KONFERENCJA RADIOKOMUNIKACJI
I TELEINFORMATYKI
KRiT 2024

ANALIZA WPŁYWU NORMALIZACJI BAZ DANYCH NA ICH WYDAJNOŚĆ I INTEGRALNOŚĆ W RELACYJNYCH SYSTEMACH ZARZĄDZANIA BAZAMI DANYCH

ANALYSIS OF THE IMPACT OF DATABASE NORMALIZATION ON ITS PERFORMANCE AND INTEGRITY IN RELATIONAL DATABASE MANAGEMENT SYSTEMS

Aleksander Brachman[1]; Jerzy Domżał[2]; Robert Wójcik[3]

[1] Akademia Górniczo-Hutnicza, Instytut Telekomunikacji, Kraków, email brachman@student.agh.edu.pl
[2] Akademia Górniczo-Hutnicza, Instytut Telekomunikacji, Kraków, email jerzy.domzal@agh.edu.pl
[3] Akademia Górniczo-Hutnicza, Instytut Telekomunikacji, Kraków, email robert.wojcik@agh.edu.pl

Streszczenie: **Proces normalizacji bazy danych pozwala na optymalizację relacyjnej bazy danych poprzez wyeliminowanie z niej redundancji danych, która występuje w bazach zdenormalizowanych. W ramach pracy dokonano normalizacji bazy zdenormalizowanej oraz porównano wydajność bazy zdenormalizowanej i normalizowanej w popularnych systemach zarządzania bazami danych, przy wykorzystaniu operacji CRUD. Zbadano również wpływ normalizacji na zachowanie integralności danych. Przeprowadzone badania wykazują, że oba typy bazy danych charakteryzują się zarówno zaletami, jak i ograniczeniami.**

Abstract: The process of database normalization allows the optimization of a relational database by eliminating the data redundancy that occurs in denormalized databases. As the part of study, the denormalized database was normalized and the performance of the denormalized and normalized database was compared in popular database management systems, using CRUD operations. The impact of normalization on data integrity was also investigated. The conducted research demonstrates that both types of database have their own advantages and limitations.

Słowa kluczowe: **integralność bazy danych, normalizacja bazy danych, system zarządzania bazą danych, wydajność bazy danych**

Keywords: **database integrity, Database Management System, database performance, normalization of database**

1. WSTĘP

Bazy danych od lat są podstawowym komponentem wielu różnych systemów IT. Zarówno małe aplikacje webowe, jak i wielkie sieci korporacyjne potrzebują rozwiązania, które pozwoli na przechowywanie odpowiednich danych, potrzebnych do prawidłowego i zamierzonego funkcjonowania danego systemu IT. Takim rozwiązaniem, od lat wykorzystywanym powszechnie w dziedzinie informatyki, są systemy zarządzania bazami danych (dalej: SZBD). Jednym z typów SZBD są relacyjne systemy zarządzania bazami danych [8], które pozwalają na przetwarzanie w intuicyjny sposób informacji przechowywanych w bazie danych.

Relacyjne bazy danych przechowują dane w uporządkowany sposób, korzystając z tabel [8]. Każda tabela składa się z kolumn (atrybutów) i wierszy (rekordów). Rekord tabeli zawiera informacje nt. określonego, pojedynczego obiektu, którego zbiór przechowywany jest w tabeli. Poszczególne cechy danego obiektu przechowywane są w oddzielnych polach, które odpowiadają zadeklarowanym kolumnom tabeli. Każda kolumna ma również zdefiniowany typ danych, który może przechowywać w swoich polach, np.: tekst o określonej długości, liczba zmiennoprzecinkowa, data i godzina itd. Jeżeli w bazie danych znajduje się więcej niż jedna tabela, wówczas między tabelami mogą zachodzić relacje, które są możliwe dzięki unikalnym kluczom: kluczowi głównemu i kluczowi obcemu. Każda kolumna w tabeli, która przechowuje informacje pozwalające jednoznacznie zidentyfikować każdy rekord w tabeli może zostać kluczem głównym. Klucz główny jednej tabeli może zostać wykorzystany jako klucz obcy w drugiej tabeli, tworząc tym samym relację pomiędzy tabelami i umożliwiając połączenie informacji w nich zawartych.

Normalizacja bazy danych to proces, który pozwala na optymalizację relacyjnej bazy danych poprzez wyeliminowanie z niej nadmiarowości (redundancji) danych [1]. Przykładem normalizacji bazy danych może być rozdzielenie jednej, dużej tabeli, w której zadeklarowane jest wiele kolumn, na kilka mniejszych tabel, w których odpowiednie wartości będą zapisywane tylko raz (w jednym rekordzie), a odniesieniem do nich będą klucze obce, zadeklarowane w innych tabelach. Taki przykład normalizacji bazy danych i analiza wpływu tegoż procesu na wydajność i integralność bazy danych jest przedmiotem badań, których wyniki zostały zaprezentowane w niniejszym artykule.

