AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE
WYDZIAŁ GEOLOGII, GEOFIZYKI I OCHRONY ŚRODOWISKA

Ćwiczenie 10
Porównanie wydajności złączeń i zagnieżdżeń

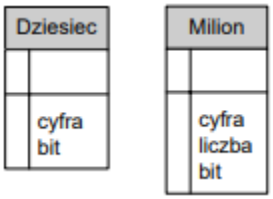
Autor: *Zofia Fabrowska (416440)*
Kierunek studiów: Geoinformatyka

Kraków, 3 czerwca 2024

Spis treści

Rozdział 1. Cel ćwiczenia

Celem ćwiczenia jest zbadanie wydajności złączeń i zagnieżdżeń poprzez odtworzenie eksperymentu naukowego[1]. Postanowiono dokonać analizy poprzez stworzenie odpowiedniej bazy danych oraz użycie indeksów oraz braku ich wykorzystania. Ćwiczenie przeprowadzono używając języka SQL w SQL Server Management Studio oraz PostgreSQL w pgAdmin. Wykorzystano tabelę zawierającą milion wpisów (Wykres 1), tabelę zawierającą dane geochronologiczne w wersji znormalizowanej (Wykres 2Wykres 3) oraz zdenormalizowanej (Wykres 4).

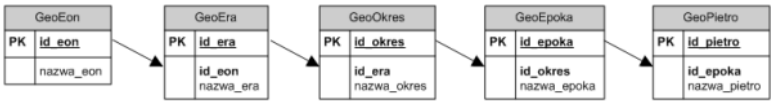


Wykres 1. Schemat tabeli zawierającej milion rekordów (Milion) oraz tabeli pomocniczej (Dziesiec).

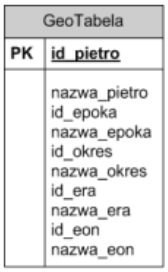
Tabela geochronologiczna

Wiek (mln lat)	Eon	Era	Okres		Epoka
0,010	FANEROZOIK	Kenozoik	Czwartorzęd		Halocen
1,8					Plejstocen
22,5			Trzeciorzęd	Neogen	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

Wykres 2. Schemat tabeli geochronologicznej. Pominięto tu kolumnę „Piętro" ze względu na jej obszerność.



Wykres 3. Schemat znormalizowanej tabeli geochronologicznej.



Wykres 4. Schemat zdenormalizowanej tabeli geochronologicznej

[1] Jajeśnica Ł, Piórkowski A. *Wydajność złączeń i zagnieżdżeń dla schematów znormalizowanych i zdenormalizowanych*. (2010).

Rozdział 2. Konfiguracja sprzętowa

Testy zostały przeprowadzone na komputerze o następujących parametrach:

- CPU: Intel(R) Core(TM) i5-8265U CPU @ 1.60GHz
- RAM: 8,00 GB
- SSD: Transcend 512GB M.2 2242 PCIe NVMe 400S
- Typ systemu: 64-bitowy system operacyjny, procesor x64
- System operacyjny: Windows 10 Home

Jako systemy zarządzania bazami danych wybrano:

- SQL Server Management Studio 20.0.70.0
- pgAdmin 4

Aby upewnić się o wiarygodności uzyskanych wyników, ćwiczenie przeprowadzono pięciokrotnie dla każdego zapytania.

