JOURNAL
COMPUTER SCIENCES INSTITUTE

JCSI 26 (2023) 48–53
Received: 25 September 2022
Accepted: 17 October 2022

Choosing the optimal database system to create a CRM system

Wybór optymalnego systemu baz danych do stworzenia systemu CRM

Łukasz Szwałek*, Jakub Smołka

Department of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract

MSSQL, MySQL and PostgreSQL are some of the most popular databases. The selection of the database for the creation of a CRM system is based mainly on the assessment of its effectiveness in terms of speed. The article aims to choose the optimal database system to create an effective CRM system. The literature review led to the hypothesis that MSSQL will be the fastest. A series of experiments using the app served as a test. During the research, a series of experiments were carried out using the order processing module, which is part of a larger CRM system aimed at the e-commerce industry. Each query was measured 10 times, the results were averaged. The research results did not confirm the hypothesis about the speed and significant advantage of the MSSQL database. The results showed the advantage of PostgreSQL over other databases.

Keywords: database system; CRM; Customer relationship management

Streszczenie

MSSQL, MySQL i PostgreSQL to jedne z najpopularniejszych systemów baz danych. Dobór bazy do stworzenia systemu CRM opiera się głównie na ocenie jej efektywności pod względem szybkości. Artykuł ma za zadanie przedstawić wybór optymalnego systemu baz danych do stworzenia efektywnego systemu CRM. Przegląd literatury skłonił do postawienia hipotezy, że MSSQL będzie najszybszy. Za badanie posłużyła seria eksperymentów z użyciem aplikacji. Podczas badań przeprowadzono serię eksperymentów z użyciem modułu przetwarzania zamówień będącego częścią większego systemu CRM skierowanego do branży e-commerce. Każde zapytanie zostało zmierzone 10-krotnie, wyniki uśredniono. Wyniki badań nie potwierdziły hipotezy o szybkości i znacznej przewadze bazy MSSQL. Wyniki pokazały przewagę bazy PostgreSQL nad innymi bazami.

Słowa kluczowe: system baza danych; CRM; Zarządzanie relacjami z klientami

*Corresponding author

Email address: lukasz.szwalek@pollub.edu.pl (Ł. Szwałek)

1. Wstęp

Baza danych jest sposobem długoterminowego przechowywania informacji, które zostały wygenerowane przez systemy informatyczne. Jest to jeden z podstawowych składników większości aplikacji lub programów. Dzięki obecności bazy danych możliwe staje się realizowanie podstawowych zadań, takich jak przechowywanie, przetwarzanie i pobieranie danych; jest to podstawa każdej aplikacji zorientowanej na dane.

CRM lub Systemy Zarządzania Relacjami z Klientami [1] charakteryzują się przetwarzaniem i generowaniem dużych ilości danych, integracją kilku systemów w jedną całość oraz kompatybilnością z innymi systemami. Gromadzenie danych ma na celu usprawnienie i automatyzację procesów sprzedaży, zarządzanie relacjami z klientami i planowanie zasobów przedsiębiorstwa. W bardziej zautomatyzowanych ustawieniach biznesowych transakcje dla całego przedsiębiorstwa są rejestrowane przez jeden system, a następnie mogą być integrowane z innymi systemami, takimi jak zasoby ludzkie, finanse i marketing. CRM to kluczowe narzędzie, dzięki któremu organizacje mogą czerpać korzyści ze zwiększonej produktywności, rentowności i przewagi konkurencyjnej. CRM jest szeroko stosowany przez firmy z branż m.in. telekomunikacyjnej, bankowej i finansowej oraz handlu detalicznego. Dane powinny być dostępne zawsze i wszędzie w przeciągu kilku, kilkunastu milisekund, dlatego tak ważne jest prawidłowe dobranie odpowiedniego silnika bazy danych. Aktualnie na rynku relacyjnych systemów zarządzania bazami danych jednymi z najpopularniejszych są MySQL, Microsoft SQL Server oraz PostgreSQL. Nierelacyjne systemy nie zostały uwzględnione w tym porównaniu ze względu na specyfikę CRM, gdzie struktura tabel i relacje między nimi są z góry ustalone.