2. ANALIZA LITERATURY

Termin normalizacji bazy danych po raz pierwszy pojawił się w literaturze w 1970 roku [1]. Codd w swoim artykule opisał założenia procesu normalizacji bazy danych, które obecnie pełnią rolę założeń do pierwszej postaci normalnej bazy danych (1NF). Wg Codd'a baza danych może zostać uznana za znormalizowaną w momencie gdy wszystkie rekordy w tabeli są unikalne - brak występowania duplikatów rekordów - oraz gdy w tabelach nie występują zagnieżdżone tabele - dane w każdym polu

tabeli są przechowywane w postaci atomowej (są niepodzielne). Następne postacie normalne zostały opisane przez Codd'a w późniejszych latach [2], [3].

W ostatnich kilkunastu latach normalizacja bazy danych w relacyjnych SZBD była przedmiotem badań kilku prac naukowych. W [5] analizowano m.in. wpływ kolejnych postaci normalnych baz danych na jej rozmiar zajmowany w poszczególnych relacyjnych SZBD.

Przejście kolejnych kroków z nieznormalizowanej bazy danych do bazy danych w trzeciej postaci normalnej zostało zaprezentowane w [10]. Autorzy artykułu zbadali również czas wykonywania operacji *SELECT* w zależności od postaci normalnej bazy danych.

Proces zdenormalizowania testowej bazy danych, która oryginalnie składała się z wielu tabel powiązanych ze sobą relacjami został przedstawiony w [4]. W ramach procesu denormalizacji autorzy artykułu scalili kolumny i dane z wielu różnych tabel w jedną tabelę i następnie porównali m.in. czasy wykonywania testowych zapytań do bazy normalizowanej i zdenormalizowanej.

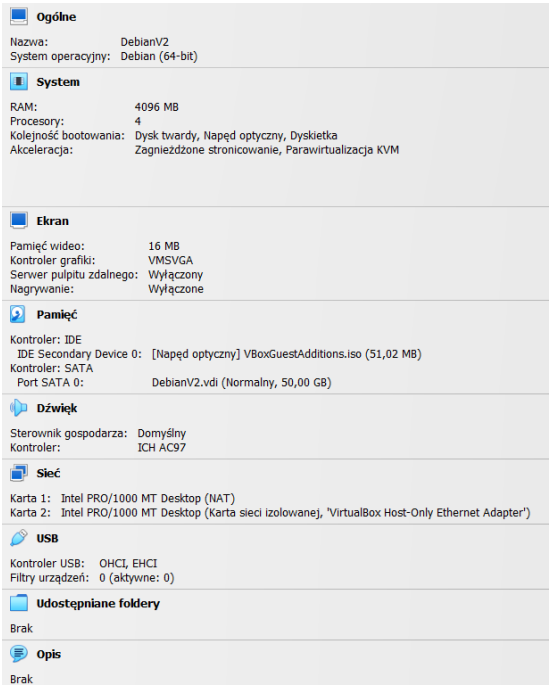
Zastosowanie teorii stojącej za drugą postacią normalną (2NF) w celu zwiększenia wydajności i efektywności zapytań do bazy danych w środowisku OLTP jest przedmiotem rozważań w [7]. W artykule zaproponowano koncepcję złożonego klucza głównego w tabeli, który pozwala na dokładniejsze spełnienie założeń 2NF.

Niniejszy artykuł skupia się na kompleksowym podejściu do analizy wpływu procesu normalizacji baz danych. W ramach pracy zbadano i porównano wydajność operacji *CRUD* pomiędzy bazą zdenormalizowaną i normalizowaną. Badania te dokonano w trzech popularnych SZBD. Dodatkowo przeanalizowano wpływ normalizacji baz danych na zachowanie integralności danych.

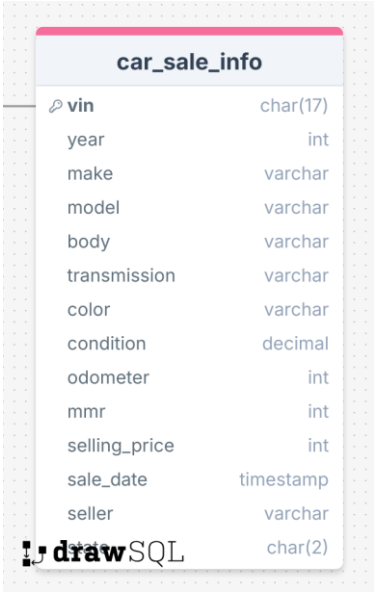
3. ŚRODOWISKO BADAWCZE

Środowisko badawcze wykorzystane w niniejszej pracy składa się z trzech elementów:

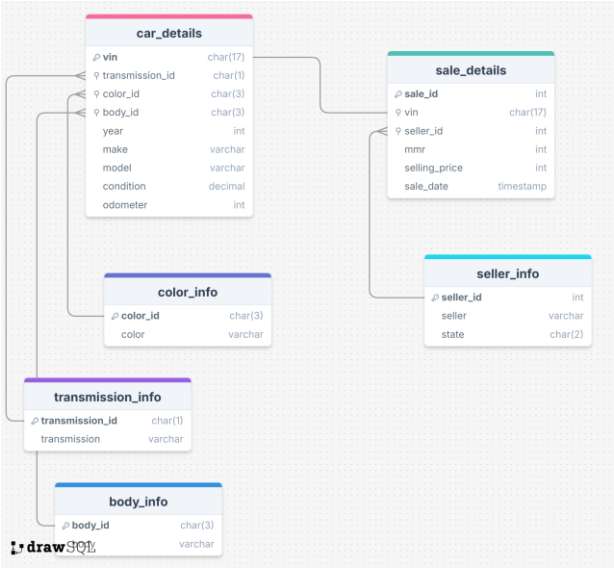
- Maszyna wirtualna: wykorzystano rozwiązanie firmy Oracle VM VirtuaBox w wersji 7.0.10. W programie VirtualBox ustanowiono maszynę wirtualną, na której zainstalowano system operacyjny Debian w wersji 12. Szczegółowe parametry maszyny wirtualnej zostały przedstawione na rys. 1. Komputer-host, na którym uruchomiona została maszyna wirtualna posiada następującą specyfikację:
    - Procesor: AMD Ryzen 5 5625U
    - Pamięć RAM: 16 GB
    - Dysk: SSD 512 GB
    - System operacyjny: Windows 11 w wersji 23H2
- SZBD: do badań wybrano trzy relacyjne systemy zarządzania bazami danych, które są jednymi z najpopularniejszych rozwiązań na rynku: MariaDB w wersji 10.11.6, MySQL w wersji 8.4.3 oraz PostgreSQL w wersji 15.8.0. Wszystkie wybrane SZBD zostały zainstalowane i skonfigurowane na maszynie wirtualnej opisanej powyżej.
- Zdenormalizowana i normalizowana baza danych: w celu utworzenia i wypełnienia baz danych informacjami wykorzystano zbiór historycznych danych (plik .csv) o sprzedawanych samochodach na aukcjach w Stanach Zjednoczonych [9]. Schematy utworzonych baz: zdenormalizowanej i normalizowanej przedstawione zostały na rys. 2 i rys. 3. Ostatecznie, po przefiltrowaniu zbioru danych, w bazach danych przechowywane są informacje z 14 kolumn pliku .csv (bez kolumny *trim* i *interior*) o 393 818 unikalnych samochodach.



Rys. 1. Parametry maszyny wirtualnej.



Rys. 2. Schemat zdenormalizowanej bazy danych (źródło: drawsql.app).



Rys. 3. Schemat normalizowanej bazy danych (źródło: drawsql.app).

4. BADANIA I WYNIKI

4.1. Wydajność

W celu zbadania wpływu normalizacji baz danych na ich wydajność wykonano operacje *CRUD (Create, Read, Update, Delete)* na analizowanych bazach danych. Pierwszy etap badań polegał na kilkukrotnym pomiarze czasu operacji wstawiania danych (*Create* poprzez zapytanie *INSERT INTO*) z pliku .csv do utworzonych już wcześniej baz danych w wybranych SZBD. Uzyskane uśrednione czasy wraz z miarami dyspersji, z podziałem na bazę zdenormalizowaną i normalizowaną, zaprezentowano w tab. 1. i tab. 2.

Tab. 1. Wyniki wykonywania operacji INSERT INTO dla bazy zdenormalizowanej

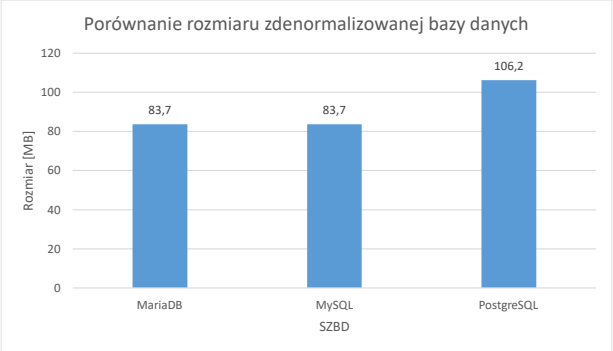
SZBD	*Śr. czas [s]*	*Odch. stand. [s]*	*Rozkład t-Studenta (95%)*
MariaDB	8,68	1,27	1,58
MySQL	12,55	0,91	1,13
PostgreSQL	131,18	4,03	5,01

