SPRAWOZDANIE

AGH UNIVERSITY OF SCIENCE AND TECHNOLOGY

Temat: Porównanie wydajności złączeń i zagnieżdżeń dla schematów znormalizowanych i zdenormalizowanych w MSSQL i PostgreSQL

Numer ćwiczenia: ćwiczenie 10

Autor: Zuzanna Nóżka (415227)

Wydział: WGGiOŚ

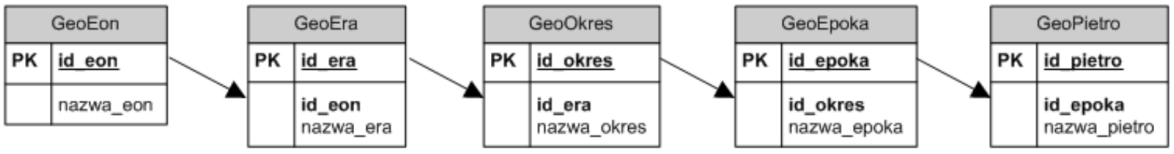
Spis treści

1. Cel ćwiczenia

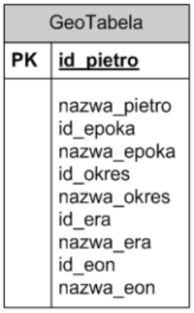
Celem ćwiczenia było odtworzenie badania opisanego w artykule „Wydajność złączeń i zagnieżdżeń dla schematów znormalizowanych i zdenormalizowanych” autorstwa Łukasza Jajeśnicy i Adama Piórkowskiego. Poniżej przedstawiono tabelę geochronologiczną (po wcześniejszym poprawieniu literówek oraz usunięciu kolumny wiek, która nie jest wykorzystywana w ćwiczeniu), dla której dokonano analiz. Widok uproszczono, usuwając kolumnę piętra, ze względu na dużą liczbę rekordów.

Eon	Era	Okres		Epoka
Fanerozoik	Kenozoik	Czwartorzęd		Holocen
				Plejstocen
		Trzeciorzęd	Neogen	Pliocen
				Miocen
			Paleogen	Oligocen
				Eocen
				Paleocen
	Mezozoik	Kreda		Górna
				Dolna
		Jura		Górna
				Środkowa
				Dolna
		Trias		Górny
				Środkowy
				Dolny
	Paleozoik	Perm		Górny
				Dolny
		Karbon		Górny
				Dolny
		Dewon		Górny
				Środkowy
				Dolny

Tabela 1.1 Tabela geochronologiczna



Schemat 1.1 Znormalizowany schemat tabeli geochronologicznej



Schemat 1.2 Zdenormalizowany schemat tabeli geochronologicznej

2. Konfiguracja sprzętowa

- **Procesor**: Intel(R) Core(TM) i7-6820HQ CPU @ 2.70GHz
- **Pamięć RAM:** 16,0 GB (2701 Mhz)
- **Dysk SSD:** Verbatim Vi560 S3 512 GB
- **System operacyjny:** Windows 10 Pro
- **MSSQL**, wersja Server Developer 16.0.1115
- **PostgreSQL**, wersja 16.2

3. Przeprowadzenie ćwiczenia

Na początku utworzono w bazie danych tabelę geochronologiczną w formie znormalizowanej, przedstawioną na *Schemat 1.1 Znormalizowany* schemat tabeli geochronologicznej

