Wydajność złączeń i zagnieżdżeń dla schematów znormalizowanych i zdenormalizowanych

Bazy danych. Ćwiczenie 10 | Szymon Hyziak | Geoinformatyka

Spis treści

Abstrakt 1
Wstęp 2
Normalizacja 2
Konfiguracja sprzętowa 2
Testy wydajności 3
  Zapytania do bazy danych 5
  Uwagi wstępne 7
  SZBD SQL Server 7
    Część I 7
    Część II 8
    Podsumowanie 9
  SZBD PostgreSQL 10
    Część I 10
    Część II 11
    Podsumowanie 12
Interpretacja i analiza wyników 12
Bibliografia 13

Abstrakt

W sprawozdaniu przeanalizowano osiągnięcia pod względem wydajności znormalizowanych oraz zdenormalizowanych schematów baz danych. Skupiono się na bazach i hurtowniach danych znacznych rozmiarów, gdyż stwierdzono, że omówione zagadnienie dot. w szczególności takich kontekstów wykorzystywanych w praktyce.

Poza konstrukcjami językowymi typowymi dla SQL sprawdzono także 2 osobne silniki bazodanowe, którymi są PostgreSQL oraz SQL Server. Dane, którymi posłużono się w czasie prac dot. wymiaru czasu geologicznego.

W połowie przypadków zaobserwowano przyspieszenie wykonywania zapytań do bazy danych po ówczesnym nałożeniu na odpowiednie kolumny w tabelach indeksów nieklastrowanych. Stwierdzono również, że w przeprowadzonych badaniach SQL Server był SZBD gwarantującym zauważalnie krótszy czas wykonywania komend niż PostgreSQL.

Wstęp

Niniejszy tekst rozważa 2 sposoby łączenia tabel:

- Zapytania zagnieżdżone,
- Złączenia wewnętrzne lub zewnętrzne.

Zapytanie zagnieżdżone opisuje sytuację, kiedy jedno zapytanie znajduje się wewnątrz drugiego. W tym przypadku zapytanie zewnętrzne może korzystać z wyników zapytania wewnętrznego. (GeeksForGeeks, 2024)

Złączenie to natomiast struktura pozwalająca na dodawanie rekordów z wielu tabel bazując na logicznym związku między nimi. (Microsoft, 2023)

Przeanalizowano również wpływ 2 schematów budowy baz i hurtowni danych:

- Schemat gwiazdy,
- Schemat płatka śniegu.

O schemacie gwiazdy mowa w sytuacji, gdy istnieje 1 centralna tabela otoczona przez tabele wymiarów. (IBM, 2021)

Schemat płatka śniegu powstaje w wyniku normalizacji bazy danych o schemacie gwiazdy. Przez to pomiędzy centralną tabelą a tabelami sąsiednimi pojawiają się tabele pośrednie. (Wikipedia, 2024)

Normalizacja

Sięgając do literatury możemy znaleźć konkretne definicje pojęć nawiązujących do procesu normalizacji tabel. (Celko, 1999) Postać normalna dla przykładu definiowana jest jako „sposób klasyfikowania tabel w oparciu o występujące w nich zależności funkcyjne”. Z kolei zależność funkcyjna „oznacza, że znając wartość jednego atrybutu możemy określić wartość innego”. W czasie tworzenia tabel dla niniejszego tekstu otrzymano 3 postać normalną tabeli, która charakteryzuje się kolejnymi cechami:

- nie zawierają powtarzających się grup informacji,
- posiadają klucz określający wszystkie niekluczowe elementy tabeli,
- ich wszystkie niekluczowe kolumny są określane kluczem, całym kluczem i tylko kluczem.

Konfiguracja sprzętowa

W celu odbycia testów uruchomiono załączone programy na komputerze przenośnym wyposażonym w nast. podzespoły i oprogramowanie istotne dla ww. testów:

- Procesor 11 gen. Intel(R) Core (TM) i5-1155G7 @ 2.50 GHz,
- System operacyjny Windows 11 64bit,
- Pamięć RAM o pojemności 8 GB,
- Dysk SSD o pojemności 512 GB,
- Microsoft SQL Server 2022 w wersji 16,

- PostgreSQL w wersji 16.