Tab. 2. Wyniki wykonywania operacji INSERT INTO dla bazy normalizowanej

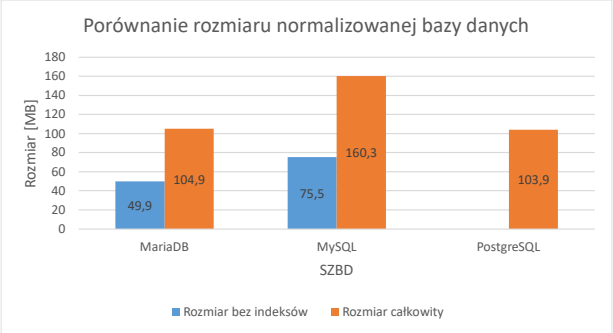
SZBD	*Śr. czas [s]*	*Odch. stand. [s]*	*Rozkład t-Studenta (95%)*
MariaDB	1587,65	22,78	56,5
MySQL	3373,84	98,6	244,94
PostgreSQL	714,34	7,12	17,69

Jak można zauważyć, średni czas wstawiania danych do bazy zdenormalizowanej jest zdecydowanie krótszy aniżeli do bazy normalizowanej, bez względu na używany SZBD. Najprawdopodobniej jest to związane z koniecznością stosowania, w przypadku dodawania danych do bazy normalizowanej, podzapytań *SELECT* w zapytaniu *INSERT INTO* – dla przykładu tabela *sale_details* wymaga do każdego rekordu pobrania wartości *seller_id* z tabeli *seller_info*.

Po wypełnieniu baz danych informacjami porównano rozmiary (w MB) baz danych w wybranych SZBD. Rezultaty przedstawiono na rys. 4 oraz rys. 5.



Rys. 4. Rozmiar zdenormalizowanej bazy danych w badanych SZBD.



Rys. 5. Rozmiar normalizowanej bazy danych w badanych SZBD.

Normalizacja bazy danych spowodowała zwiększenie zajmowanego przez bazę normalizowaną całkowitego rozmiaru pamięci w MariaDB (wzrost o ok. 21,2 MB) i MySQL (wzrost o ok. 76,6 MB) oraz zmniejszenie (spadek o ok. 2,3 MB) w PostgreSQL. Można więc stwierdzić, że MariaDB i MySQL uzyskują lepsze wyniki dla bazy zdenormalizowanej, a PostgreSQL dla bazy normalizowanej, chociaż różnica w całkowitym rozmiarze bazy normalizowanej między MariaDB, a PostgreSQL jest minimalna.

Przy analizie tych wyników warto wziąć pod uwagę różnicę w sposobie obliczania rozmiarów baz danych przez poszczególne SZBD. W przypadku MariaDB i MySQL rozmiar całkowity bazy składa się z rozmiaru samych danych przechowywanych w tabelach (niebieskie kolumny na rys. 5) oraz rozmiaru indeksów, które są związane z istnieniem relacji między tabelami. W przypadku bazy zdenormalizowanej w MariaDB i MySQL rozmiar indeksów wynosił 0 MB (brak relacji między tabelami), a więc na wynik na rys. 4. składa się wyłącznie rozmiar danych przechowywanych w jednej tabeli bazy zdenormalizowanej. Można więc zauważyć, że w MariaDB i MySQL normalizacja bazy danych zmniejszyła rozmiar danych, które przechowywane są w bazie normalizowanej. Jednakże co najistotniejsze ogólny rozmiar bazy normalizowanej w MariaDB i MySQL jest większy niż rozmiar bazy zdenormalizowanej w tych samych SZBD. W przypadku PostgreSQL rozmiary danych i indeksów są obliczane w inny sposób aniżeli w MySQL i MariaDB – w PostgreSQL, dla bazy zdenormalizowanej,

rozmiar indeksów nie był zerowy – toteż, dla PostgreSQL, nie uwzględniono rozmiaru bez indeksów na rys. 5.

W kolejnym etapie badań skupiono się na pomiarze czasów wykonywania zapytań *SELECT* do przygotowanych baz danych (operacja *Read*). W ramach tego etapu przygotowano trzy testowe zapytania SELECT - różne dla bazy zdenormalizowanej (*denorm*) i normalizowanej (*norm*), które zostały zaprezentowane na rys. 6-8.

```
--SELECT 1
-- denorm
SELECT * FROM car_sale_info;
-- norm
SELECT c.vin, c.year, c.make, c.model, b.body,
t.transmission, cl.color, c.car_condition, c.odometer, sa.mmr,
sa.selling_price, sa.sale_date, se.seller, se.state
FROM car_details c
JOIN body_info b ON c.body_id = b.body_id
JOIN transmission_info t ON c.transmission_id = t.transmission_id
JOIN color_info cl ON c.color_id = cl.color_id
JOIN sale_details sa ON c.vin = sa.vin
JOIN seller_info se ON sa.seller_id = se.seller_id;
```

Rys. 6. Zapytanie SELECT 1 – pobranie wszystkich rekordów z bazy.

