MAGDALENA PAJOR GR. III NR INDEKSU 406189

WPROWADZENIE.

Niniejszy projekt został opracowany na podstawie artykułu pt. „Wydajność złączeń i zagnieżdżeń dla schematów znormalizowanych i zdenormalizowanych" mr inż. Łukasza Jajeśnica pod nadzorem dr hab. inż. Adama Piórkowskiego z Akademii Górniczo-Hutniczej im. Stanisława Staszica w Krakowie.

Normalizacja to bezstratny proces organizowania danych w tabelach mający na celu zmniejszenie ilości danych składowanych w bazie oraz wyeliminowanie potencjalnych anomalii aktualizacji, wstawiania lub usuwania danych. Czasem przez to spowalniamy wykonywanie zapytań – cena jaką płacimy za konieczność łączenia tabel, ale też przyspieszamy wykonywanie określonych zapytań – tworzymy osobne tabele więc możemy utworzyć więcej indeksów klastrowych, lepsza efektywność przechowywania danych w tabelach.

Denormalizacja jest to natomiast celowe złamanie reguły normalizacji za pomocą celowej redundacji danych, kosztem wielkości bazy danych i zwiększonego ryzyka związanego z możliwością utraty jej spójności. Głównym celem tego jest możliwość przyśpieszenia zwracania odpowiedzi kilka razy szybciej.

Zaprojektowanie wydajnej bazy danych wymaga więc czasem kompromisu między szybkim dostępem a dublowaniem danych. Zdublowane dane wymagają bardziej złożonych algorytmów zachowania integralności, a to powoduje większą złożoność programów.

Poniższy artykuł ma na celu pokazanie wydajności schematów znormalizowanych i zdenormalizowanych oraz wpływ indeksów na szybkość działania zapytań w poszczególnych systemach bazodanowych. W tym celu przetestowano dwa najpopularniejsze systemy zarządzania bazami danych: SQL Server i PostgreSQL. Projekt został przeprowadzony w dwóch etapach (zindeksowanej i niezindeksowanej), aby pokazać jak budowa optymalnych indeksów może o rząd wielkości zmienić czas wykonania zapytania.

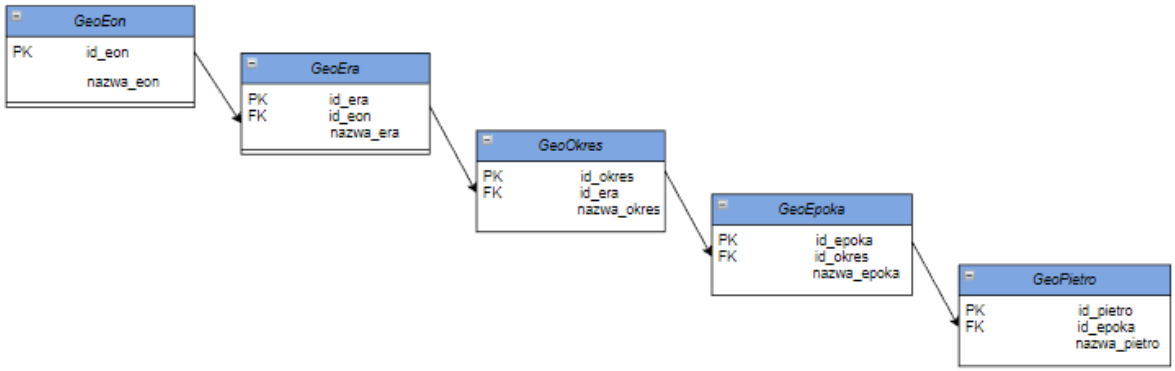
DANE.

Do przeprowadzenia testów posłużyła tabela geochronologiczna. Tabela ta to graficzne przedstawienie podziału geologicznych dziejów Ziemi, od momentu jej powstania 4,6 mld lat temu aż do czasów dzisiejszych.
Tabela zawiera jednostki czasowe (geochronologiczne) wydzielone na podstawie badań geologicznych i paleontologicznych. Granice pomiędzy poszczególnymi jednostkami są czysto umowne i nie muszą być związane z konkretnymi zdarzeniami z historii Ziemi.

Jednostki stratygraficzne (geochronologiczne) są zagnieżdżone, to znaczy dzielą się na podjednostki niższego rzędu. W ten sposób powstaje charakterystyczna struktura tabeli stratygraficznej. Największe jednostki geochronologiczne, czyli eony, dzielą się kolejno na ery, okresy, epoki i wieki.