Rozdział 3. Metody

Podrozdział 3.1. Wykonane ćwiczenie w SQL Server Management Studio

Na początku utworzono odpowiednią bazę danych i tabele. W tym celu posłużono się kodem:

```
CREATE DATABASE geochronologia;

USE geochronologia;

CREATE SCHEMA geo;

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
);
```

```
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
);

ALTER TABLE geo.GeoEra ADD FOREIGN KEY (id_eon) REFERENCES geo.GeoEon(id_eon);
ALTER TABLE geo.GeoOkres ADD FOREIGN KEY (id_era) REFERENCES geo.GeoEra(id_era);
ALTER TABLE geo.GeoEpoka ADD FOREIGN KEY (id_okres) REFERENCES geo.GeoOkres(id_okres);
ALTER TABLE geo.GeoPietro ADD FOREIGN KEY (id_epoka) REFERENCES geo.GeoEpoka(id_epoka);

-- wartosci eon
INSERT INTO geo.GeoEon VALUES(1, 'Fanerozoik');
SELECT * FROM geo.GeoEon;

-- wartosci era
INSERT INTO geo.GeoEra VALUES(1, 'Paleozoik', 1);
INSERT INTO geo.GeoEra VALUES(2, 'Mezozoik', 1);
INSERT INTO geo.GeoEra VALUES(3, 'Kenozoik', 1);
SELECT * FROM geo.GeoEra;

-- wartosci okres
INSERT INTO geo.GeoOkres VALUES(1, 'Dewon',1);
INSERT INTO geo.GeoOkres VALUES(2, 'Karbon',1);
INSERT INTO geo.GeoOkres VALUES(3, 'Perm',1);
INSERT INTO geo.GeoOkres VALUES(4, 'Trias',2);
INSERT INTO geo.GeoOkres VALUES(5, 'Jura',2);
INSERT INTO geo.GeoOkres VALUES(6, 'Kreda',2);
INSERT INTO geo.GeoOkres VALUES(7, 'Trzeciorzad - Palogen',3);
INSERT INTO geo.GeoOkres VALUES(8, 'Trzeciorzad - Neogen',3);
INSERT INTO geo.GeoOkres VALUES(9, 'Czwartorzad',3);
SELECT * FROM geo.GeoOkres;
```

W każdej tabeli ustawiono klucz główny oraz dodano klucze obce. Poszczególne tabele wypełniono rekordami zgodnie ze wzorem powyżej (oraz analogicznie dla wartości-epoka, wartości-piętro). Utworzono również tabelę zdenormalizowaną (za pomocą złączeń tabeli znormalizowanej), tabelę pomocniczą *Dziesiec* oraz tabelę *Milion*:

```
--tabela zdenormalizowana
SELECT p.id_pietro, p.nazwa_pietro, ep.id_epoka,
ep.nazwa_epoka, o.id_okres, o.nazwa_okres, er.id_era,
er.nazwa_era, eo.id_eon, eo.nazwa_eon
INTO GeoTabela
FROM geo.GeoPietro p
JOIN geo.GeoEpoka ep ON ep.id_epoka = p.id_epoka
JOIN geo.GeoOkres o ON o.id_okres = ep.id_okres
JOIN geo.GeoEra er ON er.id_era = o.id_era
JOIN geo.GeoEon eo ON eo.id_eon = er.id_eon;
ALTER TABLE GeoTabela ADD PRIMARY KEY (id_pietro);
SELECT * FROM GeoTabela;
```

```
--tabela dziesiec
CREATE TABLE Dziesiec(
    cyfra INT,
    bit INT
);

INSERT INTO Dziesiec VALUES(0,1);
INSERT INTO Dziesiec VALUES(1,1);
INSERT INTO Dziesiec VALUES(2,1);
INSERT INTO Dziesiec VALUES(3,1);
INSERT INTO Dziesiec VALUES(4,1);
INSERT INTO Dziesiec VALUES(5,1);
INSERT INTO Dziesiec VALUES(6,1);
INSERT INTO Dziesiec VALUES(7,1);
INSERT INTO Dziesiec VALUES(8,1);
INSERT INTO Dziesiec VALUES(9,1);
SELECT * FROM Dziesiec;

--tabela milion
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 + 100000*a6.cyfra , a1.cyfra AS cyfra, a1.bit AS bit
FROM Dziesiec a1, Dziesiec a2, Dziesiec a3, Dziesiec a4, Dziesiec a5, Dziesiec a6;
SELECT * FROM Milion;
```

Następnie przeprowadzono testy wydajności. Były to odpowiednio:

- **Zapytanie 1 (1 ZL)** polegające na złączeniu tablicy Milion z tabelą geochronologiczną zdenormalizowaną. Do warunku złączenia dodano operację modulo dopasowującą zakresy wartości złączanych kolumn.