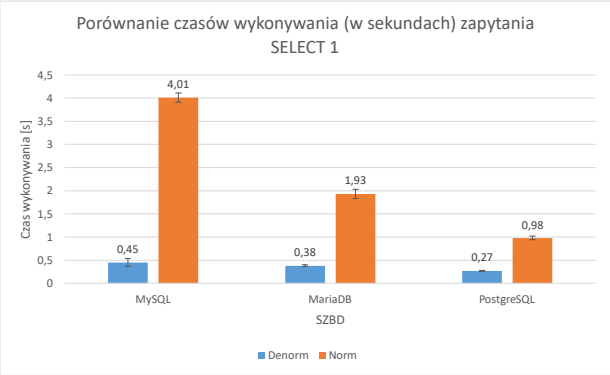
```
--SELECT 2
-- denorm
SELECT make, round(avg(car_condition),2) AS 'avg_car_condition',
round(avg(mmr),2) AS 'avg_mmr'
FROM car_sale_info
GROUP BY make
ORDER BY avg(mmr) DESC;
-- norm
SELECT c.make AS 'make', round(avg(c.car_condition),2) AS 'avg_car_condition',
round(avg(sa.mmr),2) AS 'avg_mmr'
FROM car_details c
JOIN sale_details sa ON c.vin = sa.vin
GROUP BY c.make
ORDER BY avg(sa.mmr) DESC;
```

Rys. 7. Zapytanie SELECT 2 – średnia kondycja samochodu i średnia wartość samochodu z podziałem na markę samochodu.

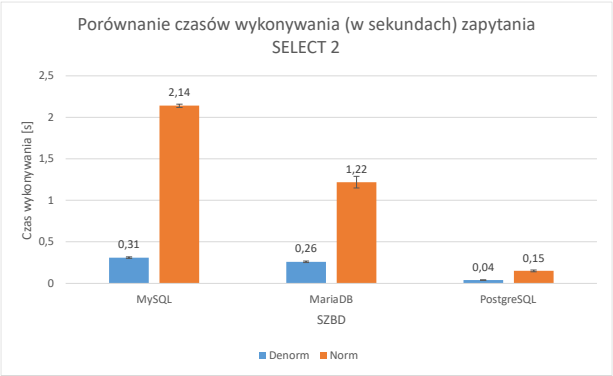
```
--SELECT 3
-- denorm
SELECT state, round(avg(selling_price),2) as "avg_selling_price"
FROM car_sale_info
GROUP BY state
ORDER BY avg(selling_price) DESC;
-- norm
SELECT se.state AS "state", round(avg(sa.selling_price),2) AS "avg_selling_price"
FROM seller_info se
JOIN sale_details sa ON se.seller_id = sa.seller_id
GROUP BY se.state
ORDER BY avg(sa.selling_price) DESC;
```

Rys. 8. Zapytanie SELECT 3 – średnia cena sprzedaży samochodu z podziałem na stany.

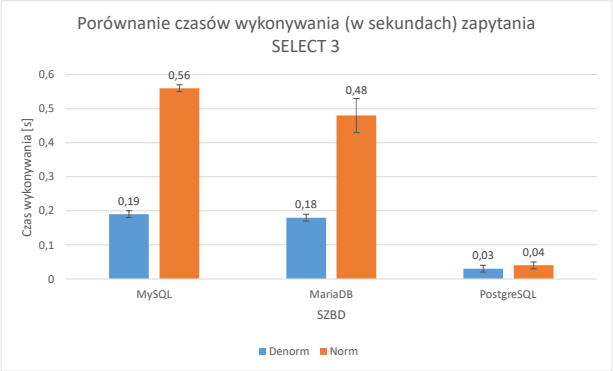
Uśrednione czasy wykonywania zapytań *SELECT 1-3* w badanych SZBD, wraz z błędami wyliczonymi z rozkładu t-Studenta (95%), zaprezentowano na rys. 9-11.



Rys. 9. Uśrednione czasy wykonywania SELECT 1.



Rys. 10. Uśrednione czasy wykonywania SELECT 2.



Rys. 11. Uśrednione czasy wykonywania SELECT 3.

Dla każdego z zapytań *SELECT* najkrótszy czas ich wykonywania, zarówno dla bazy zdenormalizowanej, jak i normalizowanej, uzyskano w PostgreSQL. Czasy wykonywania poszczególnych zapytań *SELECT* w bazie zdenormalizowanej, w różnych SZBD, są do siebie zbliżone. Większe różnice w wynikach (nawet >1 sekundy) występują dla bazy normalizowanej.