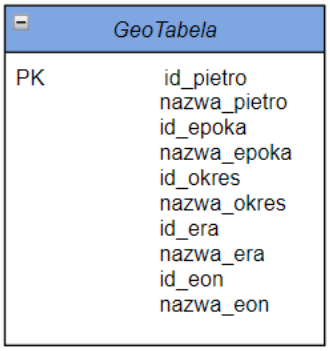
RYS. 1. TABELA GEOCHRONOLOGICZNA

Dane z tabeli geochronologicznej zostały przedstawione w schemacie płatka śniegu, czyli zostały znormalizowane i zaprojektowane zgodnie z modelem relacyjnej bazy danych. Najczęściej jest on używany wtedy gdy rozmiary tabeli są duże. Schemat ten został przedstawiony na poniższym rysunku.



RYS. 2. ZNORMALIZOWANY SCHEMAT TABELI GEOCHRONOLOGICZNEJ

Dla porównania dane również zostały przedstawione w postaci zdenormalizowanej czyli w schemacie gwiazdy. Jest to najprostszy i najbardziej efektywny schemat w hurtowni danych. Tabela ta jest zaprojektowana zupełnie inaczej niż w schemacie płatka śniegu. Jej denormalizacja jest celowa i ma ona za zadanie zwiększenie wydajności i responsywności zapytań kierowanych do bazy.



RYS. 3. ZDENORMALIZOWANY SCHEMAT TABELI GEOCHRONOLOGICZNEJ

Do przeprowadzenia testów wydajności posłużono się również tabelą „Milion” , która została wypełniona kolejnymi liczbami naturalnymi w zakresie od 0 do 999 999. Utworzono ją w poprzez autozłączenie tabeli „Dziesięć”, w której znajdowały się liczby naturalne od 0 do 9.

KOD W SQL SERVER.

TWORZYMY NOWĄ BAZE DANYCH TABELA GEOCHRONOLOGICZNA:

```
CREATE DATABASE tabela_geochronologiczna
```

STWORZENIE TABEL NA PODSTAWIE TABELI GEOCHRONOLOGICZNEJ, WYPEŁNIENIE DANYMI I POWIĄZANIE ICH KLUCZAMI GŁÓWNYMI:

```
CREATE TABLE GeoEon
(
id_eon INT NOT NULL PRIMARY KEY,
nazwa_eon VARCHAR(12) NOT NULL
);

INSERT INTO GeoEon VALUES (1, 'Fanerozoik');

CREATE TABLE GeoEra
(
id_era INT PRIMARY KEY NOT NULL,
id_eon INT FOREIGN KEY REFERENCES GeoEon(id_eon),
nazwa_era VARCHAR(50) NOT NULL
);

INSERT INTO GeoEra VALUES(1, 1, 'Paleozoik');
INSERT INTO GeoEra VALUES(2, 1, 'Mezozoik');
INSERT INTO GeoEra VALUES(3, 1, 'Kenozoik');

CREATE TABLE GeoOkres
(
id_okres INT PRIMARY KEY NOT NULL,
id_era INT FOREIGN KEY REFERENCES GeoEra(id_era),
nazwa_okres VARCHAR(20) NOT NULL
);

INSERT INTO GeoOkres VALUES(1, 1, 'Dewon');
INSERT INTO GeoOkres VALUES(2, 1, 'Karbon');
INSERT INTO GeoOkres VALUES(3, 1, 'Perm');
INSERT INTO GeoOkres VALUES(4, 2, 'Trias');
INSERT INTO GeoOkres VALUES(5, 2, 'Jura');
INSERT INTO GeoOkres VALUES(6, 2, 'Kreda');
INSERT INTO GeoOkres VALUES(7, 3, 'Trzeciorzed_Paleogen');
INSERT INTO GeoOkres VALUES(8, 3, 'Trzeciorzed_Neogen');
INSERT INTO GeoOkres VALUES(9, 3, 'Czwartorzed');

CREATE TABLE GeoWiek
(
id_wiek INT NOT NULL PRIMARY KEY,
wiek INT NOT NULL
);

INSERT INTO GeoWiek VALUES(1, 395);
INSERT INTO GeoWiek VALUES(2, 345);
INSERT INTO GeoWiek VALUES(3, 280);
INSERT INTO GeoWiek VALUES(4, 230);
INSERT INTO GeoWiek VALUES(5, 195);
INSERT INTO GeoWiek VALUES(6, 140);
INSERT INTO GeoWiek VALUES(7, 65);
INSERT INTO GeoWiek VALUES(8, 22.5);
INSERT INTO GeoWiek VALUES(9, 1.8);
INSERT INTO GeoWiek VALUES(10, 0.010);
```