Posłużył do tego kod przedstawiony poniżej. Polecenia ***SET STATISTICS TIME ON*** oraz ***SET STATISTICS TIME OFF*** umożliwiły znalezienie czasu wykonania zapytania (*elapsed time*).

```
-- 1 ZL
SET STATISTICS TIME ON
SELECT COUNT(*)
FROM Milion
INNER JOIN GeoTabela
ON (Milion.liczba % 68 = GeoTabela.id_pietro);
SET STATISTICS TIME OFF
```

- **Zapytanie 2 (2 ZL)** polegające na złączeniu tablicy Milion z tabelą geochronologiczną znormalizowaną.

Zapytanie wykonano posługując się kodem:

```
-- 2 ZL
SET STATISTICS TIME ON
SELECT COUNT(*) FROM Milion
INNER JOIN geo.GeoPietro ON (Milion.liczba % 68=GeoPietro.id_pietro)
INNER JOIN geo.GeoEpoka ON GeoPietro.id_epoka=GeoEpoka.id_epoka
INNER JOIN geo.GeoOkres ON GeoEpoka.id_okres= GeoOkres.id_okres
INNER JOIN geo.GeoEra ON GeoEra.id_era=GeoOkres.id_era
INNER JOIN geo.GeoEon ON GeoEon.id_eon=GeoEra.id_eon
SET STATISTICS TIME OFF
```

- **Zapytanie 3 (3 ZG)** polegające na złączeniu tablicy Milion z tabelą geochronologiczną zdenormalizowaną. Złączenie jest wykonywane poprzez zagnieżdżenie skorelowane.

Zapytanie wykonano posługując się kodem:

```
-- 3 ZG
SET STATISTICS TIME ON
SELECT COUNT(*)
FROM Milion
WHERE Milion.liczba % 68 =
(SELECT id_pietro FROM GeoTabela WHERE Milion.liczba % 68 = id_pietro);
SET STATISTICS TIME OFF
```

- **Zapytanie 4 (4 ZG)** polegające na złączeniu tablicy Milion z tabelą geochronologiczną znormalizowaną. Złączenie jest wykonywane poprzez zagnieżdżenie skorelowane, a zapytanie wewnętrzne jest złączeniem tabel poszczególnych jednostek geochronologicznych.

Zapytanie wykonano posługując się kodem:

```
-- 4 ZG
SET STATISTICS TIME ON
SELECT COUNT(*)
FROM Milion
WHERE Milion.liczba % 68 IN
  (SELECT geo.GeoPietro.id_pietro FROM geo.GeoPietro
   INNER JOIN geo.GeoEpoka ON GeoPietro.id_epoka = GeoEpoka.id_epoka
   INNER JOIN geo.GeoOkres ON GeoEpoka.id_okres = GeoOkres.id_okres
   INNER JOIN geo.GeoEra ON GeoOkres.id_era = GeoEra.id_era
   INNER JOIN geo.GeoEon ON GeoEra.id_eon = GeoEon.id_eon);
SET STATISTICS TIME OFF
```

Każde zapytanie wykonano pięciokrotnie, a następnie znaleziono średni czas obliczeń oraz jego minimalną wartość.