2. Cel i zakres badań

Celem artykułu jest analiza wydajności trzech wybranych systemów bazodanowych w kontekście systemu CRM. Porównanie wyników tej analizy pozwoli określić, który system bazodanowy najlepiej współpracuje z tego typu systemami. Badania zostaną przeprowadzone na serii scenariuszy, które reprezentują typowe przypadki użycia. Przeprowadzane badanie będzie wykonywane przy użyciu programu Postman [2] umożliwiającego sprawdzenie czasów odpowiedzi API systemu CRM. Dzięki powyższemu rozwiązaniu możliwa będzie weryfikacja szybkości wykonywania zapytań i obserwacja wykorzystywanych zasobów przez poszczególne bazy danych. Porównanie to pomoże w wyborze konkretnego systemu bazodanowego, który najlepiej odpo-

wiada wybranemu zastosowaniu. Badania zostały oparte na aplikacji wykorzystującej bibliotekę Doctrine [3] oraz na badanych bazach danych. Ze względu na ograniczenia sprzętowe i zasobowe badanie zostało ograniczone do analizy wydajności baz danych. W niniejszym artykule, została sformułowana następująca teza badawcza:

T1: Baza danych oparta o MSSQL jest najwydajniejsza spośród testowanych systemów.

Prawdziwość zaprezentowanej tezy zostanie sprawdzona na podstawie wykonanych badań.

3. Przegląd literatury

W artykule „Performance analysis of NoSQL and relational databases with MongoDB and MySQL" [4] autorstwa Benymol'a Jose'a i Sajimon'a Abraham'a przedstawiono zalety baz nierelacyjnych względem baz relacyjnych, wyjaśniono różnice w czasie wykonywania operacji. Badania zostały przeprowadzone poprzez wykonywanie zapytań różnego typu za pomocą programów MySQL Workbench oraz MangoDB studio3T. Podsumowując informacje zwarte w artykule należy wskazać, iż testy zostały przeprowadzone na bazach zorientowanych na przechowywanie danych w postaci dokumentów bez zdefiniowanej struktury danych. Autorzy wykazali, że do baz przechowujących duże ilości danych w formie dokumentów warto stosować nierelacyjne bazy danych.

Kolejną pracą naukową poświęconą opisywanej tematyce jest „Comparison of MySQL, MSSQL, PostgreSQL, Oracle databases performance, including virtualization" [5] autorstwa Rafała Kleweka, Wojciecha Truszkowskiego, Marii Skublewska-Paszkowskiej. Autorzy oceniali wydajność czterech relacyjnych baz danych: Oracle, MSSQL, MySQL oraz PostgreSQL. Bazy były testowane za pomocą aplikacji webowej umożliwiającej tworzenie i rozwiązywanie testów wielokrotnego wyboru. Testowano podstawowe operatory języka SQL, a także klauzulę WHERE w kilku wariantach. Postawiono hipotezę, że silnik Oracle będzie najszybszy. Przeprowadzone testy wykazały słuszność tej tezy. We wnioskach zostały zawarte następujące konkluzje:

- Bez względu na rozmiar bazy silnik Oracle osiągał porównywalne czasy.
- Silnik Oracle nagorzej radził sobie z zapytaniami zawierającymi podzapytania.

W artykule „Efficiency of databases in Django-based applications" [6] autorstwa Bartosza Nejmana i Beaty Pańczyk zostało przedstawione porównanie trzech systemów bazodanowych: MySQL, PostgreSQL oraz MongoDB. Autorzy przeprowadzili testy tworzenia, pobierania i warunkowego pobierania danych z testowanych baz danych przy pomocy aplikacji stworzonej na szkielecie Django [7]. We wnioskach artykułu zostały zawarte następujące stwierdzenia:

- Silnik MangoDB nie nadaje się do współpracy ze szkieletem Django ze względu na potrzebę zastosowania zewnętrznej biblioteki do przetwarzania zapytań ORM na składnię MangoDB.
- Z operacjami zapisu danych lepiej radzi sobie baza MySQL, natomiast z odczytem PostgreSQL.