Jak można zauważyć, czasy wykonywania zapytań *SELECT* w bazie normalizowanej są dłuższe aniżeli w bazie zdenormalizowanej. Wynika to z konieczności formułowania, w bazie normalizowanej, bardziej rozbudowanych zapytań *SELECT*, które łączą informacje przechowywane w kilku różnych tabelach (klauzula *JOIN*).

Następnie zbadano czas wykonywania dwóch testowych zapytań *UPDATE*, które zostały zawarte na rys. 12.

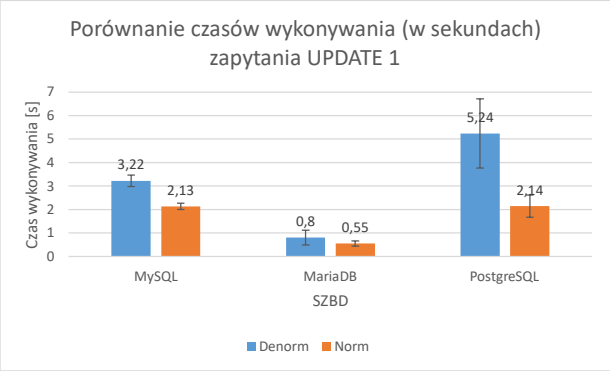
```
-- UPDATE 1
-- denorm
UPDATE car_sale_info SET selling_price = 1.05 * selling_price;
--norm
UPDATE sale_details SET selling_price = 1.05 * selling_price;

-- UPDATE 2
-- denorm
UPDATE car_sale_info SET seller = 'max & leo venters motors inc'
WHERE seller = 'leo venters motors inc' AND state = 'nc';
-- norm
UPDATE seller_info SET seller='leo venters motors inc'
WHERE seller_id=5793;
```

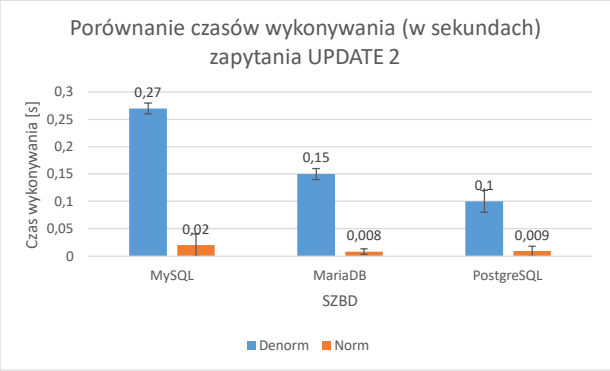
Rys. 12. Zapytanie UPDATE 1 – aktualizacja cen sprzedaży samochodów o 5% oraz zapytanie UPDATE 2 – aktualizacja nazwy jednego z komisów samochodowych.

Uśrednione czasy wykonywania zapytań *UPDATE 1* oraz *UPDATE 2* w badanych SZBD, wraz z błędami

wyliczonymi z rozkładu t-Studenta (95%), zaprezentowano na rys. 13 oraz rys. 14.



Rys. 13. Uśrednione czasy wykonywania UPDATE 1.



Rys. 14. Uśrednione czasy wykonywania UPDATE 2.

W przypadku zapytań *UPDATE* można zauważyć istotną różnicę w uzyskanych wynikach. Zapytanie *UPDATE 1*, które dotyczyło aktualizacji wartości liczbowych w bazie danych, najszybciej zostało wykonane w MariaDB, a najdłużej w PostgreSQL. Rezultaty odwróciły się w przypadku aktualizacji wartości tekstowych – w tym przypadku czas wykonywania zapytania *UPDATE 2* był najkrótszy w PostgreSQL, chociaż różnice między czasami z pozostałych SZBD są minimalne.

Operacja *UPDATE* najlepiej przedstawia korzystny wpływ normalizacji bazy danych na proces aktualizacji danych w bazie – w bazie normalizowanej zapytanie *UPDATE 2* dotyczyło aktualizacji tylko jednego rekordu w tabeli *seller_info*, kiedy w bazie zdenormalizowanej, dla uzyskania takiego samego efektu, konieczna była analiza wszystkich 393 818 rekordów zawartych w tabeli *car_sale_info*.

Ostatnim etapem badania wpływu normalizacji bazy danych na jej wydajność było przeprowadzenie operacji *DELETE* na badanych SZBD. W związku z różnicami w implementacji języka SQL w używanych SZBD, operacja *DELETE* musiała zostać odpowiednio zmodyfikowana, w zależności od SZBD, co jest widoczne na rys. 15.

