Kraków, 25.05.2023 r.

Bartosz Staroń, nr albumu: 410642

Badanie wydajności zagnieżdżeń i złączeń w zależności od schematów zdenormalizowanych oraz znormalizowanych

Wstęp

Celem przeprowadzonego projektu było sprawdzenie zależności między normalizacją bazy danych, a czasem wykonywania poleceń w języku SQL zawierających złączenia i zagnieżdżenia. Aby tego dokonać utworzono tabele, które zdenormalizowano do tabeli stratygraficznej (Tab. 1), a następnie przeprowadzono 4 testy.

Eon	Era	Okres	Epoka
Fanerozoik	Kenozoik	Czwartorzęd	Holocen
			Plejstocen
		Neogen	Pliocen
			Miocen
		Paleogen	Oligocen
			Eocen
			Paleocen
	Mezozoik	Kreda	Kreda górna
			Kreda dolna
		Jura	Jura górna
			Jura środkowa
			Jura dolna
		Trias	Trias górny
			Trias środkowy
			Trias dolny
	Paleozoik	Perm	Loping
			Gwadalup
			Cisural
		Karbon	Pensylwan
			Missisip
		Dewon	Dewon górny
			Dewon środkowy
			Dewon dolny
		Sylur	Przydol
			Ludlow
			Wenlok
			Landower
		Ordowik	Ordowik górny
			Ordowik środkowy
			Ordowik dolny
		Kambr	Furong
			Oddział 3 (kambr)
			Oddział 2 (kambr)
			Terenew

Tab. 1 – Tabela stratygraficzna (bez pięter) na podstawie danych ICS

Konfiguracja sprzętowa i programowa

Do zarządzania bazą danych użyto systemów ogólnodostępnych:

- MySQL 8.0.33 Community Edition / MySQL Workbench 8.0
- PostgreSQL 15.3 / pgAdmin 4 v7.1
- Microsoft SQL Server / SQL Server Management Studio 19

Testy zostały przeprowadzone na komputerze o następującej specyfikacji:

- CPU: Intel(R) Core(TM) i7-8650U CPU @ 1.90GHz
- SSD: SAMSUNG MZVLW256HEHP-000L7
- RAM: 16GB
- SO: Windows 10 Pro

Utworzenie tabeli stratygraficznej przy użyciu denormalizacji

Aby otrzymać zdenormalizowaną formę tabeli stratygraficznej *TabelaStr*, utworzono znormalizowane tabele *GeoEon, GeoEra, GeoOkres, GeoEpoka* i *GeoPietro*.

```
CREATE TABLE tabela_stratygraficzna.GeoPietro(
	id_pietro VARCHAR(10) PRIMARY KEY,
	id_epoka VARCHAR(10) NOT NULL,
	nazwa_pietro VARCHAR(40) NOT NULL
);
```

Przykładowy kod dla tabeli GeoPietro w PostgreSQL

Następnie przy pomocy złączeń naturalny, połączono stworzone tabele, czego wynikiem była zdenormalizowana tabela *TabelaStr.*

```
SELECT
geo_p.id_pietro AS ID_pietra,
geo_p.nazwa_pietro AS Pietro,
geo_ep.id_epoka AS ID_epoki,
geo_ep.nazwa_epoka AS Epoka,
geo_o.id_okres AS ID_okresu,
geo_o.nazwa_okres AS Okres,
geo_er.id_era AS ID_ery,
geo_er.nazwa_era AS Era,
geo_eo.id_eon AS ID_eonu,
geo_eo.nazwa_eon AS Eon
INTO
tabela_stratygraficzna.TabelaStr
FROM
tabela_stratygraficzna.GeoPietro geo_p
NATURAL JOIN
tabela_stratygraficzna.GeoEpoka geo_ep
NATURAL JOIN
tabela_stratygraficzna.GeoOkres geo_o
NATURAL JOIN
tabela_stratygraficzna.GeoEra geo_er
NATURAL JOIN
tabela_stratygraficzna.GeoEon geo_eo;
```