Testy wydajności

Do przeprowadzenia eksperymentu przyjęto schemat tabeli geochronologicznej. (Bartuś, b.d.) W niniejszym tekście ma ona jednak schemat wyłącznie poglądowy i została wykorzystana ze względu na swoją specyficzną głęboko zagnieżdżoną budowę. Schemat znormalizowany zadeklarowano z użyciem nast. poleceń języka SQL:

```sql
CREATE TABLE GeoEon (
    id_eon INT PRIMARY KEY,
    nazwa_eon VARCHAR(255)
);
CREATE TABLE GeoEra (
    id_era INT PRIMARY KEY,
    id_eon INT FOREIGN KEY REFERENCES GeoEon(id_eon),
    nazwa_era VARCHAR(255)
);
CREATE TABLE GeoOkres (
    id_okres INT PRIMARY KEY ,
    id_era INT FOREIGN KEY REFERENCES GeoEra(id_era),
    nazwa_okres VARCHAR(255)
);
CREATE TABLE GeoEpoka (
    id_epoka INT PRIMARY KEY,
    id_okres INT FOREIGN KEY REFERENCES GeoOkres(id_okres),
    nazwa_epoka VARCHAR(255)
);
CREATE TABLE GeoPietro (
    id_pietro INT PRIMARY KEY,
    id_epoka INT FOREIGN KEY REFERENCES GeoEpoka(id_epoka),
    nazwa_pietro VARCHAR(255)
);
```

Blok kodu 1

Powyższe zostały wypełnione używając analogicznych zapytań do załączonego:

```sql
INSERT INTO GeoEra (id_era, nazwa_era, id_eon)
VALUES (3, 'Kenozoik', 1);
INSERT INTO GeoEra (id_era, nazwa_era, id_eon)
VALUES (2, 'Mezozoik', 1);
INSERT INTO GeoEra (id_era, nazwa_era, id_eon)
VALUES (1, 'Paleozoik', 1);
```

Blok kodu 2

Do utworzenia schematu zdenormalizowanego posłużono się daną deklaracją tabeli:

```sql
CREATE TABLE GeoTabela (
    id_pietro INT PRIMARY KEY,
    nazwa_pietro VARCHAR(255),
    id_epoka INT,
    nazwa_epoka VARCHAR(255),
    id_okres INT,
    nazwa_okres VARCHAR(255),
    id_era INT,
    nazwa_era VARCHAR(255),
    id_eon INT,
    nazwa_eon VARCHAR(255)
);
```

Blok kodu 3

Wypełniono ją posługując się złączeniem odpowiednich kolumn z wcześniej zadeklarowanych tabel schematu znormalizowanego.

```sql
INSERT INTO GeoTabela (
    id_pietro, nazwa_pietro, id_epoka, nazwa_epoka, id_okres,
    nazwa_okres, id_era, nazwa_era, id_eon, nazwa_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 GeoEpoka
    INNER JOIN GeoPietro 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;
```

Blok kodu 4

Utworzono też inną przeznaczoną do testów syntetyczną tabelę wypełnioną kolejnymi liczbami naturalnymi z zakresu (0; 999999).

```sql
CREATE TABLE Milion (
    cyfra INT,
    liczba INT,
    bit_wart INT
);
DECLARE @i INT = 0;
WHILE @i < 1000000
BEGIN
   INSERT INTO Milion (cyfra, liczba, bit_wart)
   VALUES (@i % 10, @i, @i % 10)
   SET @i = @i + 1;
END;
```

Blok kodu 5

Zapytania do bazy danych

Eksperyment polegał na wykonaniu szeregu zapytań bazodanowych. Wszystkie z nich wykonano najpierw na bazach o zindeksowanej jedynie kolumnie kluczowej. Później dodano indeksy dla wszystkich kolumn biorących udział w zapytaniu i uruchomiono polecenia ponownie. Sprawdzenie istnienia indeksów w danej tabeli było możliwie dzięki wykonaniu podanych komend.

```sql
-- SQL Server
EXEC sp_helpindex 'GeoTabela';
-- PostgreSQL
SELECT * FROM pg_indexes
WHERE tablename = 'GeoTabela'
```