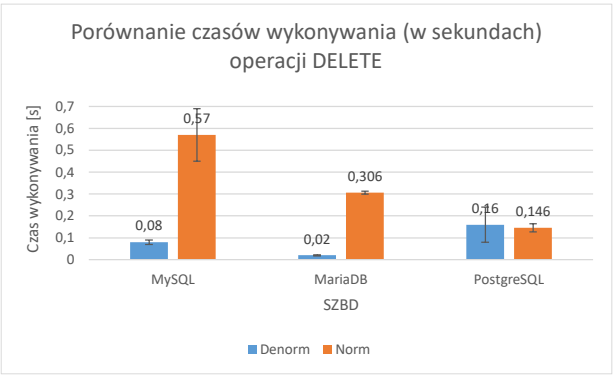
```
-- DELETE 1
-- denorm
DELETE FROM car_sale_info LIMIT 10000; -- wersja dla MySQL/MariaDB
DELETE FROM car_sale_info
WHERE vin IN (
    SELECT vin FROM car_sale_info LIMIT 10000
    ); -- wersja dla PostgreSQL

-- norm
CREATE TEMPORARY TABLE temp_vin AS SELECT vin FROM car_details LIMIT 10000;
DELETE sa FROM sale_details sa JOIN temp_vin t ON sa.vin = t.vin;
DELETE c FROM car_details c JOIN temp_vin t ON c.vin = t.vin;
DROP TEMPORARY TABLE temp_vin; -- wersja dla MySQL/MariaDB

DELETE FROM car_details WHERE vin IN (SELECT vin FROM car_details LIMIT 10000);
DELETE FROM sale_details WHERE vin IN (SELECT vin FROM car_details LIMIT 10000); -- wersja dla PostgreSQL
```

Rys. 15. Zapytania wchodzące w skład operacji DELETE.

Uśrednione czasy wykonywania operacji DELETE w badanych SZBD, wraz z błędami wyliczonymi z rozkładu t-Studenta (95%), przedstawia rys. 16.



Rys. 16. Uśrednione czasy wykonywania operacji DELETE.

Konieczność wykonania w bazie normalizowanej kilku zapytań (w tym zastosowania tabel tymczasowych), w celu uzyskania porównywalnego efektu usunięcia danych, wpłynęła na wydłużony czas operacji *DELETE*, w porównaniu z wynikami dla bazy zdenormalizowanej, dla systemów MariaDB oraz MySQL. W przypadku PostgreSQL, użycie podzapytania w operacji *DELETE* w bazie zdenormalizowanej z pewnością wpłynęło na wydłużenie czasu wykonania operacji w porównaniu z odpowiednimi wynikami dla MariaDB i MySQL. Niemniej jednak, pomimo użycia podzapytań, PostgreSQL uzyskał najlepszy wynik dla operacji *DELETE* w bazie normalizowanej.

4.2. Integralność

Proces normalizacji bazy danych może zmniejszyć lub całkowicie wyeliminować ryzyko wystąpienia w bazie danych tzw. anomalii: wstawiania, aktualizacji oraz usuwania [6]. W ramach analizy wpływu normalizacji bazy danych na integralność danych w niej przechowywanych przedstawiono trzy scenariusze, w których może dojść do wystąpienia ww. anomalii w utworzonej bazie zdenormalizowanej. Dla każdego scenariusza zaprezentowano również w jaki sposób baza normalizowana może zapobiec wystąpieniu potencjalnych problemów.

a. Anomalia wstawiania (*INSERT INTO*)

Problem anomalii wstawiania może wystąpić w bazie zdenormalizowanej w sytuacji, kiedy będziemy chcieli dodać informacje np. o nowym komisie samochodowym (nazwa komisu i stan, w którym komis się znajduje). Struktura bazy zdenormalizowanej (a dokładniej tabeli *car_sale_info*) uniemożliwia dodanie informacji o komisie bez jednoczesnego dodania wielu

informacji o samochodzie, który w tym komisie został sprzedany. W przypadku braku dostępu do danych samochodu, dane związane z komisem nie będą mogły zostać umieszczone w bazie.

W bazie normalizowanej anomalia ta nie wystąpi, ponieważ za przechowywanie informacji o komisach samochodowych odpowiada wyłącznie tabela *seller_info*. Tym samym umieszczenie nowego komisu w bazie danych nie wymaga posiadania informacji o sprzedaży danego samochodu.

b. Anomalia aktualizacji (*UPDATE*)

Anomalia aktualizacji może pojawić się w sytuacji, kiedy zmiana pewnej wartości wymaga aktualizacji wielu rekordów jednocześnie. Załóżmy, że w bazie zdenormalizowanej błędnie zapisano jedną z wartości, która jest związana z typem nadwozia samochodu (kolumna *body*). W celu naprawy błędu konieczna będzie aktualizacji wielu (setek albo tysięcy) rekordów, które przechowują błędne wpisaną wartość.