NASTĘPNIE Z WCZEŚNIEJ STWORZONYCH TABEL DZIĘKI ZŁĄCZENIU NATURALNEMU ZOSTAŁA UTWORZONA TABELA W POSTACI ZDENORMALIZOWANEJ O NAZWIE „GEOTABELA”

```
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
INTO GeoTabela
FROM GeoPietro
JOIN GeoEpoka ON GeoEpoka.id_epoka = GeoPietro.id_epoka
JOIN GeoOkres ON GeoOkres.id_okres = GeoEpoka.id_okres
JOIN GeoEra ON GeoEra.id_era = GeoOkres.id_era
JOIN GeoEon ON GeoEon.id_eon = GeoEra.id_eon;
```

ZMODYFIKOWANIE TABELI POPRZEZ DODANIE KLUCZA GŁÓWNEGO

```
ALTER TABLE GeoTabela
ADD PRIMARY KEY (id_pietro);
```

NASTĘPNIE DO PRZEPOROWADZENIA TESTÓW WYDAJNOŚCI KONIECZNE BYŁO UTWORZENIE RÓWNIEŻ DWÓCH TABEL „DZIESIĘĆ” ORAZ „MILION” ORAZ WYPEŁNIENIE ICH WARTOŚCIAMI

```
CREATE TABLE Dziesiec
(
cyfra INT,
bit int
);
INSERT INTO Dziesiec VALUES(0,0000000);
INSERT INTO Dziesiec VALUES(1,0000001);
INSERT INTO Dziesiec VALUES(2,0000010);
INSERT INTO Dziesiec VALUES(3,0000011);
INSERT INTO Dziesiec VALUES(4,0000100);
INSERT INTO Dziesiec VALUES(5,0000101);
INSERT INTO Dziesiec VALUES(6,0000110);
INSERT INTO Dziesiec VALUES(7,0000111);
INSERT INTO Dziesiec VALUES(8,0001000);
INSERT INTO Dziesiec VALUES(9,0001001);
```

```
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;
```

KONFIGURACJA SPRZĘTOWA I PROGRAMOWA.

CPU: AMD RYZEN 5 4500U WITH RADEON GRAPHICS 2.38 GHZ

RAM: 8,00 GB DDR4 3200MHZ

DYSK: SSD PCIE NVME M.2 512

S.O.: WINDOWS 10 HOME

Jako systemy zarządzania bazami danych wybrano oprogramowania wolno dostępne:

SQL Server, wersja SQL Server Management Studio 15.0.18410.0

PostgreSQL, wersja 14.2-1-windows-x64

KRYTERIA I WYNIKI TESTÓW.

Pierwszym etapem testów było przeprowadzenie szeregu zapytań, w których była sprawdzana wydajność złączeń i zagnieżdżeń w postaci znormalizowanej i zdenormalizowanej bez nałożonych indeksów na atrybuty danych. Drugi etap został przedstawiony natomiast w postaci zindeksowanej. Każdy etap został przeprowadzony w dziesięciu próbach, a wyniki skrajne zostały pominięte. Poniżej w tabelach zostały przedstawione uśrednione wyniki dla każdego systemu zarządzania bazami danych oraz wyłonione ich wartości minimalne i maksymalne ze wszystkich prób.

ZAPYTANIE 1

Celem jest złączenie syntetycznej tablicy miliona wyników z tabelą geochronologiczną w postaci zdenormalizowanej, przy czym do warunku złączenia dodano operacje modulo, dopasowującą zakresy wartości złączanych kolumn:

```
SELECT COUNT(*)
FROM Milion
JOIN GeoTabela ON (Milion.liczba % 62)=(GeoTabela.id_pietro);
```

I ETAP (BEZ INDEKSÓW):

SQL SERVER	1	2	3	4	5	6	7	8	9	10
1ZL	23	32	26	30	29	27	37	30	26	25
ŚREDNIA:	28,5									
MIN:	23									
MAX:	37									

POSTGRESQL	1	2	3	4	5	6	7	8	9	10
1ZL	200	125	167	128	123	132	148	137	198	127
ŚREDNIA:	148,5									
MIN:	123									
MAX:	200									