Na koniec wykonano te same testy wydajności, ale po nałożeniu indeksowania na dane:

```
-- indeksy
CREATE INDEX idxEon ON geo.GeoEon(id_eon);
CREATE INDEX idxEra ON geo.GeoEra(id_era, id_eon);
CREATE INDEX idxOkres ON geo.GeoOkres(id_okres, id_era);
CREATE INDEX idxEpoka ON geo.GeoEpoka(id_epoka, id_okres);
CREATE INDEX idxPietro ON geo.GeoPietro(id_pietro, id_epoka);
CREATE INDEX idxLiczba ON Milion(liczba);
CREATE INDEX idxGeoTabela ON GeoTabela(id_pietro, id_epoka, id_era, id_okres,id_eon);
```

Podrozdział 3.2. Wykonane ćwiczenie w PostgreSQL

Ćwiczenie wykonano analogicznie przy użyciu PostgreSQL za pomocą pgAdmin. Początkowo utworzono odpowiednią bazę danych, schemat oraz tabele *GeoEon*, *GeoEra*, *GeoOkres*, *GeoEpoka*, *GeoPietro*. Każdą z tabel wypełniono odpowiednimi wartościami oraz dodano klucze obce posługując się kodem:

```
CREATE DATABASE geochronologia;

CREATE SCHEMA geo;

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
);
```

```
--dodanie kluczy obcych
ALTER TABLE geo.GeoEra ADD FOREIGN KEY (id_eon) REFERENCES geo.GeoEon(id_eon);
ALTER TABLE geo.GeoOkres ADD FOREIGN KEY (id_era) REFERENCES geo.GeoEra(id_era);
ALTER TABLE geo.GeoEpoka ADD FOREIGN KEY (id_okres) REFERENCES geo.GeoOkres(id_okres);
ALTER TABLE geo.GeoPietro ADD FOREIGN KEY (id_epoka) REFERENCES geo.GeoEpoka(id_epoka);

-- uzupelnianie Eonu
INSERT INTO geo.GeoEon VALUES(1, 'Fanerozoik');
SELECT * FROM geo.GeoEon;

-- uzupelnianie er
INSERT INTO geo.GeoEra VALUES(1, 'Paleozoik', 1);
INSERT INTO geo.GeoEra VALUES(2, 'Mezozoik', 1);
INSERT INTO geo.GeoEra VALUES(3, 'Kenozoik', 1);
SELECT * FROM geo.GeoEra;
```

Następnie utworzono zdenormalizowaną łączącą wszystkie powyższe tabele - *GeoTabela*.

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

Utworzono również tabelę pomocniczą *Dziesiec* za pomocą której utworzono tabelę z milionem rekordów *Milion*:

```
-- Tabela Milion
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 + 100000*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;
SELECT * FROM Milion WHERE ;
SELECT COUNT(*) FROM Milion;
```

Następnie wykonano te same zapytania, jak w przypadku SQL Server Management Studio. Dopasowano jednak ich składnię do PostgreSQL:

- **Zapytanie 1 (1 ZL)** polegające na złączeniu tablicy Milion z tabelą geochronologiczną zdenormalizowaną. Do warunku złączenia dodano operację modulo dopasowującą zakresy wartości złączanych kolumn.

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

- **Zapytanie 2 (2 ZL)** polegające na złączeniu tablicy Milion z tabelą geochronologiczną znormalizowaną.

```sql
--2 ZL
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;
```

- **Zapytanie 3 (3 ZG)** polegające na złączeniu tablicy Milion z tabelą geochronologiczną zdenormalizowaną. Złączenie jest wykonywane poprzez zagnieżdżenie skorelowane.

```sql
--3 ZG
SELECT COUNT(*) FROM Milion
    WHERE mod(Milion.liczba,68)=
(SELECT id_pietro FROM GeoTabela WHERE mod(Milion.liczba,68)=(id_pietro));
```

- **Zapytanie 4 (4 ZG)** polegające na złączeniu tablicy Milion z tabelą geochronologiczną znormalizowaną. Złączenie jest wykonywane poprzez zagnieżdżenie skorelowane, a zapytanie wewnętrzne jest złączeniem tabel poszczególnych jednostek geochronologicznych.

```sql
--4 ZG
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);
```

Każde zapytanie wykonano pięciokrotnie, a następnie znaleziono średni czas obliczeń oraz minimalną wartość.