Przykładowy kod w PostgreSQL

Przeprowadzenie testów

Do zbadania wydajności zagnieżdżeń i złączeń wykonano 4 testy. W celu ich przeprowadzenia utworzono tabelę *milion* zawierającą liczby od 0 do 999999 (tabela *dziesięć* zawierała cyfry 0, 1, 2, 3, 4, 5, 6, 7, 8, 9). Podczas testów tabele nie posiadały indeksów.

```
SELECT
      d1.wartosc+10*d2.wartosc+100*d3.wartosc+1000*d4.wartosc+10000*d5.wartosc+1
00000*d6.wartosc AS liczba
INTO
      liczby.milion
FROM
      liczby.dziesiec d1,
      liczby.dziesiec d2,
      liczby.dziesiec d3,
      liczby.dziesiec d4,
      liczby.dziesiec d5,
      liczby.dziesiec d6;
```

Przykładowy kod w PostgreSQL

- Test 1
  Celem testu było złączenie tabeli zawierającej milion wyników oraz zdenormalizowanej tabeli stratygraficznej. W celu dopasowania wartości obu tablic użyto operacji modulo.

  PostgreSQL:

  ```
  SELECT
        COUNT(*)
  FROM
        liczby.milion m
  INNER JOIN
        tabela_stratygraficzna.TabelaStr t
  ON
        m.liczba%95=CAST(RIGHT(t.ID_pietra, LENGTH(t.ID_pietra)-3) AS INT);
  ```

  MySQL:

  ```
  SELECT
        COUNT(*)
  FROM
        liczby.milion m
  INNER JOIN
        tabela_stratygraficzna.TabelaStr t
  ON
        m.liczba%95=CAST(RIGHT(t.ID_pietra, LENGTH(t.ID_pietra)-3) AS UNSIGNED);
  ```

  SQL Server:

  ```
  SELECT
        COUNT(*)
  FROM
        liczby.milion m
  INNER JOIN
        tabela_stratygraficzna.TabelaStr t
  ON
        m.liczba%95=CAST(RIGHT(t.ID_pietra, LEN(t.ID_pietra)-3) AS INT);
  ```

- Test 2
  Celem testu było złączenie tabeli zawierającej milion wyników oraz znormalizowanej tabeli stratygraficznej (złączenia 5 tabel). W celu dopasowania wartości obu tablic użyto operacji modulo.

  PostgreSQL:

```
SELECT
	COUNT(*)
FROM
	liczby.milion m
INNER JOIN
	tabela_stratygraficzna.GeoPietro geo_p
ON
	m.liczba%95=CAST(RIGHT(geo_p.id_pietro, LENGTH(geo_p.id_pietro)-3) AS
INT)
NATURAL JOIN
	tabela_stratygraficzna.GeoEpoka geo_ep
NATURAL JOIN
	tabela_stratygraficzna.GeoOkres geo_o
NATURAL JOIN
	tabela_stratygraficzna.GeoEra geo_er
NATURAL JOIN
	tabela_stratygraficzna.GeoEon geo_eo;
```

  MySQL:

```
SELECT
	COUNT(*)
FROM
	liczby.milion m
INNER JOIN
	tabela_stratygraficzna.GeoPietro geo_p
ON
	m.liczba%95=CAST(RIGHT(geo_p.id_pietro, LENGTH(geo_p.id_pietro)-3) AS
UNSIGNED)
NATURAL JOIN
	tabela_stratygraficzna.GeoEpoka geo_ep
NATURAL JOIN
	tabela_stratygraficzna.GeoOkres geo_o
NATURAL JOIN
	tabela_stratygraficzna.GeoEra geo_er
NATURAL JOIN
	tabela_stratygraficzna.GeoEon geo_eo;
```