II ETAP (Z INDEKSAMI):

SQL SERVER	1	2	3	4	5	6	7	8	9	10
1ZL	27	29	23	27	21	19	22	24	25	31
ŚREDNIA:	24,8									
MIN:	19									
MAX:	31									

POSTGRESQL	1	2	3	4	5	6	7	8	9	10
1ZL	120	166	189	153	190	160	145	114	169	186
ŚREDNIA:	159,2									
MIN:	114									
MAX:	190									

ZAPYTANIE 2

Celem jest złączenie syntetycznej tablicy miliona wyników z tabelą geochronologiczną w postaci znormalizowanej, reprezentowaną przez złączenia pięciu tabel:

```
SELECT COUNT(*) FROM Milion
JOIN GeoPietro ON (Milion.liczba % 62)=GeoPietro.id_pietro
JOIN GeoEpoka ON GeoEpoka.id_epoka = GeoPietro.id_epoka
JOIN GeoOkres ON GeoOkres.id_okres = GeoEpoka.id_okres
JOIN GeoEra ON GeoEra.id_era = GeoOkres.id_era
JOIN GeoEon ON GeoEon.id_eon = GeoEra.id_eon;
```

I ETAP (BEZ INDEKSÓW):

SQL SERVER	1	2	3	4	5	6	7	8	9	10
2ZL	39	35	47	43	31	30	36	43	28	36
ŚREDNIA:	36,8									
MIN:	28									
MAX:	47									

POSTGRESQL	1	2	3	4	5	6	7	8	9	10
2ZL	261	227	244	256	248	238	231	234	250	243
ŚREDNIA:	243,2									
MIN:	227									
MAX:	261									

II ETAP (Z INDEKSAMI):

SQL SERVER	1	2	3	4	5	6	7	8	9	10
2ZL	22	32	34	31	23	41	37	21	33	27
ŚREDNIA:	30,1									
MIN:	21									
MAX:	41									

POSTGRESQL	1	2	3	4	5	6	7	8	9	10
2ZL	223	212	191	219	226	164	195	191	208	176
ŚREDNIA:	200,5									
MIN:	164									
MAX:	226									

ZAPYTANIE 3

Celem jest złączenie syntetycznej tablicy miliona wyników z tabelą geochronologiczną w postaci zdenormalizowanej, przy czym złączenie jest wykonywane poprzez zagnieżdżenie skorelowane:

```
SELECT COUNT(*)
FROM Milion
WHERE (Milion.liczba % 62) =
       (SELECT id_pietro
       FROM GeoTabela
       WHERE (Milion.liczba % 62) = (id_pietro)

       );
```

I ETAP (BEZ INDEKSÓW):

SQL SERVER	1	2	3	4	5	6	7	8	9	10
3ZL	31	27	38	24	32	26	22	34	28	33
ŚREDNIA:	29,5									
MIN:	22									
MAX:	38									

POSTGRESQL	1	2	3	4	5	6	7	8	9	10
3ZL	5782	6955	5812	5702	5666	5684	5647	5601	5533	5568
ŚREDNIA:	5795									
MIN:	5533									
MAX:	6955									

II ETAP (Z INDEKSAMI):

SQL SERVER	1	2	3	4	5	6	7	8	9	10
3ZG	26	30	29	21	27	19	31	22	26	24
ŚREDNIA:	25,5									
MIN:	19									
MAX:	31									

POSTGRESQL	1	2	3	4	5	6	7	8	9	10
3ZG	5533	5556	5577	5593	5466	5500	5495	5588	5539	5465
ŚREDNIA:	5531,2									
MIN:	5465									
MAX:	5593									

ZAPYTANIE 4

Celem jest złączenie syntetycznej tablicy miliona wyników z tabelą geochronologiczną w postaci znormalizowanej, przy czym złączenie jest wykonywane poprzez zagnieżdżenie skorelowane, a zapytanie wewnętrzne jest złączeniem tabel poszczególnych jednostek geochronologicznych:

```
SELECT COUNT(*)
FROM Milion
WHERE (Milion.liczba % 62) IN
       (SELECT GeoPietro.id_pietro
       FROM GeoPietro
       JOIN GeoEpoka ON GeoEpoka.id_epoka = GeoPietro.id_epoka
       JOIN GeoOkres ON GeoOkres.id_okres = GeoEpoka.id_okres
       JOIN GeoEra ON GeoEra.id_era = GeoOkres.id_era
       JOIN GeoEon ON GeoEon.id_eon = GeoEra.id_eon

       );
```

