SPRAWOZDANIE

AGH UNIVERSITY OF SCIENCE AND TECHNOLOGY

Temat: Porównanie wydajności złączeń i zagnieżdżeń

Numer ćwiczenia: ćwiczenie 9

Autor: Filip Hałys

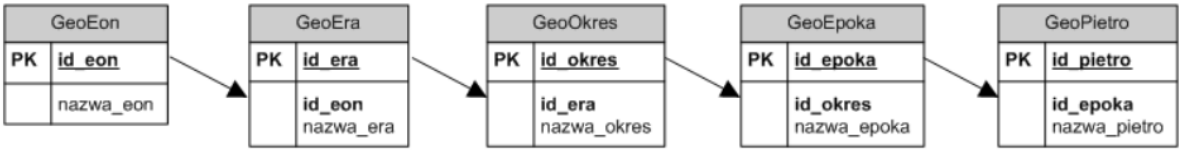
Wydział: WGGiOŚ

1. Cel ćwiczenia

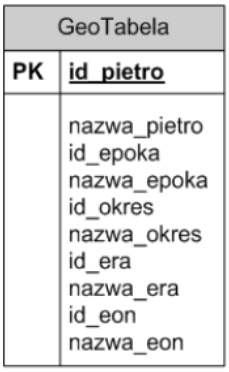
Celem wykonanego ćwiczenia było zbadanie wydajności kwerend bazujących na złączeniach i zagnieżdżeniach dla tabeli geochronologicznej przedstawionej poniżej *(Tabela 1)*:

Wiek (mln lat)	Eon	Era	Okres		Epoka
0,010	FANEROZOIK	Kenzoik	Czwar-torząd		Halocen
1,8					Plejstocen
22,5			Trzeciorząd	Neo-gen	Pliocen
					Miocen
65				Paleogen	Oligocen
					Eocen
					Paleocen
140		Mezozoik	Kreda		Górna
					Dolna
195			Jura		Górna
					Środkowa
					Dolna
230			Trias		Górna
					Środkowa
					Dolna
280		Paleozoik	Perm		Górny
					Dolny
345			Karbon		Górny
					Dolny
395			Dewon		Górny
					Środkowy
					Dolny

(Tabela 1) Tabela Geochronologiczna



(Schemat 1) Znormalizowany schemat tabeli geochronologicznej



(Schemat 2) Zdenormalizowany schemat tabeli geochronologicznej

Ćwiczenie wykonano na systemie PostgreSQL i SQL Server Management Studio, a następnie dokonano porównania i analizy otrzymanych wyników.

2. Konfiguracja sprzętowa

- Procesor: AMD Ryzen 5 4500U with Radeon Graphics 2.38 GHz
- Pamięć RAM: 8 GB
- Typ system: 64-bitowy system operacyjny, procesor x64
- System operacyjny: Windows 11 Home
- PostgreSQL: wersja 15.3; 64 Bit
- SQL Server Management Studio: wersja 19.0.1

3. Przeprowadzenie ćwiczenia

Na początku utworzono w postgreSQL znormalizowany schemat tabeli geochronologicznej zgodnie z modelem przedstawionym na *(schemat 1)*.

```sql
CREATE TABLE geo.GeoEon(
    id_eon INT PRIMARY KEY,
    nazwa_eon VARCHAR(30)
);

CREATE TABLE geo.GeoEra(
    id_era INT PRIMARY KEY,
    nazwa_era VARCHAR(30),
    id_eon INT
);

CREATE TABLE geo.GeoOkres(
    id_okres INT PRIMARY KEY,
    nazwa_okres VARCHAR(30),
    id_era INT
);
```

```sql
CREATE TABLE geo.GeoEpoka(
    id_epoka INT PRIMARY KEY,
    nazwa_epoka VARCHAR(30),
    id_okres INT
);

CREATE TABLE geo.GeoPietro(
    id_pietro INT PRIMARY KEY,
    nazwa_pietro VARCHAR(30),
    id_epoka INT
);
```

Wszystkie pięć tabeli uzupełniono danymi geochronologicznymi. Następnie utworzono tabele zdenormalizowaną, zgodnie z modelem przedstawionym na *(schemat 2)*.