MSSQL	PostgreSQL

`CREATE TABLE GeoEon (`
`  id_eon INT PRIMARY KEY,`
`  nazwa_eon VARCHAR(50),`
`);`
`CREATE TABLE GeoEra (`
`  id_era INT PRIMARY KEY,`
`  id_eon INT,`
`  nazwa_era VARCHAR(50),`
`  FOREIGN KEY (id_eon) REFERENCES GeoEon(id_eon)`
`);`
`CREATE TABLE GeoOkres (`
`  id_okres INT PRIMARY KEY,`
`  id_era INT,`
`  nazwa_okres VARCHAR(50),`
`  FOREIGN KEY (id_era) REFERENCES GeoEra(id_era)`
`);`
`CREATE TABLE GeoEpoka (`
`  id_epoka INT PRIMARY KEY,`
`  id_okres INT,`
`  nazwa_epoka VARCHAR(50),`
`  FOREIGN KEY (id_okres) REFERENCES GeoOkres(id_okres)`
`);`
`CREATE TABLE GeoPietro (`
`  id_pietro INT PRIMARY KEY,`
`  id_epoka INT,`
`  nazwa_pietro VARCHAR(50),`
`  FOREIGN KEY (id_epoka) REFERENCES GeoEpoka(id_epoka)`
`);`
	`CREATE TABLE GeoEon (`
	`  id_eon SERIAL PRIMARY KEY,`
	`  nazwa_eon VARCHAR(50)`
	`);`
	`CREATE TABLE GeoEra (`
	`  id_era SERIAL PRIMARY KEY,`
	`  id_eon INT,`
	`  nazwa_era VARCHAR(50),`
	`  FOREIGN KEY (id_eon) REFERENCES GeoEon(id_eon)`
	`);`
	`CREATE TABLE GeoOkres (`
	`  id_okres SERIAL PRIMARY KEY,`
	`  id_era INT,`
	`  nazwa_okres VARCHAR(50),`
	`  FOREIGN KEY (id_era) REFERENCES GeoEra(id_era)`
	`);`
	`CREATE TABLE GeoEpoka (`
	`  id_epoka SERIAL PRIMARY KEY,`
	`  id_okres INT,`
	`  nazwa_epoka VARCHAR(50),`
	`  FOREIGN KEY (id_okres) REFERENCES GeoOkres(id_okres)`
	`);`
	`CREATE TABLE GeoPietro (`
	`  id_pietro SERIAL PRIMARY KEY,`
	`  id_epoka INT,`
	`  nazwa_pietro VARCHAR(50),`
	`  FOREIGN KEY (id_epoka) REFERENCES GeoEpoka(id_epoka)`
	`);`

Po wypełnieniu tabel wartościami, połączono je do tabeli zdenormalizowanej *GeoTabela* (zgodnie ze *Schemat 1.2* Zdenormalizowany schemat tabeli geochronologicznej):

MSSQL	PostgreSQL

`CREATE TABLE GeoTabela (`
`  id_pietro INT PRIMARY KEY,`
`  nazwa_pietro VARCHAR(50),`
`  id_epoka INT,`
	`CREATE TABLE GeoTabela (`
	`  id_pietro INT PRIMARY KEY,`
	`  nazwa_pietro VARCHAR(50),`
	`  id_epoka INT,`

```
    nazwa_epoka VARCHAR(50),
    id_okres INT,
    nazwa_okres VARCHAR(50),
    id_era INT,
    nazwa_era VARCHAR(50),
    id_eon INT,
    nazwa_eon VARCHAR(50)
);
INSERT INTO GeoTabela (id_pietro, nazwa_pie-
tro, id_epoka, nazwa_epoka, id_okres, na-
zwa_okres, id_era, nazwa_era, id_eon, na-
zwa_eon)
SELECT
    GeoPietro.id_pietro,
    GeoPietro.nazwa_pietro,
    GeoEpoka.id_epoka,
    GeoEpoka.nazwa_epoka,
    GeoOkres.id_okres,
    GeoOkres.nazwa_okres,
    GeoEra.id_era,
    GeoEra.nazwa_era,
    GeoEon.id_eon,
    GeoEon.nazwa_eon
FROM GeoPietro
       INNER JOIN GeoEpoka ON GeoPie-
tro.id_epoka = GeoEpoka.id_epoka
       INNER JOIN GeoOkres ON GeoE-
poka.id_okres = GeoOkres.id_okres
       INNER JOIN GeoEra ON GeoOkres.id_era
= GeoEra.id_era
       INNER JOIN GeoEon ON GeoEra.id_eon =
GeoEon.id_eon;
```

```
    nazwa_epoka VARCHAR(50),
    id_okres INT,
    nazwa_okres VARCHAR(50),
    id_era INT,
    nazwa_era VARCHAR(50),
    id_eon INT,
    nazwa_eon VARCHAR(50)
);
INSERT INTO GeoTabela (id_pietro, nazwa_pie-
tro, id_epoka, nazwa_epoka, id_okres, na-
zwa_okres, id_era, nazwa_era, id_eon, na-
zwa_eon)
SELECT
    GeoPietro.id_pietro,
    GeoPietro.nazwa_pietro,
    GeoEpoka.id_epoka,
    GeoEpoka.nazwa_epoka,
    GeoOkres.id_okres,
    GeoOkres.nazwa_okres,
    GeoEra.id_era,
    GeoEra.nazwa_era,
    GeoEon.id_eon,
    GeoEon.nazwa_eon
FROM GeoPietro
       INNER JOIN GeoEpoka ON GeoPie-
tro.id_epoka = GeoEpoka.id_epoka
       INNER JOIN GeoOkres ON GeoE-
poka.id_okres = GeoOkres.id_okres
       INNER JOIN GeoEra ON GeoOkres.id_era
= GeoEra.id_era
       INNER JOIN GeoEon ON GeoEra.id_eon =
GeoEon.id_eon;
```