Artykuł „Analiza wydajności relacyjnych baz danych Oracle oraz MSSQL na podstawie aplikacji desktopowej" [8], której autorami są Grzegorz Dziewit, Jakub Korczyński oraz Maria Skublewska-Paszkowska, zawiera porównanie wydajności baz danych Oracle i MS SQL w aplikacjach desktopowych. Artykuł zawiera informacje dotyczące metody wykorzystywanej podczas porównywania relacyjnych systemów bazodanowych w zakresie średniego czasu wykonywania poszczególnych zapytań. System opracowany przez badaczy umożliwia określenie, który system jest lepszy w zależności od funkcjonalności aplikacji. Autorzy w artykule udowodnili, iż baza danych MS SQL lepiej wykonuje operacje INSERT oraz UPDATE niż baza Oracle, natomiast w przypadku zapytań SELECT: krótszy czas wykonania uzyskał system Oracle.

Istnieje szereg badań ukazujących różnicę pomiędzy popularnymi bazami relacyjnymi jak i nierelacyjnymi. Jednak każdy omawiany artykuł bada w inny sposób - wykorzystując dedykowane narzędzia do każdej z baz, przez co wyniki mogą być zaburzone - lub przez pełnoprawne aplikacje webowe, gdzie na końcowy czas składa się także generowanie widoku aplikacji wraz z danymi. Powyższe uwagi przyczyniły się do stworzenia testów, w których mniejsze znaczenie mają czynniki środowiskowe.

4. Aplikacja testowa

Do przeprowadzenia badań stworzona została aplikacja typu REST API [9], której podstawowym założeniem jest możliwość odczytywania, aktualizacji i zapisywania danych. Do tego celu powstały trzy endpointy (Listing 1), po jednym dla każdej z operacji. Aplikacja została stworzona w oparciu o szkielet aplikacji Symfony [10].

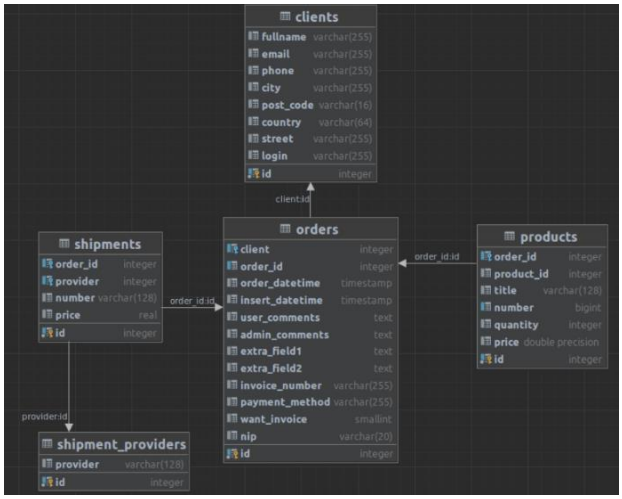
Listing 1: Endpointy API

```
getOrders:
  path: /api/getOrders
  controller: App\Controller\DefaultController::getOrders
  methods: [GET]

insertOrders:
  path: /api/insertOrders
  controller: App\Controller\DefaultController::insertOrders
  methods: [POST]

updateOrders:
  path: /api/updateOrders
  controller: App\Controller\DefaultController::updateOrders
  methods: [ PUT ]
```

Dzięki zastosowaniu biblioteki Doctrine w jednej aplikacji za pomocą zmiany jedynie konfiguracji ten sam kod źródłowy jest wstanie obsłużyć wszystkie testowane bazy danych. Testowane bazy danych zostały przygotowane w trzech silnikach bazodanowych:

- MySQL ver.8.0.29,
- PostgreSQL ver. 14.4,
- Microsoft SQL Server ver.2019-latest.