  SQL Server:

```
SELECT
	COUNT(*)
FROM
	liczby.milion m
INNER JOIN
	tabela_stratygraficzna.GeoPietro geo_p
ON
	m.liczba%95=CAST(RIGHT(geo_p.id_pietro, LEN(geo_p.id_pietro)-3) AS INT)
INNER JOIN
	tabela_stratygraficzna.GeoEpoka geo_ep
```

```
ON
	geo_p.id_epoka=geo_ep.id_epoka
INNER JOIN
	tabela_stratygraficzna.GeoOkres geo_o
ON
	geo_ep.id_okres=geo_o.id_okres
INNER JOIN
	tabela_stratygraficzna.GeoEra geo_er
ON
	geo_o.id_era=geo_er.id_era
INNER JOIN
	tabela_stratygraficzna.GeoEon geo_eo
ON
	geo_er.id_eon=geo_eo.id_eon;
```

- Test 3
  Celem testu było złączenie tabeli zawierającej milion wyników oraz zdenormalizowanej tabeli stratygraficznej. W celu dopasowania wartości obu tablic użyto operacji modulo. Złączenie zostało przeprowadzone poprzez zagnieżdżenie skorelowane.

  PostgreSQL:

```
SELECT
	COUNT(*)
FROM
	liczby.milion m
WHERE
	m.liczba%95=
	(SELECT
		CAST(RIGHT(t.ID_pietra, LENGTH(t.ID_pietra)-3) AS INT)
	FROM
		tabela_stratygraficzna.TabelaStr t
	WHERE
		m.liczba%95=CAST(RIGHT(t.ID_pietra, LENGTH(t.ID_pietra)-3) AS INT));
```

  MySQl:

```
	COUNT(*)
FROM
	liczby.milion m
WHERE
	m.liczba%95=
	(SELECT
		CAST(RIGHT(t.ID_pietra, LENGTH(t.ID_pietra)-3) AS UNSIGNED)
	FROM
		tabela_stratygraficzna.TabelaStr t
	WHERE
		m.liczba%95=CAST(RIGHT(t.ID_pietra, LENGTH(t.ID_pietra)-3) AS UNSIGNED));
```

  SQL Server:

```
SELECT
	COUNT(*)
FROM
	liczby.milion m
WHERE
	m.liczba%95=
	(SELECT
		CAST(RIGHT(t.ID_pietra, LEN(t.ID_pietra)-3) AS INT)
```

```
FROM
	tabela_stratygraficzna.TabelaStr t
WHERE
	m.liczba%95=CAST(RIGHT(t.ID_pietra, LEN(t.ID_pietra)-3) AS INT));
```

- Test 4

  Celem testu było złączenie tabeli zawierającej milion wyników oraz znormalizowanej tabeli stratygraficznej. W celu dopasowania wartości obu tablic użyto operacji modulo. Złączenie zostało przeprowadzone poprzez zagnieżdżenie skorelowane, gdzie zapytanie wewnętrzne to złączenie pięciu tabel.

  PostgreSQL:

```
SELECT
	COUNT(*)
FROM
	liczby.milion m
WHERE
	m.liczba%95 IN
	(SELECT
		CAST(RIGHT(geo_p.id_pietro, LENGTH(geo_p.id_pietro)-3) AS INT)
	FROM
		tabela_stratygraficzna.GeoPietro geo_p
	NATURAL JOIN
		tabela_stratygraficzna.GeoEpoka geo_ep
	NATURAL JOIN
		tabela_stratygraficzna.GeoOkres geo_o
	NATURAL JOIN
		tabela_stratygraficzna.GeoEra geo_er
	NATURAL JOIN
		tabela_stratygraficzna.GeoEon geo_eo);
```

  MySQL:

```
SELECT
	COUNT(*)
FROM
	liczby.milion m
WHERE
	m.liczba%95 IN
	(SELECT
		CAST(RIGHT(geo_p.id_pietro, LENGTH(geo_p.id_pietro)-3) AS
UNSIGNED)
	FROM
		tabela_stratygraficzna.GeoPietro geo_p
	NATURAL JOIN
		tabela_stratygraficzna.GeoEpoka geo_ep
	NATURAL JOIN
		tabela_stratygraficzna.GeoOkres geo_o
	NATURAL JOIN
		tabela_stratygraficzna.GeoEra geo_er
	NATURAL JOIN
		tabela_stratygraficzna.GeoEon geo_eo);
```