Blok kodu 6

Do przetestowania najlepszych osiągów czasowych zapytań posłużono się 4 poleceniami. Pierwsze z nich polegało na złączaniu poszczególnych kolumn tabeli syntetycznej oraz tabeli zdenormalizowanej.

```
SELECT COUNT(*) FROM Milion
    INNER JOIN GeoTabela
ON (Milion.liczba % 54) = GeoTabela.id_pietro;
```

Blok kodu 7

Drugie dot. łączenia tabeli znormalizowanej z tabelą syntetyczną poprzez złączenia wewnętrzne.

```
SELECT COUNT(*) FROM Milion
    INNER JOIN GeoPietro
 ON (Milion.liczba % 54) = 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;
```

Blok kodu 8

Trzecie zapytanie przedstawia operację łączenia tabeli zdenormalizowanej z tabelą syntetyczną poprzez zapytanie zagnieżdżone.

```
SELECT COUNT(*) FROM Milion
WHERE (Milion.liczba % 54) = (
    SELECT id_pietro FROM GeoTabela
    WHERE (Milion.liczba % 54) = id_pietro
);
```

Blok kodu 9

Celem zapytania czwartego było złączenie tabeli syntetycznej z tabelą znormalizowaną poprzez zagnieżdżenie skorelowane a w jego wnętrzu złączenie wewnętrzne poszczególnych tabel.

```sql
SELECT COUNT(*) FROM Milion
WHERE (Milion.liczba % 54) IN (
    SELECT GeoPietro.id_pietro
    FROM GeoPietro
        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
);
```

Blok kodu 10

Uwagi wstępne

W kolejnej części sprawozdanie zamieszczone zostaną tabele zawierające czasy wykonania zapytań. Dla każdego przypadku wykonano 5 pomiarów z czego odrzucono maksymalny i minimalny wynik. Stąd w każdej tabeli pozostawione puste miejsca.

SZBD SQL Server

Pomiar czasu w SZBD SQL Server został zrealizowany poprzez wykonanie niniejszego zapytania zwracającego m.in. czas realizacji komendy poprzez procesor.

```sql
SET STATISTICS TIME ON;
```

Blok kodu 11

Część I

Jak wspomniano we wstępnej części badania nie dodawano dodatkowych indeksów do tabel. Tabela GeoTabela oraz inne powstałe na skutek jej znormalizowania zawierały jedynie jeden klastrowany, unikalny indeks klucza głównego. Syntetyczna tabela nie zawierała zaś żadnego indeksu. Fragmentacja obu tabel wynosiła wtedy 0%.

Dane dot. fragmentacji uzyskano uruchamiając dane lub analogiczne do niego polecenie:

```
SELECT index_id, avg_fragmentation_in_percent
FROM sys.dm_db_index_physical_stats (DB_ID(N'Geo'),
    OBJECT_ID(N'Milion'), NULL, NULL, 'DETAILED'
);
```

Blok kodu 12

Pomierzone wartości zamieszczone w tabeli odpowiadają ms.

Tabela 1

	Próba 1	Próba 2	Próba 3	Próba 4	Próba 5	Średnia	Odch. std.	Błąd wzgl.
Kw. 1	220	140	169	-	-	176	41	23%
Kw. 2	110	76	108	-	-	98	19	19%
Kw. 3	125	140	107	-	-	124	17	13%
Kw. 4	-	92	93	-	125	103	19	18%

Największy procentowo błąd względny, który odnotowano podczas pomiarów wynosi 23% i jest o 10 pp. wyższy od najniższego. Jest to poziom akceptowalny dla tego badania jednak w przypadku dalszej (potrzebnej wg autora) analizy należałoby wykonać większą liczbę pomiarów oraz dążyć do minimalizacji błędu względnego.

