Emilia Brandys

Nr albumu: 409963, Geoinformatyka

Ćwiczenie nr. 9

**TESTOWANIE WYDAJNOŚCI ZŁĄCZEŃ I ZAGNIEŻDŻEŃ DLA SCHEMATÓW ZNORMALIZOWANYCH I ZDENORMALIZOWANYCH**

## Wstęp

Celem ćwiczenia nr. 9 jest szybkości działania dwóch serwerów SQL: PostgreSQL oraz MySQL. Czerpiąc z badania dotyczącego relacyjnych baz danych pt.: „Wydajność złączeń i zagnieżdżeń dla schematów znormalizowanych i zdenormalizowanych” zrealizowanego przez Łukasza Jajeśnicę oraz Adama Piórkowskiego wykonano te same przedstawione tam kroki, aby samodzielnie zbadać wydajność złączeń oraz zapytań na własnym sprzęcie.

## Stworzenie tabeli geochronologicznej

W języku SQL stworzono nową bazę danych, a w niej skonstruowano skróconą tabelę geochronologiczną (Tabela 1) aby zaprezentować:

1. Schemat znormalizowany (schemat płatka śniegu)

Stworzono go w następujący sposób:

Utworzono 5 znormalizowanych tabel: GeoEon, GeoEra, GeoOkres, GeoEpoka, GeoPietro oraz przedstawiono na schemacie (Rysunek 1). Tabele wypełniono nazwami oraz indeksami eonu, er, okresów, epok i pięter oraz ustawiono klucze obce.

Obraz zawierający tekst, zrzut ekranu, diagram, Czcionka

Opis wygenerowany automatycznie

Rysunek

1. Schemat zdenormalizowany (schemat gwiazdy)

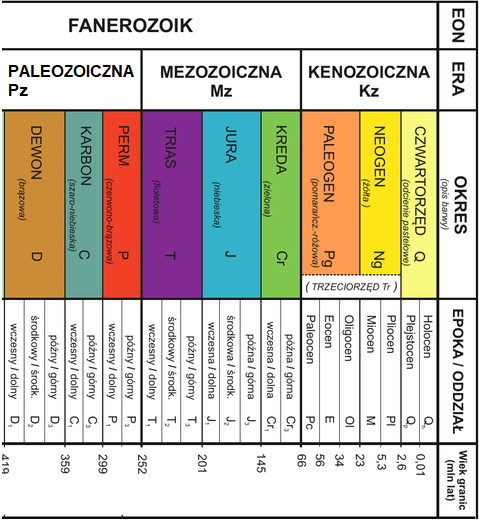
Utworzono tabelę GeoTabela, która jest złączeniem naturalnym wszystkich powyższych tabel za pomocą zapytania:

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

Obraz zawierający tekst, zrzut ekranu, Czcionka, numer

Opis wygenerowany automatycznie

Rysunek



Tabela

## Testy wydajności

W zapytaniach łączono dane razem z danymi o rozkładzie jednostajnym z Tabeli milion(Rysunek 3), która została stworzona na podstawie złączenia Tabeli dziesięć(Rysunek 3) w następujący sposób:

create table Dziesiec(cyfra int,bit int);

create table Milion(liczba int,cyfra int, bit int);

insert into Dziesiec values (0,1);

…

insert into Dziesiec values (9,1);

CREATE TABLE Milion(liczba int,cyfra int, bit int); 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 ;

Obraz zawierający tekst, zrzut ekranu, Czcionka, linia

Opis wygenerowany automatycznie

Rysunek Schemat tabel

* 1. Parametry sprzętu i oprogramowania na którym zostały wykonane poniższe zapytania

1. CPU: Intel Core i5-8265U 1.6GHz with Turbo Boost up to 3.9 GHz
2. RAM: 8GB DDR4 Memory
3. SSD: 512GB PCIe NVMe
4. S.O. : Windows 11
5. Systemy zarządzania bazami danych: MySQL wersja 8.0.33 oraz PostgreSQL wersja
   1. Zapytania

1:

SELECT COUNT(\*) FROM Milion INNER JOIN GeoTabela ON (mod(Milion.liczba,68)=(GeoTabela.id\_pietro));

2:

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;

3:

SELECT COUNT(\*) FROM Milion WHERE mod(Milion.liczba,68)= (SELECT id\_pietro FROM GeoTabela WHERE mod(Milion.liczba,68)=(id\_pietro));

4:

SELECT COUNT(\*) FROM Milion WHERE mod(Milion.liczba,68)= (SELECT GeoPietro.id\_pietro FROM GeoPietro NATURAL JOIN GeoEpoka NATURAL JOIN GeoOkres NATURAL JOIN GeoEra NATURAL JOIN GeoEon;

Testy wykonano w dwóch częściach:

1. Z indeksami tylko na kluczach głównych tabel

CREATE INDEX IndEon ON GeoEon(id\_eon);

CREATE INDEX IndEpoka ON GeoEpoka(id\_epoka);

CREATE INDEX IndEra ON GeoEra(id\_era);

CREATE INDEX IndOkres ON GeoOkres(id\_okres);

CREATE INDEX IndPietro ON GeoPietro(id\_pietro);

CREATE INDEX IndTabGeo ON GeoTabela(id\_pietro);

1. Z indeksami na wszystkich kolumnach tabel, które brały udział w złączeniu

CREATE INDEX IndTab ON GeoTabela(id\_eon, id\_era, id\_okres, id\_epoka);

CREATE INDEX IndMilion ON Milion(Liczba);

## Wyniki testów

Powtarzając kilkukrotnie zapytania otrzymano następujące wyniki i zestawiono je w tabeli:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **1 ZL** | | **2 ZL** | | **3 ZG** | | **4 ZG** | |
|  | **MIN[s]** | **SR[s]** | **MIN[s]** | **SR[s]** | **MIN[s]** | **SR[s]** | **MIN[s]** | **SR[s]** |
| **BEZ INDEKSÓW** |  | | | | | | | |
| **MySQL** | 1,562 | 1,582 | 2,547 | 2,711 | 1,609 | 1,687 | 2,578 | 2,652 |
| **PostgreSQL** | 0,195 | 0,230 | 0,327 | 0,346 | 9,392 | 10,162 | 0,192 | 0,262 |
| **Z INDEKSAMI** |  | | | | | | | |
| **MySQL** | 0,875 | 0,965 | 1,407 | 1,473 | 0,953 | 1,039 | 1,437 | 1,480 |
| **PostgreSQL** | 0,166 | 0,193 | 0,280 | 0,320 | 9,398 | 9,763 | 0,196 | 0,224 |

## Wnioski

Podsumowując, czas wykonania zapytań w PostgreSQL jest mniejszy w przypadku wykonywania wszystkich zapytań, nie licząc 3, gdzie jest zdecydowanie dłuższy. Po dodaniu indeksowania czas jest krótszy i w MySQL jak i PostgreSQL.