I ETAP (BEZ INDEKSÓW):

SQL SERVER	1	2	3	4	5	6	7	8	9	10
4ZL	32	27	31	26	35	43	40	32	38	32
ŚREDNIA:	33,6									
MIN:	26									
MAX:	43									

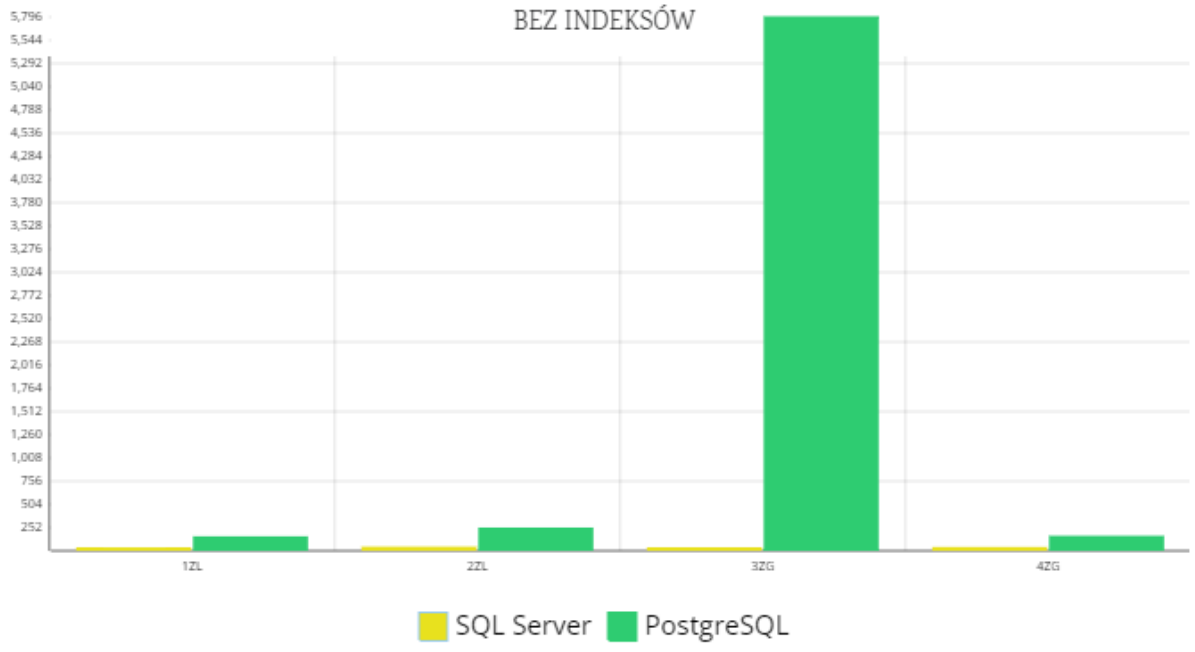
POSTGRESQL	1	2	3	4	5	6	7	8	9	10
4ZG	134	155	176	141	131	157	165	177	167	141
ŚREDNIA:	154,4									
MIN:	131									
MAX:	177									

II ETAP (Z INDEKSAMI):