Część II

W drugiej części zadania dodano indeksy na dotychczas nieindeksowane kolumny we wszystkich badanych tabelach. Posłużono się nast. komendą a także analogicznymi dla poszczególnych tabel.

```
CREATE NONCLUSTERED INDEX Eon_Ind
ON GeoEon(id_eon, nazwa_eon);
```

Blok kodu 13

Uzyskane wyniki są nast.:

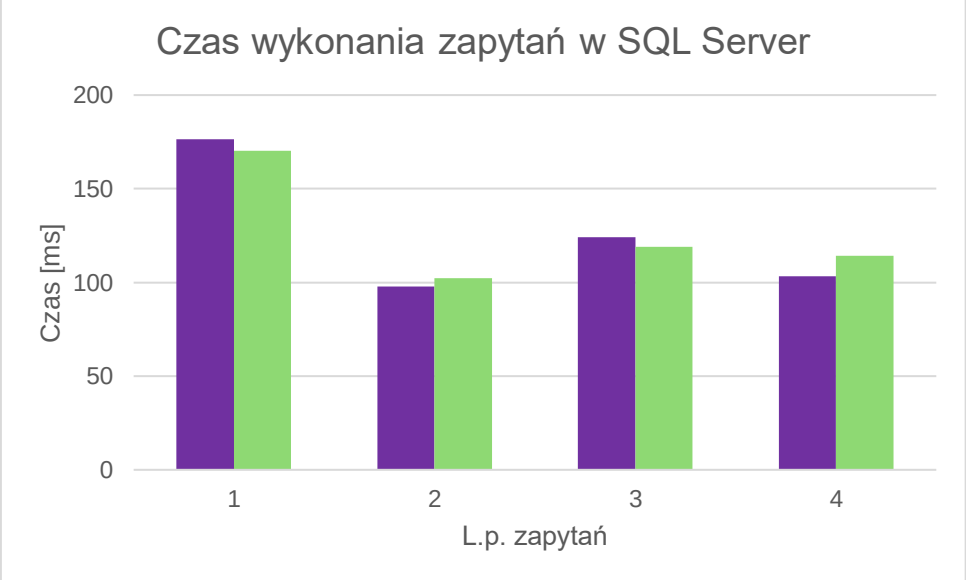
Tabela 2

	Próba 1	Próba 2	Próba 3	Próba 4	Próba 5	Średnia	Odch. std.	Błąd wzgl.
Kw. 1	170	186	-	155	-	170	16	9%
Kw. 2	107	122	78	-	-	102	22	22%
Kw. 3	109	-	-	125	123	119	9	7%
Kw. 4	124	-	125	-	94	114	18	15%

W przypadku wykonania zapytania na indeksowanych tabelach odnotowane błędy są mniejsze zarówno w wartości bezwzględnej jak i procentowej. Największy błąd względny wynosi co prawda 22%. Natomiast jest on i 15 pp. większy od najmniejszego błędu co świadczy o większej precyzji pomiarów w bieżącym przypadku.

Podsumowanie

Na wykresie umieszczono średnie międzyczasy dla każdego zapytania porównując wykonywanie ich na tabelach nieindeksowanych (co zaznaczono kolorem fioletowym) oraz na tabelach indeksowanych (co zaznaczono kolorem zielonym). Wartości te pokrywają się z kolumnami zaznaczonymi na szaro we wszystkich tabelach. Wartości na wykresie naniesiono w skali liniowej.



Wykres 1

Czas wykonywania zapytania pierwszego zdecydowanie odbiega od pozostałych, które wahają się w okolicy 100 ms. W 2 przypadkach stwierdzono zmniejszenie czasu wykonywania zapytania przy zastosowaniu indeksów nieklastrowanych natomiast różnice na korzyść tej techniki są mniejsze niż zaobserwowane odchylenie standardowe. Z tego powodu nie można określić czy zaobserwowane zjawisko rzeczywiście zachodzi czy jest ono jedynie dziełem przypadku.