SQL Server:

```
SELECT
	COUNT(*)
FROM
	liczby.milion m
WHERE
	m.liczba%95 IN
	(SELECT
		CAST(RIGHT(geo_p.id_pietro, LEN(geo_p.id_pietro)-3) AS INT)
	FROM
		tabela_stratygraficzna.GeoPietro geo_p
	INNER JOIN
		tabela_stratygraficzna.GeoEpoka geo_ep
	ON
		geo_p.id_epoka=geo_ep.id_epoka
	INNER JOIN
		tabela_stratygraficzna.GeoOkres geo_o
	ON
		geo_ep.id_okres=geo_o.id_okres
	INNER JOIN
		tabela_stratygraficzna.GeoEra geo_er
	ON
		geo_o.id_era=geo_er.id_era
	INNER JOIN
		tabela_stratygraficzna.GeoEon geo_eo
	ON
		geo_er.id_eon=geo_eo.id_eon);
```

Wszystkie testy zostały przeprowadzone również dla tabel z nałożonymi indeksami.

```
CREATE INDEX ix_geoeon_eon
ON tabela_stratygraficzna.GeoEon(id_eon);
```

Przykładowy kod w PostgreSQL

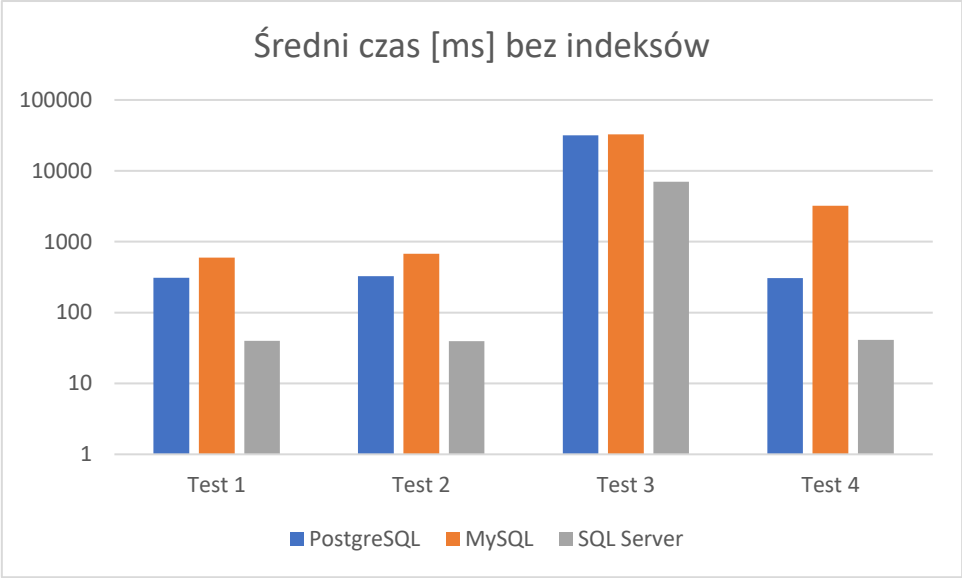
Wyniki

Każdy z testów został wykonany 10 razy. Następnie z otrzymanych czasów obliczono średnią. Uzyskane wyniki przedstawiono poniżej (Tab. 2, Tab. 3), czas jest wyrażony w milisekundach.