Następnie, na podstawie tabeli *Dziesiec* wypełnionej kolejnymi liczbami naturalnymi od 0 do 9, utworzono tabelę *Milion*, wypełnioną liczbami od 0 do 999 999:

MSSQL

```
CREATE TABLE Dziesiec (cyfra INT PRIMARY KEY, bit INT);
INSERT INTO Dziesiec (cyfra, bit) VALUES
       (0, 0), (1, 1), (2, 0), (3, 1), (4, 0), (5, 1), (6, 0), (7, 1), (8, 0), (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 ;
```

PostgreSQL

```
CREATE TABLE Dziesiec (cyfra INT PRIMARY KEY, bit INT);
INSERT INTO Dziesiec (cyfra, bit) VALUES
       (0, 0), (1, 1), (2, 0), (3, 1), (4, 0), (5, 1), (6, 0), (7, 1), (8, 0), (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 CROSS JOIN Dziesiec a2 CROSS JOIN Dziesiec a3 CROSS JOIN Dziesiec
a4 CROSS JOIN Dziesiec a5 CROSS JOIN Dziesiec a6 ;
```

Właściwa część ćwiczenia – badająca wydajność – została przeprowadzona z wykorzystaniem czterech zapytań:

I. Zapytanie 1 (1 ZL) – złączenie tabeli *Milion* z tabelą geochronologiczną w postaci zdenormalizowanej, przy czym do warunku złączenia dodano operację modulo, dopasowującą zakresy wartości złączanych kolumn:

MSSQL

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

PostgreSQL

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

II. Zapytanie 2 (2 ZL) – złączenie tabeli *Milion* z tabelą geochronologiczną w postaci znormalizowanej, reprezentowaną przez złączenia pięciu tabel:

MSSQL

```
SELECT COUNT(*) FROM Milion
       INNER JOIN GeoPietro ON(Milion.liczba % 68 = GeoPietro.id_pietro)
       INNER JOIN GeoEpoka ON GeoPietro.id_epoka = GeoEpoka.id_epoka
       INNER JOIN GeoOkres ON GeoEpoka.id_okres = GeoOkres.id_okres
       INNER JOIN GeoEra ON GeoOkres.id_era = GeoEra.id_era
       INNER JOIN GeoEon ON GeoEra.id_eon = GeoEon.id_eon;
```

PostgreSQL

```
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;
```

III. Zapytanie 3 (3 ZG) – złączenie tabeli *Milion* z tabelą geochronologiczną w postaci zdenormalizowanej, przy czym złączenie jest wykonywane poprzez zagnieżdżenie skorelowane:

MSSQL

```
SELECT COUNT(*) FROM Milion
       WHERE Milion.liczba % 68 = (SELECT id_pietro FROM GeoTabela
              WHERE Milion.liczba % 68 = id_pietro);
```

PostgreSQL

```
SELECT COUNT(*) FROM Milion
       WHERE mod(Milion.liczba, 68) = (SELECT id_pietro FROM GeoTabela
              WHERE mod(Milion.liczba, 68) = (id_pietro));
```

IV. Zapytanie 4 (4 ZG) – złączenie syntetycznej tabeli *Milion* z tabelą geochronologiczną w postaci znormalizowanej, przy czym złączenie jest wykonywane poprzez zagnieżdżenie skorelowane, a zapytanie wewnętrzne jest złączeniem tabel poszczególnych jednostek geochronologicznych:

MSSQL

```
SELECT COUNT(*) FROM Milion
       WHERE Milion.liczba % 68 IN (SELECT GeoPietro.id_pietro FROM GeoPietro
              JOIN GeoEpoka ON GeoPietro.id_epoka = GeoEpoka.id_epoka
              INNER JOIN GeoOkres ON GeoEpoka.id_okres = GeoOkres.id_okres
              INNER JOIN GeoEra ON GeoOkres.id_era = GeoEra.id_era
              INNER JOIN GeoEon ON GeoEra.id_eon = GeoEon.id_eon);
```

PostgreSQL

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

```
            NATURAL JOIN GeoEon);
```

Zapytania wykonano zarówno bez indeksów, jak i z dodanym indeksowaniem:

MSSQL

```
CREATE INDEX iEon ON GeoEon(id_eon);
CREATE INDEX iEra ON GeoEra(id_era, id_eon);
CREATE INDEX iOkres ON GeoOkres(id_okres, id_era);
CREATE INDEX iEpoka ON GeoEpoka(id_epoka, id_okres);
CREATE INDEX iPietro ON GeoPietro(id_pietro, id_epoka);
CREATE INDEX iLiczba ON Milion(liczba);
CREATE INDEX iGeoTabela ON GeoTabela(id_pietro, id_epoka, id_era, id_okres,id_eon);
```

PostgreSQL

```
CREATE INDEX iEon ON GeoEon(id_eon);
CREATE INDEX iEra ON GeoEra(id_era, id_eon);
CREATE INDEX iOkres ON GeoOkres(id_okres, id_era);
CREATE INDEX iEpoka ON GeoEpoka(id_epoka, id_okres);
CREATE INDEX iPietro ON GeoPietro(id_pietro, id_epoka);
CREATE INDEX iLiczba ON Milion(liczba);
CREATE INDEX iGeoTabela ON GeoTabela(id_pietro, id_epoka, id_era, id_okres,id_eon);
```

Czas pomierzono funkcjami:

MSSQL

```
SET STATISTICS IO, TIME ON

DBCC FREEPROCCACHE;
DBCC DROPCLEANBUFFERS;
CHECKPOINT
GO

-- zapytanie 1 --

DBCC FREEPROCCACHE;
DBCC DROPCLEANBUFFERS;
CHECKPOINT
GO

...

SET STATISTICS TIME OFF
```

PostgreSQL

```
EXPLAIN (ANALYZE, BUFFERS)
       -- zapytanie 1 --
;
CHECKPOINT;

EXPLAIN (ANALYZE, BUFFERS)
       -- zapytanie 2 --
;
CHECKPOINT;

...
```