SZBD PostgreSQL

W celu zachowania kompatybilności należało zmodyfikować niektóre z poleceń SQL. Przykładowo w inny sposób dodano w SZBD PostgreSQL klucze obce tabel:

```
CREATE TABLE GeoEpoka (
    id_epoka INT PRIMARY KEY,
    id_okres INT,
    FOREIGN key(id_okres) REFERENCES GeoOkres(id_okres),
    nazwa_epoka VARCHAR(255)
);
```

Blok kodu 14

Implementacja pomiaru czasu wykonywania zapytań również musiała ulec zmianie. SZBD PostgreSQL obsługuje tę funkcję poprzez dedykowane polecenie modyfikujące zapytanie. W poniższym przykładzie dla 1 analizowanej kwerendy.

```
EXPLAIN ANALYZE
SELECT COUNT(*) FROM Milion
    INNER JOIN GeoTabela
ON (Milion.liczba % 54) = GeoTabela.id_pietro;
```

Blok kodu 15

Część I

Wykonując zapytania na tabelach posiadających jedynie indeksy na elementach kluczowych uzyskano niniejsze wyniki.

Tabela 3

	Próba 1	Próba 2	Próba 3	Próba 4	Próba 5	Średnia	Odch. std.	Błąd wzgl.
Kw. 1	-	-	125	126	130	127	3	2%
Kw. 2	364	-	338	334	-	345	16	5%
Kw. 3	7613	-	7593	-	7603	7603	10	0%
Kw. 4	-	114	115	-	118	116	2	2%

W SZBD PostgreSQL zaobserwowano bardzo małe błędy względne i odchylenia standardowe czasów wykonywanych zapytań. Świadczyć to może o przewidywalności wspomnianego SZBD.

Część II

W SZBD PostgreSQL indeksy nieklastrowane stanowią domyślne rozwiązanie dla tworzenia indeksów w przypadku kolumn niekluczowych, dlatego wykorzystano tworzące je zapytanie o nast. (i analogicznej) formie:

```
1 CREATE INDEX Pietro_Ind
2 ON GeoPietro(id_pietro, id_epoka, nazwa_pietro);
```

Blok kodu 16

Wykonanie wspomnianych zapytań na tabeli posiadającej indeksy na wszystkich elementach niekluczowych przyniosło niniejsze rezultaty:

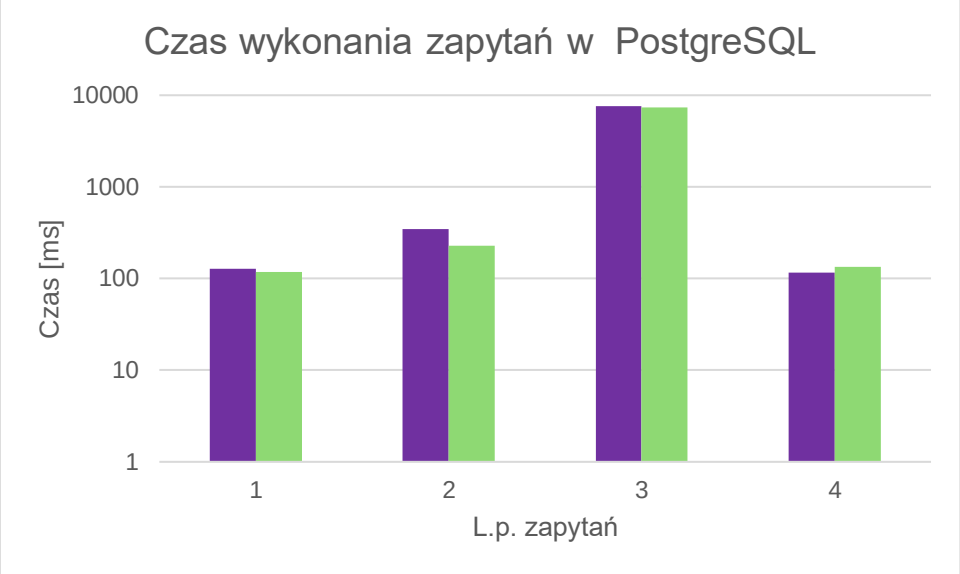
Tabela 4

	Próba 1	Próba 2	Próba 3	Próba 4	Próba 5	Średnia	Odch. std.	Błąd wzgl.
Kw. 1	-	120	119	-	115	118	3	2%
Kw. 2	-	215	233	241	-	230	13	6%
Kw. 3	-	7301	7391	7491	-	7394	95	1%
Kw. 4	140	-	129	132	-	134	6	4%