Na koniec wykonano te same testy wydajności, ale po nałożeniu indeksowania na dane:

```sql
--indeksy
CREATE INDEX idxEon ON geo.GeoEon(id_eon);
CREATE INDEX idxEra ON geo.GeoEra(id_era, id_eon);
CREATE INDEX idxOkres ON geo.GeoOkres(id_okres, id_era);
CREATE INDEX idxEpoka ON geo.GeoEpoka(id_epoka, id_okres);
CREATE INDEX idxPietro ON geo.GeoPietro(id_pietro, id_epoka);
CREATE INDEX idxLiczba ON Milion(liczba);
CREATE INDEX idxGeoTabela ON GeoTabela(id_pietro, id_epoka, id_era, id_okres,id_eon);
```

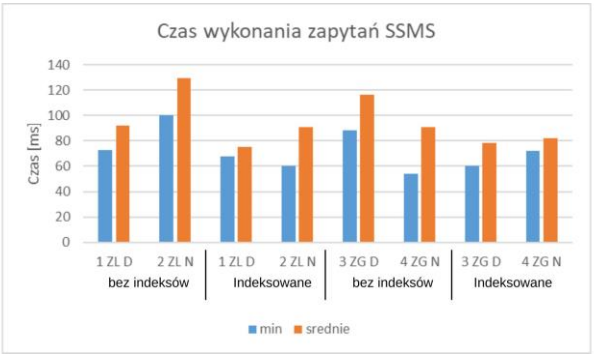
Rozdział 4. Wyniki

Każdy test przeprowadzono pięciokrotnie. Dla otrzymanych wyników wyliczono średnią oraz minimalną wartość, a następnie zestawiono dane w tabeli (Tabela 1).

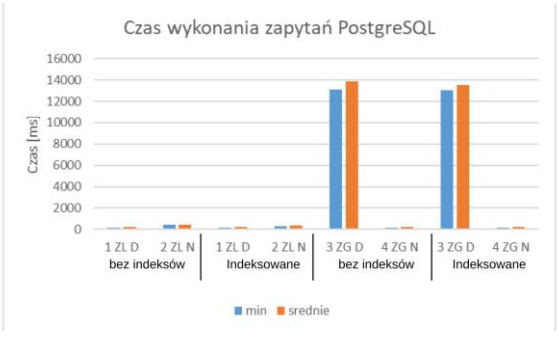
Tabela 1. Czas wykonania zapytań 1 ZL, 2 ZL, 3 ZG, 4 ZG [ms]

	1 ZL		2 ZL		3 ZG		4 ZG	
	min	średni	min	średni	min	średni	min	średni
	brak indeksowania							
SSMS	73	92,2	100	129,2	88	116,6	60	90,6
PostgreSQL	165	221,8	406	423,6	13109	13897,6	163	222,2
	z indeksowaniem							
SSMS	68	75,2	60	90,6	54	78,6	72	82,2
PostreSQL	173	220	297	351,6	13030	13497,6	170	203,4

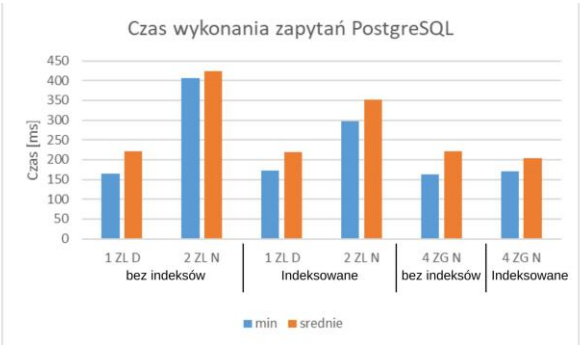
W celu ułatwienia analizy wyników z powyższych danych utworzono wykresy kolumnowe. Podzielono poszczególne zapytania ze względu na fakt indeksowania (brak indeksowania lub z indeksowaniem), normalizację danych (D – dane zdenormalizowane, N – dane znormalizowane) oraz użyty program (Wykres 5Wykres 6). Ze względu na duży rozrzut czasu wykonania zapytań przy użyciu PostgreSQL utworzono jeszcze Wykres 7 dla lepszej wizualizacji danych.