Każda z nich została napełniona tymi samymi danymi. Aby wyeliminować czynnik środowiskowy każda baza danych została uruchomiona w oddzielnym kontenerze Docker [11]. Każda baza została stworzona w oparciu o oficjalny obraz, bez modyfikacji zmiennych środowiskowych. Schemat bazy danych został zaprezentowany na Rysunku 1.



Rysunek 1: Schemat bazy danych.

5. Metoda badawcza

Eksperyment polegał na przetestowaniu i uruchomieniu testów przygotowanych w programie Postman. Testy zostały wykonane dla trzech scenariuszy, w których zostały wykorzystane polecenia SELECT, INSERT, UPDATE języka SQL:

- odczyt z bazy (SELECT),
- zapis do bazy (INSERT),
- modyfikacja danych (UPDATE).

Każdy scenariusz został wykonany w trzech seriach. Każda seria badała inny rozmiar zapytania, ilość danych w tabelach była za każdym razem taka sama i została przedstawiona w Tabeli 1. Zapytania były tworzone dla 10, 100 i 1000 wierszy z tabeli orders.

Tabela 1: Liczba wierszy w poszczególnych tabelach

Tabela	Ilość danych
orders	2 000 000
clients	1 750 000
products	3 300 000
shipments	2 000 000
shipment_providers	100

Testy były uruchamiane na systemie Ubuntu 20.04 LTS, a bazy danych w kontenerach platformy Docker w wersji 20.10. W Tabeli 2 została przedstawiona specyfikacja maszyny, na której zostały przeprowadzone testy. W trakcie wykonywania testów kontenery zawierające bazy danych były monitorowane za pomocą komendy „docker stats", która wyświetla takie dane jak: wykorzystanie procesora, pamięci RAM.
Sposób realizacji każdego z testów przez kod źródłowy interfejsu REST API został przedstawiony w poniższych podpunktach.

Tabela 2: Specyfikacja urządzenia testowego

Procesor	Procesor Intel(R) Core(TM) i7-7700HQ
Dysk	512GB SSD M.2
RAM	16GB 2133MHz

5.1. Odczyt z bazy

Listing 2 przedstawia funkcję odpowiadającą za testowanie polecenia SELECT. Przyjmuje dwa parametry pozwalające manipulować wielkością otrzymywanych wyników. Zapytanie łączy dwie tabele za pomocą operacji LEFT JOIN. Dla potwierdzenia zwracana jest jedynie liczba wierszy wygenerowanych przez zapytanie.

Listing 2: Ciało funkcji getOrders

```
public function getOrders(Request $request): Response
{
    $offset = $request->query->get('offset');
    $limit = $request->query->get('limit');
    $query = $this->em->createQueryBuilder()->select(['o', 'p'])
        ->from(Orders::class, 'o')
        ->leftJoin(Products::class, 'p', 'WITH', 'p.order = o.id')
        ->setFirstResult($offset)
        ->setMaxResults($limit);
    $results = $query->getQuery()->getResult();

    return new Response(count($results), 200);
}
```

5.2. Zapis do bazy

Funkcja przedstawiona na Listingu 3 odpowiada za testowanie polecenia INSERT. Parametrem wejściowym jest tablica zamówień w zawierająca dane zamówienia, zamawiającego i produktów. Ta funkcja testuje równoczesny i wielokrotny zapis do trzech tabel: orders, clients, products.