Wyniki w powyższej oraz wcześniejszej tabeli nie odbiegają znacząco od siebie. Cechują się podobnymi wartościami pomierzonymi oraz podobną dokładnością międzyczasów.

Podsumowanie

Załączony wykres zestawia średnie międzyczasy wykonywania kolejnych zapytań w tabelach nieindeksowanych oraz indeksowanych. Symbolizacja wykresu odpowiada tej dla SZBD SQL Server.



Wykres 2

Ze względu na zdecydowanie dłuższy czas wykonywania zapytania nr 3 w powyższym wykresie zastosowano skalę logarytmiczną. Wartości dla zapytań nr 1, nr 2 i nr 3 wahają się ok. 100 ms natomiast zapytanie nr 3 wykonuje się średnio przed ok. 7,4 s, czyli o rząd wielkości wolniej. W tym przypadku można stwierdzić, że dodanie indeksów do tabel wpłynęło pozytywnie na czas wykonywania zapytań nr 1, nr 2 i nr 3. Efektu tego nie zaobserwowano w przypadku zapytania 4.

Interpretacja i analiza wyników

W przypadku dużych tabel i tabel zdenormalizowanych, co pokazują wyniki dla zapytania nr 1, dodanie indeksów do tabeli niezależnie od badanego SZBD powoduje przyspieszenie wykonania zapytania o kilka procent. Analogiczny rezultat zaobserwowano dla zapytania nr 3, które wykonało się jednak w zdecydowanie dłuższym czasie w PostgreSQL.

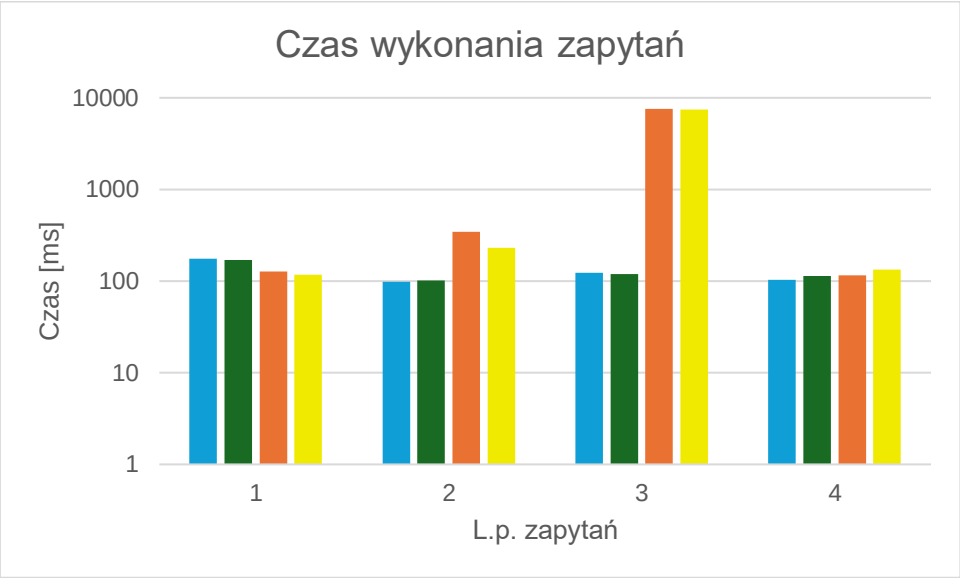
Zapytanie nr 3 polegało na łączeniu tabeli zdenormalizowanej z tabelą syntetyczną poprzez zapytanie zagnieżdżone. Wykonało się ono dokładnie ponad 60 razy wolniej w PostgreSQL. Może to świadczyć o skrajnym nieprzystosowaniu tego SZBD do wykonywania zapytań zagnieżdżonych. Przyspieszenie związane z nałożeniem indeksów na tabelę nie zrekompensowało różnicy czasu pomiędzy SQL Server a PostgreSQL.