4. Wyniki

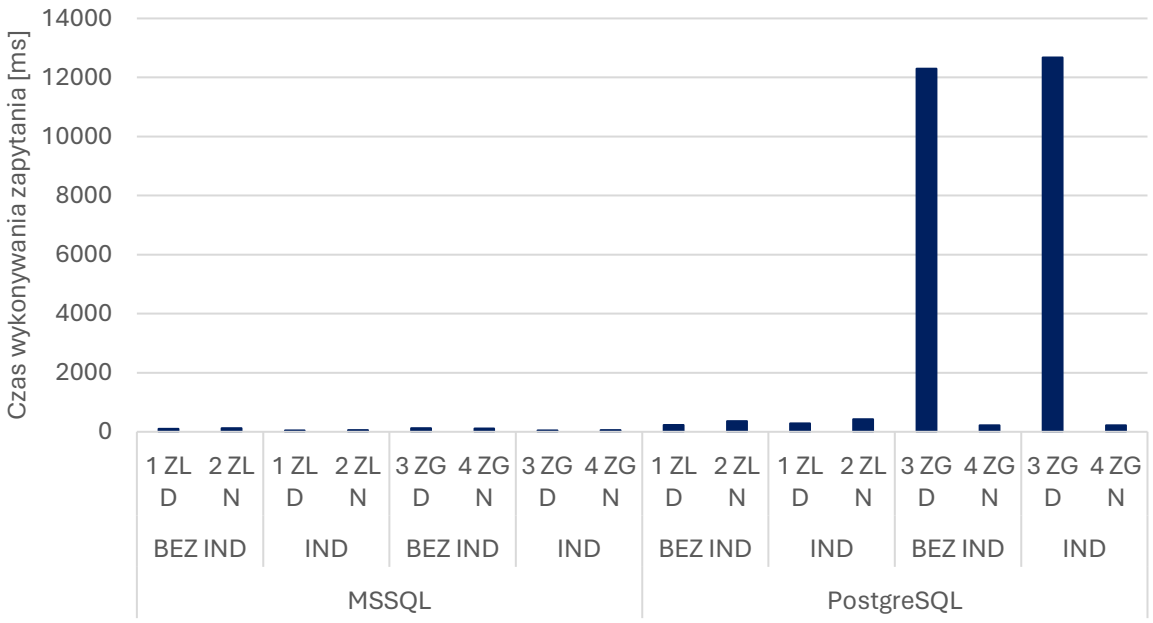
Każde z zapytań zostało wykonane 20 razy:

- 5x w MSSQL bez indeksowania
- 5x w PostgreSQL bez indeksowania
- 5x w MSSQL z indeksowaniem
- 5x w PostgreSQL z indeksowaniem,

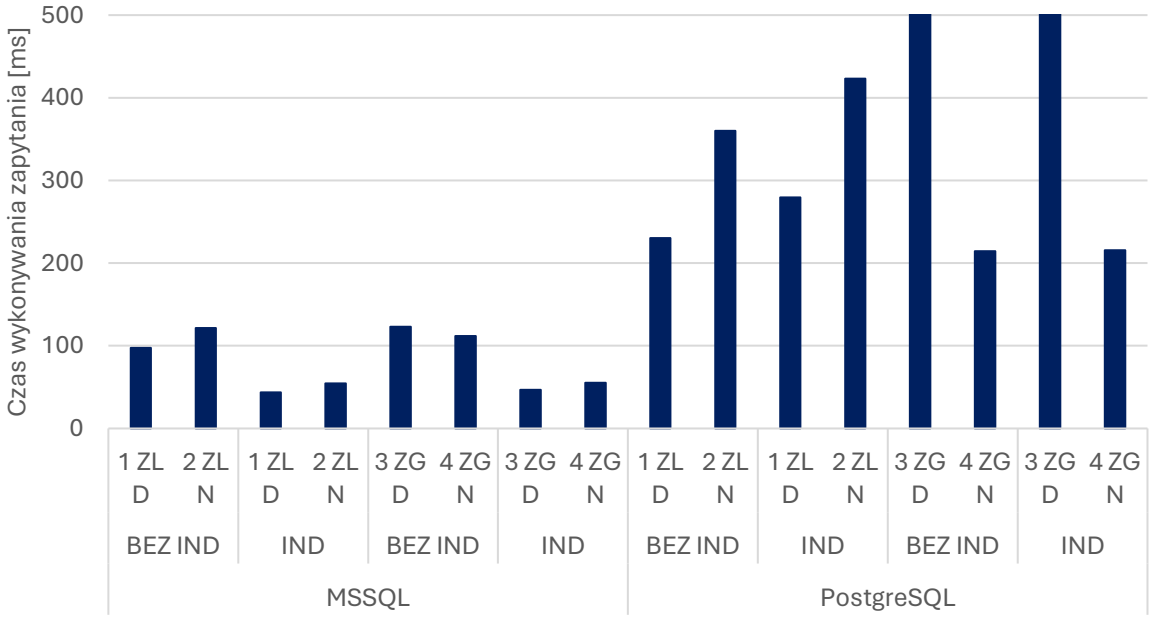
a z otrzymanych pomiarów czasu wybrano wartości najmniejsze oraz wyliczono średnie. Wyniki w milisekundach [ms] zestawiono w poniższej tabeli:

