dla MySQL cechuje się odwrotną zależnością
- PostgreSQL jest w większości przypadków wydajniejszy od środowiska MySQL, dla zapytania 3 różnica wykonania zapytania wyniosła ponad 25 sekund
- Najmniejsza różnica między wynikami dla danych bez indeksów, a danymi z indeksami występowała w zapytaniach 1 i 3, działających na tabeli zdenormalizowanej
- Najdłuższy czas wykonania miało zapytanie 3 wykonywane poprzez zagnieżdżenie skorelowane, kilkukrotnie wyższe od pozostałych
- W obu programach najkrótszy czas miało zapytanie pierwsze

- W PostgreSQL wyniki przed i po indeksacji są do siebie zbliżone, podczas gdy w MySQL jest zauważalna różnica

## 6. Podsumowanie

Wprowadzenie indeksacji na kolumny tabeli w większości przypadków skraca czas wykonywania zapytania. Zdarzają się jednak przypadki, że mogą wydłużyć czas jak w przypadku MySQL dla 2 i 4 zapytania. Patrząc na czas wykonywania zapytań możemy zauważyć większą wydajność PostgreSQL nad MySQL. Wprowadzenie indeksów na dane w tabelach w Postgresie nie wpłynęło znacząco na czas wykonania zapytania w porównaniu z MySQL, gdzie widzimy znaczną zmianę w czasie testowania zapytania.