W zapytaniu nr 4 łączono tabelę syntetyczną z tabelą znormalizowaną poprzez zagnieżdżenie skorelowane a w jego wnętrzu złączenie wewnętrzne poszczególnych tabel. Zapytanie to wykonywało się dłużej po nałożeniu indeksu na tabele w obu badanych SZBD. Spowolnienie wynosiło od kilku do kilkunastu procent.

Ze względu na konieczność użycia zarówno złączeń wewnętrznych jak i operatora IN oraz ze względu na osiągnięcia czasowe zapytanie nie jest rekomendowane przez autora do wykorzystywania w codziennej pracy.

Najwięcej wątpliwości dostarczają wyniki zapytania nr 2, które wykazuje pozytywny skutek obecności indeksów w PostgreSQL a negatywny w SQL Server. Ze względu na mniejszą niż błąd różnicę wykonania tego zapytania w SQL Server nie można stwierdzić jednoznacznie spowolnienia wykonania zapytania. Ostrożnie i podkreślając konieczność wykonania dalszych testów można założyć, że indeksy w przypadkach obydwu SZBD powodują przyspieszenie wykonania zapytań.

Poniżej umieszczono zestawienie porównawcze wyników z SZBD SQL Server i PostgreSQL.



Wykres 3

Kolorem niebieskim zaznaczono zapytanie w SQL Server bez nałożonych indeksów, kolorem zielonym z nałożonymi indeksami, kolorem pomarańczowym zapytanie bez nałożonych indeksów w PostgreSQL a kolorem żółty zapytanie z nałożonymi indeksami w PostgreSQL.

W większości przypadków zapytania wykonywane w SQL Server zajmowały mniej czasu. Zapytanie nr 2, nr 3 i nr 4 krócej wykonywały się w tym SZBD. Jedynie zapytanie nr 1 poskutkowało odwrotnym rezultatem. Dla zapytania nr 2 i nr 3 różnica była drastyczna i wyraźnie widoczna na wykresie.

Bazując na wyżej opisanym eksperymencie można postawić tezę, że SQL Server jest SZBD pozwalającym na szybsze wykonywanie zapytań do bazy danych. Jednak ze względu na ograniczoną liczbę przetestowanych zapytań oraz niereprezentatywną liczbę uruchomień nie można jednoznacznie stwierdzić czy wyraźnie zaobserwowane zjawisko ma potwierdzenie w rzeczywistości.

Bibliografia

GeeksForGeeks (2024) *Nested Queries in SQL*, https://www.geeksforgeeks.org/nested-queries-in-sql/, dostęp: 02/06/2024

Microsoft (2023) *Joins (SQL Server)*, https://learn.microsoft.com/en-us/sql/relational-databases/performance/joins, dostęp: 02/06/2024

IBM (2021) *Star schemas*, https://www.ibm.com/docs/en/ida/9.1.2?topic=schemas-star, dostęp: 11/06/2024

Wikipedia. The Free Encyclopedia (2024) *Snowflake scheme*, https://en.wikipedia.org/wiki/Snowflake_schema, dostęp: 09/06/2024

Tomasz Bartuś (b. d.) *Tabela geochronologiczna*, https://home.agh.edu.pl/~bartus/index.php?action=dydaktyka&subaction=geologia&item=tabela_geochronologiczna, dostęp: 06/06/2024

Joe Celko, red. Justyna Domasłowska-Szulc (1999) *SQL Zaawansowane techniki programowania*, Warszawa, Wydawnictwo „MIKOM"

Praca na podstawie:

Łukasz Jajeśnica, Adam Piórkowski (2010) *Wydajność złączeń i zagnieżdżeń dla schematów znormalizowanych i zdenormalizowanych*, Kraków, Studia Informatica wyd. 31 nr 89