	1 ZL		2 ZL		3 ZG		4 ZG	
	BEZ INDEKSÓW							
	MIN	**ŚR**	**MIN**	**ŚR**	**MIN**	**ŚR**	**MIN**	**ŚR**
MSSQL	96	97,6	105	121,4	101	123	102	111,8
PostgreSQL	202	230,4	351	360	12057	12294	182	214,4
	Z INDEKSAMI							
MSSQL	26	43,8	51	54,6	45	46,6	52	55,4
PostgreSQL	227	279,6	328	423	11961	12682,2	192	215,6

Tabela 4.1 Zestawienie pomiarów czasu dla zapytań



Wykres 4.1 Zestawienie pomiarów czasu w formie pełnego wykresu



Wykres 4.2 Zestawienie pomiarów czasu w formie wykresu – ograniczenie osi pionowej do 500 [ms]

5. Wnioski

I. System bazy danych:

MSSQL wykazał znacznie wyższą wydajność dla wszystkich testowanych zapytań. W przypadku bez indeksów był co najmniej 2 razy szybszy przy każdym zapytaniu, a z zastosowaniem indeksów różnica ta wzrastała do co najmniej 4 razy. PostgreSQL wykazywał o wiele dłuższe czasy wykonywania, a szczególnie ekstremalne wyniki osiągał dla zapytania 3 ZG.

II. Indeksowanie:

Dodanie indeksów poprawiło wydajność w MSSQLu o mniej więcej połowę, szczególne korzyści przyniosło dla zapytań na schemacie zdenormalizowanym. Natomiast w przypadku PostgreSQLa, jeszcze bardziej wydłużyło czasy wykonywania zapytań.

III. Normalizacja:

W przypadku MSSQL zapytania na schemacie zdenormalizowanym (1 ZL, 3 ZG) były w większości przypadków nieco szybsze niż na znormalizowanym (2 ZL, 4 ZG), jednak różnice te były minimalne. Dodanie indeksów dodatkowo je zmniejszyło. W PostgreSQL zapytanie 1 ZL na schemacie zdenormalizowanym również było szybsze od 2 ZL na znormalizowanym, natomiast 3 ZG wykazywało ekstremalnie wyższy czas wykonywania niż 4 ZG.

IV. Zapytania zagnieżdżone:

Zapytania niezagnieżdżone (1 ZL, 2 ZL) nie wykazały znacząco wyższej wydajności niż zapytania zagnieżdżone (3 ZG, 4 ZG) w obu systemach baz danych. Jedynym wyjątkiem jest 3 ZG, które w PostgreSQL osiągnęło ekstremalnie wysoki czas wykonywania, niezależnie od obecności indeksów.

Podsumowując, MSSQL wydaje się być bardziej wydajnym systemem bazodanowym dla przeprowadzonych testów, a jego efektywność można dodatkowo podnieść stosując indeksowanie. Normalizacja tabel jedynie nieznacznie spowalnia wykonywanie zapytania, zatem w większości przypadków warto wprowadzać ją do bazy, w celu lepszego uporządkowania danych.