Testy bez nałożonych indeksów:

	Test 1			Test 2			Test 3			Test 4		
	Min	Max	Śr	Min	Max	Śr	Min	Max	Śr	Min	Max	Śr
PostgreSQL	259	372	310,7	256	385	327,9	30115	32534	31677,9	269	369	305,3
MySQL	562	640	598,2	641	718	673,6	29906	36094	32617,2	3109	3312	3203,1
SQL Server	39	42	39,9	38	42	39,6	6014	7491	7047,1	39	46	41

Tab. 2 – maksymalny, minimalny i średni czas dla tabel bez indeksów

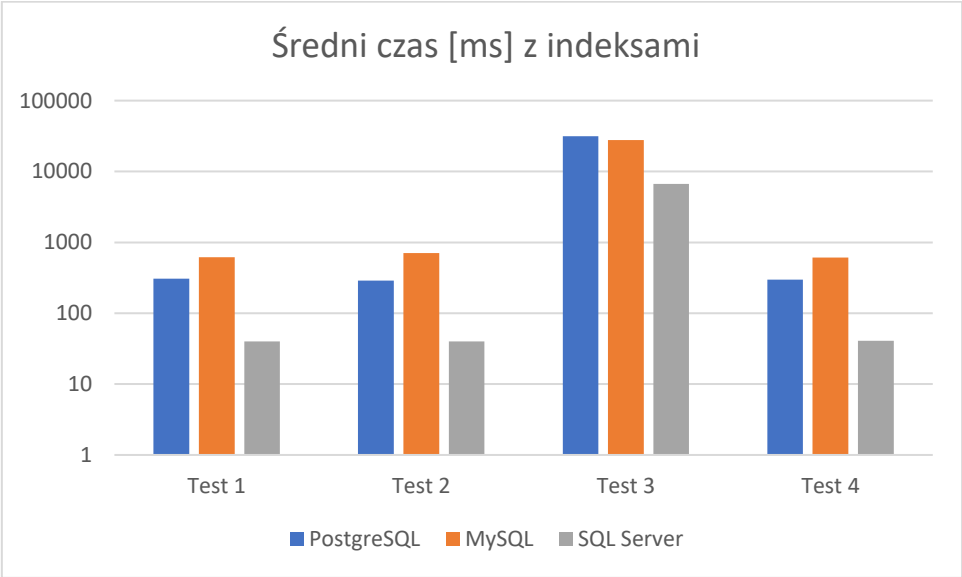


Wyk. 1 – Średni czas wykonania testów bez indeksów

Testy z nałożonymi indeksami:

	Test 1			Test 2			Test 3			Test 4		
	Min	Max	Śr	Min	Max	Śr	Min	Max	Śr	Min	Max	Śr
PostgreSQL	267	364	305,7	261	344	288,7	29580	32185	31546,3	253	339	297,3
MySQL	578	735	618,7	671	766	706,2	25968	28954	27857,5	562	641	609,3
SQL Server	38	43	39,8	37	41	39,8	5874	7447	6640,1	40	44	40,9

Tab. 3 – maksymalny, minimalny i średni czas dla tabel bez indeksów



Wyk. 2 – Średni czas wykonania testów z indeksami

Wnioski

Po analizie otrzymanych wyników stwierdzono następujące wnioski:

- Indeksowanie tabel poprawiło wydajność zapytań w przypadku zagnieżdżeń, jednak używanie jej podczas złączeń, w niektórych przypadkach, ją zmniejszało
- Średni czas wykonywania zapytań jest najkrótszy dla bazy SQL Server, a najdłuższy dla MySQL
- W przypadku zagnieżdżeń normalizacja tabeli znacznie polepsza wydajność zapytania
- W przypadku złączeń korzystanie z postaci zdenormalizowanej przyspiesza wykonywanie zapytań w stosunku do postaci znormalizowanej (w PostgreSQL i MySQL)

Bibliografia:

1. Jaję́nica Ł., Piórkowski A.: *Wydajność złączeń i zagnieżdżeń dla schematów znormalizowanych i zdenormalizowanych*. Studia Informatica, Wyd. Pol. Śląskiej, Vol. 31, No. 2A (89), 2010
2. Rockoff L., *Język SQL. Przyjazny podręcznik. Wydanie III,* Helion S.A., Gliwice 2022
3. https://stratygrafia.pgi.gov.pl/Chronostratigraphy/Table - dostęp: 22.05.2023