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

W kolejnym kroku utworzono tabelę „Milion”, zawierającą milion liczb naturalnych z zakresu od 0 do 999 999. Została ona utworzona za pomocą autozłączenia tabeli „Dziesiec” (jest to tabela wypełniona cyframi 0-9). Liczby wygenerowano kodem:

```sql
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;
```

Ostatnim krokiem w tej części ćwiczenia było wygenerowanie czterech zapytań:

1. Zapytanie numer 1 (ZL) – celem tego zapytania było złączenie syntetycznej tablicy miliona wyników z tabelą geochronologiczną w postaci zdenormalizowanej, przy czym do warunku złączenia dodano operację modulo, dopasowującą zakresy wartości złączanych kolumn:

```sql
-- zapytanie nr 1
SELECT COUNT(*)
FROM Milion
    INNER JOIN GeoTabela ON (mod(Milion.liczba,68)=(GeoTabela.id_pietro));
```

2. Zapytanie numer 2 (ZL) – celem tego zapytania było złączenie syntetycznej tablicy miliona wyników z tabelą geochronologiczną w postaci zdenormalizowanej, przy czym złączenie jest wykonywane poprzez zagnieżdżenie skorelowane:

```sql
-- zapytanie nr 2
SELECT COUNT(*)
FROM Milion
    INNER JOIN  geo.GeoPietro
        ON (mod(Milion.liczba,68)=GeoPietro.id_pietro)
    NATURAL JOIN geo.GeoEpoka
    NATURAL JOIN geo.GeoOkres
    NATURAL JOIN geo.GeoEra
    NATURAL JOIN geo.GeoEon;
```

3. Zapytanie numer 3 (ZG) – celem tego zapytania było złączenie syntetycznej tabeli miliona wyników z tabelą geochronologiczną w postaci zdenormalizowanej, przy czym złączenie jest wykonywane poprzez zagnieżdżenie skorelowane:

```sql
-- zapytanie nr 2
SELECT COUNT(*)
FROM Milion
WHERE mod(Milion.liczba,68)= (SELECT id_pietro FROM GeoTabela   WHERE mod(Milion.liczba,68)=(id_pietro));
```

4. Zapytanie numer 4 (ZG) – celem tego zapytania było złączenie syntetycznej tabeli miliona wyników z tabelą geochronologiczną w postaci znormalizowanej, przy czym złączenie jest wykonywane poprzez zagnieżdżenie skorelowane, a zapytanie jest złączeniem tabel poszczególnych jednostek geochronologicznych:

```sql
-- zapytanie nr 4
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);
```

Wszystkie kody wykonane w tej części (3. Przeprowadzenie ćwiczenia) zostały odwzorowane w SQL Server Management Studio. Kody w obu środowiskach różnią się nieznacznie (zauważone różnice to chociażby inne nazwy funkcji).

4. Wyniki przeprowadzonych testów

Każdy test uruchomiono łącznie 20 razy; pięciokrotnie w PostgreSQL bez indeksowania, pięciokrotnie w PostgreSQL z indeksowaniem, pięciokrotnie w SQL Server Management Studio bez indeksowania oraz pięciokrotnie w SQL Server Management Studio z indeksowaniem. Warto zaznaczyć, że pomiary czasowe znacznie różniące się od pozostałych zostały usunięte, a w ich miejsce zrobione nowe.

Tabela *(tabela 3)* przedstawia statystyki (wartość minimalną czasu i wartość średnią) obliczone na potrzeby wyciągnięcia wniosków. Czas przedstawiono w jednostce milisekund (ms).

	1 (ZL)		2 (ZL)		3 (ZG)		4 (ZG)	
	MIN	AVG	MIN	AVG	MIN	AVG	MIN	AVG
				BEZINDEKSOWANIA				
PostgreSQL	221	231	461	470	8635	8772	212	216
SQL SMS	16	21	15	20	12	18	15	17
				Z INDEKSOWANIEM				
PostgreSQL	271	280	527	535	8749	8856	236	242
SQL SMS	15	17	26	27	84	87	22	24

(Tabela 3)

5. Wnioski

Na podstawie powyższych obserwacji zauważam, iż indeksacja wydłużyła czas wykonywania się poszczególnych zapytań. Zaledwie w jednym przypadku średnia czasu kompilacji się zmniejszyła po dodaniu indeksów (jest to średnia 1 testu w SQL Server Management Studio). Wnioskuję zatem, że indeksowanie niekorzystnie wpływa na wydajność zapytań.

Ponadto, dla stworzonych tabel SQL Server Management Studio sprawuje się znacznie lepiej niż PostgreSQL. Wyjątkowo dużą różnicę w czasach zaobserwowano dla testu 3.

Zaobserwowano również, że normalizacja w większości przypadków prowadzi do spadku wydajności. Zauważyć natomiast trzeba, iż pozwala ona na dużo lepsze segregowanie danych, niż ma się to w tabelach nieznormalizowanych. Argumentem za tabelami znormalizowanymi jest również fakt, że są one przyjazne dalszemu rozwojowi tabel (to znaczy modyfikacji, dodawaniu i usuwaniu danych).