W przypadku bazy normalizowanej, obecność tabeli *body_info* oraz zdefiniowanie dla każdego typu nadwozia unikalnej wartości ID (kolumna *body_id*), która jest przechowywana jako klucz obcy w tabeli *car_details*, umożliwia naprawienie analogicznego błędu za pomocą aktualizacji zaledwie jednego rekordu w tabeli *body_info*.

c. Anomalia usuwania (*DELETE*)

Anomalia usuwania jest szczególnie niebezpieczna dla zachowania integralności danych przechowywanych w bazie. Anomalia ta występuje w momencie, gdy usunięcie pewnych informacji z bazy wiąże się z utratą innych informacji. Jeżeli z tabeli *car_sale_info* bazy zdenormalizowanej chcielibyśmy usunąć rekord danego samochodu, a informacje związane z komisem samochodowym, który sprzedał ten samochód, nie występują w innych rekordach tabeli *car_sale_info*, to operacja usunięcia tego rekordu z bazy spowoduje utratę informacji zarówno o tym samochodzie, jak i o komisie.

Podobnie jak w przypadku anomalii wstawiania problem ten, w bazie normalizowanej, rozwiązuje obecność tabeli *seller_info*, która przechowuje informacje o komisach samochodowych niezależnie od przechowywanych informacji o samochodach, przez co usunięcie informacji z *car_details* nie powoduje utraty informacji z *seller_info*.

5. PODSUMOWANIE

Niniejszy artykuł skupia się na porównaniu wpływu normalizacji bazy danych na jej wydajność i integralność, z uwzględnieniem różnych SZBD. Uzyskane wyniki i przedstawione analizy oraz wnioski powinny zachęcić do rozważań nt. wyboru odpowiedniej dla naszych potrzeb architektury projektowanej bazy danych oraz optymalnego relacyjnego systemu zarządzania bazą danych.

W zależności od celów jakie stawiamy przed bazą danych, lepszym wyborem może być baza zdenormalizowana – kiedy nacisk kładziemy na szybkość operacji *Read* i/lub *Create* - lub baza normalizowana – kiedy istotna jest dla nas szybkość operacji *Update* i/lub zachowanie integralności danych poprzez unikanie występowania przedstawionych w rozdziale 4.2 anomalii.

MariaDB oferuje bardzo dobre wyniki w przypadku operacji *Create* oraz *Update*. PostgreSQL zdecydowanie najlepiej radzi sobie z operacjami *Read*. Wyniki operacji *Delete* w badanych SZBD są zbliżone do siebie. MySQL w większości eksperymentów wydajnościowych uzyskiwał najgorsze wyniki spośród trzech wybranych systemów. Różnice na niekorzyść MySQL są widoczne w szczególności w czasach wykonywania zapytań w bazie normalizowanej oraz w rozmiarze bazy normalizowanej w systemie.

LITERATURA

[1] Codd, E. F. 1970. „A relational model of data for large shared data banks”. *Communications of the ACM*, 13 (6) : 377–387.

[2] Codd, E. F. 1971. „Further Normalization of the Relational Model”. *IBM Research Report RJ909*.

[3] Codd, E. F. 1974. „Recent Investigations into Relational Data Base Systems”. *IBM Research Report RJ1385*.

[4] Figueiredo R., Lucas M., Abbasi M., Martins P., Wanzeller C. 2023. „Comparative Analysis of Normalized and Non-Normalized Databases: Performance, Flexibility, and Scaling Considerations”. *2023 Second International Conference On Smart Technologies For Smart Nation (SmartTechCon)*, 625–629.

[5] Khalil O. K., Boudjella A., Samir, B. B. 2013. „Comparison between Normalized Databases Implemented with Different Database Systems”. *Advanced Materials Research*, 774-776 : 1827-1832.

[6] Kulawiak M., Goczyła K. „Normalizacja baz danych”. Politechnika Gdańska, Wydział Elektroniki, Telekomunikacji i Informatyki. Dostępny w internecie: https://pg.edu.pl/documents/1403427/1bacfc1e-012d-4800-b6b2-4f0c4cce3c99. Data uzyskania dostępu: 20 grudnia 2024.

[7] Rolik O., Ulianytska K., Khmeliuk M., Khmeliuk V., Kolomiiets U. 2021. „Increase Efficiency of Relational Databases Using Instruments of Second Normal Form”. *2021 IEEE 3rd International Conference on Advanced Trends in Information Theory (ATIT)*, 221-225.

[8] Rudnicki Z. 2015. „Wprowadzenie do informatyki i programowania. Wydanie drugie, poprawione”. *Wydawnictwa AGH, Kraków*.

[9] Tunguz B. 2021. „Used Car Auction Prices”. Dostępny w internecie: https://www.kaggle.com/datasets/tunguz/used-car-auction-prices. Data uzyskania dostępu: 12 listopada 2024.

[10] Yunianto D., Putra Y., Rahmad C. 2023. „Comparison of Relational Database Modeling Performance based on Number of Normalized Entities”. *SMARTICS Journal*, 9 (1) : 42-48.