Wykres 5. Czas wykonania zapytań SSMS.



Wykres 6. Czas wykonania zapytań PostgreSQL.



Wykres 7. Czas wykonania zapytań PostgreSQL – z pominięciem 3 ZG.

Rozdział 5. Wnioski

Otrzymane wyniki pozwalają wyciągnąć następujące wnioski:

- Postać zdenormalizowana jest w większości przypadków wydajniejsza od postaci znormalizowanej. Wyjątek stanowi zapytanie używające zagnieżdżenia skorelowanego. Czas wykonania tego zapytania bez indeksów w SSMS był nieznacznie dłuższy dla postaci zdenormalizowanej. Przy użyciu PostgreSQL czas wykonania zapytania z zagnieżdżeniem skorelowanym był znacznie dłuższy dla postaci zdenormalizowanej, Dotyczyło to zarówno wersji z indeksami i bez indeksów.

- Przy realizacji zadania w SSMS zauważono, że czas *elapsed time* był mniejszy od *CPU time*. Jest to nietypowa sytuacja, która może wynikać ze zrównoleglenia na wątki[2].

- Czas realizacji zapytań zawierających złączenia i zagnieżdżenia był podobny w SSMS. W PostgreSQL wszystkie zapytania okazały się dużo szybsze od zapytania z zagnieżdżeniem skorelowanym dla postaci zdenormalizowanej.

- Zastosowanie indeksów przyspieszyło realizację wszystkich zapytań w SSMS.

- W przypadku PostgreSQL zastosowanie indeksów minimalnie przyspieszyło realizację zapytań 1 ZL, 3 ZG, 4 ZG oraz znacznie przyspieszyło realizację zapytania 2 ZL.

- W wykonanym ćwiczeniu wszystkie zapytania zostały szybciej zrealizowane w SSMS.

Uzyskane wnioski odnoszą się jedynie do przeprowadzonego ćwiczenia. Przy realizacji innego zadania lub innej konfiguracji sprzętowej istnieje możliwość osiągnięcia innych rezultatów. Również zwiększenie liczby powtórzeń każdego testu mogłoby zwiększyć dokładność i wiarygodność uzyskanych wyników.

Podsumowując, należy zauważyć, że normalizacja prowadzi w większości przypadków do zmniejszenia wydajności i wydłużenia czasu realizacji zapytania. Nie należy jednak zapominać o zaletach normalizacji – lepszą organizację bazy danych, spadek ryzyka redundancji danych, a także możliwość zwiększenia bezpieczeństwa bazy[3].

[2] *How to read SSMS active expensive query CPU time*. https://learn.microsoft.com/en-us/answers/questions/1314811/how-to-read-ssms-active-expensive-query-cpu-time. Dostęp: 07.06.2024.

[3] *Data Normalization: Definition, Importance, and Advantages*. https://coresignal.com/blog/data-normalization/. Dostęp: 07.06.2024.

Rozdział 6. Literatura

Data Normalization: Definition, Importance, and Advantages. https://coresignal.com/blog/data-normalization/. Dostęp: 07.06.2024.

How to read SSMS active expensive query CPU time. https://learn.microsoft.com/en-us/answers/questions/1314811/how-to-read-ssms-active-expensive-query-cpu-time. Dostęp: 07.06.2024.

Wydajność złączeń i zagnieżdżeń dla schematów znormalizowanych i zdenormalizowanych. Jaję śnica Ł, Piórkowski A. (2010).

Rozdział 7. Spis tabeli i wykresów