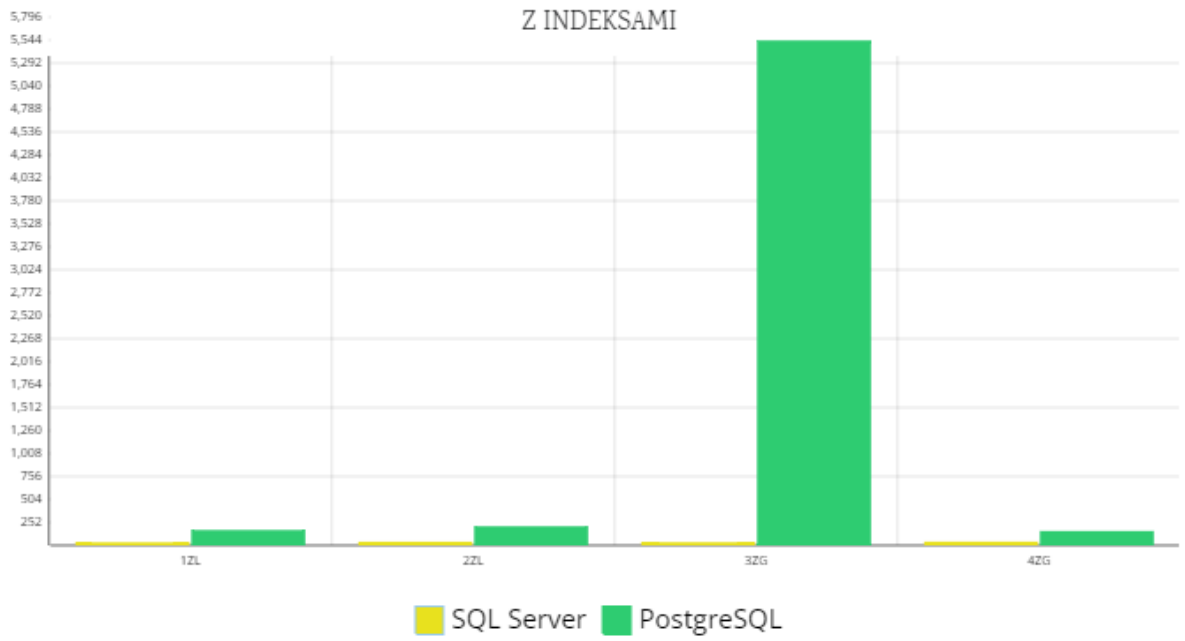
SQL SERVER	1	2	3	4	5	6	7	8	9	10
4ZG	31	27	36	26	35	29	37	29	23	22
ŚREDNIA:	29,5									
MIN:	22									
MAX:	37									

POSTGRESQL	1	2	3	4	5	6	7	8	9	10
4ZG	145	164	143	176	151	129	134	131	177	132
ŚREDNIA:	148,2									
MIN:	129									
MAX:	177									

ANALIZA WYNIKÓW TESTU I WNIOSKI.



RYS. 4 WYNIKI TESTÓW W I ETAPIE W SQL SERVER I W POSTGRESQL



RYS. 5 WYNIKI TESTÓW W II ETAPIE W SQL SERVER I POSTGRESQL

Pierwszy wniosek jaki możemy zauważyć, który najbardziej rzuca nam się w oczy to fakt, że najbardziej wydajnym systemem bazodanowym jest SQL Server, którego czas wykonywania zapytania jest znacząco mniejszy od systemu PostgreSQL. Szczególnie możemy to zauważyć w 3 zapytaniu jego wartości odstają kilkaset razy w postaci zagnieżdżonej (3ZG) w obu etapach w porównaniu z drugim produktem. Nie wolno też zapominać o zaletach PostgreSQL'a, gdzie ceniony jest na rynku światowym za swoją stabilność oraz szeroki zestaw funkcji.

Kolejny wniosek jaki nasuwa się po przeprowadzeniu tego doświadczenia, to spostrzeżenie, że zapytania w postaci zdenormalizowanej w większości przypadków są znacznie szybciej wykonywane. Jednakże wyjątek występuje w 3 zapytaniu, gdzie w PostgreSQL jest znacznie wydłużony czas wykonywania zapytania. Wynika z tego, że normalizacja w większości przypadków prowadzi do spadku wydajności, lecz nie zapominajmy o zaletach jakie niesie ze sobą, a mianowicie eliminacje potencjalnych anomalii aktualizacji, wstawiania lub usuwania danych.

Testy przeprowadzone w tym projekcie przedstawiają także indeksy w pozytywnym świetle. Jednakże w 1 zapytaniu w PostgreSQL nastąpiło kilkuprocentowe spowolnienie działania. W obu produktach we wszystkich pozostałych przypadkach indeksy znacznie przyśpieszyły wykonywanie zapytań, zarówno złączeń, jaki i zagnieżdżeń skorelowanych. Udowadnia to tezę, że budowa optymalnych indeksów może o rząd wielkości zmienić czas wykonania zapytania.

BIBLIOGRAFIA.

- „Wydajność złączeń i zagnieżdżeń dla schematów znormalizowanych i zdenormalizowanych” Łukasz JAJEŚNICA, Adam PIÓRKOWSKI
  Akademia Górniczo – Hutnicza, Katedra Geoinformatyki i Informatyki Stosowanej
- http://stareaneksy.pwn.pl/historia_ziemi/przyklady/?pokaz=tabela
- https://mst.mimuw.edu.pl/lecture.php?lecture=bad&part=Ch15
- Bazy Danych I - dr inż. Michał Lupa, dr hab. inż. Adam Piórkowski, prof AGH