Listing 3: Ciało funkcji insertOrders

```
public function insertOrders(Request $request): Response
{
    $orders = json_decode($request->getContent(), true);
    foreach ($orders['orders'] as $orderData) {
        $client = $this->createClientObject($orderData);
        $this->em->persist($client);
        $this->em->flush();
        $order = $this->createOrderObject($orderData, $client);
        $this->em->persist($order);
        $this->em->flush();
        foreach ($orderData['products'] as $productData) {
            $product = $this->createProductObject($productData, $order);
            $this->em->persist($product);
        }
        $this->em->flush();
    }

    return new Response(count($orders['orders']), 200);
}
```

5.3. Modyfikacja danych

Listing 4 przedstawia funkcję odpowiadającą za aktualizowanie danych. Tak jak w przypadku odczytu: przyjmuje dwa parametry umożliwiające manipulację liczbą edytowanych danych. Funkcja pobiera wskazaną w teście liczbę wierszy. Następnie modyfikowana jest

jedna kolumna z tabeli orders - extra_fields2 - przechowująca zmienne tekstowe, a następnie jedna kolumna o nazwie quantity – przechowującą liczbę produktów, z tabeli products dla wszystkich powiązanych wierszy.

Test polecenia SELECT i UPDATE został wykonany dla trzech wielkości zapytań 10, 100, 1000 wierszy. Test polecenia UPDATE polegał na zapisaniu do bazy 10, 100, 1000 zamówień. Każdy z testów polegał na dziesięciokrotnym wykonaniu danego scenariusza w celu uzyskania możliwie jak najbardziej realnych wyników. Program Postman pozwala na przygotowanie i zautomatyzowanie testów, dzięki czemu każdy test był wykonany w jednakowy sposób. Czasy wykonania każdego z testu zostały odczytane z konsoli programu i uśrednione dla każdego wariantu w serii.

Listing 4: Ciało funkcji updateOrders

```
public function updateOrders(Request $request): Response
{
    $offset = $request->query->get('offset');
    $limit = $request->query->get('limit');
    $orders = $this->em->getRepository(Orders::class)
        ->findBy([], null, $limit, $offset);
    foreach ($orders as $order) {
        $order->setExtraField2($order->getOrderId());
        $products = $this->em->getRepository(Products::class)
            ->findBy(['order' => $order]);
        foreach ($products as $product) {
            $product->setQuantity(99);
        }
    }
    $this->em->flush();

    return new Response(count($orders), 200);
}
```

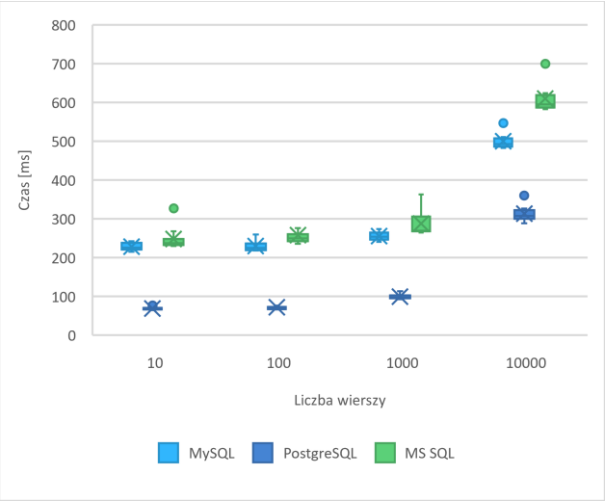
6. Analiza wyników

Przed rozpoczęciem badań został wykonany pomiar użycia zasobów przez bazy nie wykonywały żadnych operacji. Wynik obserwacji przedstawiony został w Tabeli 3.

Tabela 3: Zużycie zasobów w trakcie spoczynku

MySQL		PostgreSQL		MSSQL	
CPU [%]	RAM [MB]	CPU [%]	RAM [MB]	CPU [%]	RAM [MB]
0,34	382	0,1	34	2,1	1167

W pierwszym badanym scenariuszu sprawdzany był czas wykonywania instrukcji z poleceniem *SELECT*, wyniki zostały przedstawione na Rysunku 2 a uśrednione wyniki serii w Tabeli 4 natomiast zużycie zasobów zostało przedstawione w Tabeli 5. Porównując poszczególne serie zapytań widać bardzo niewielką różnicę w czasie wykonania, dlatego dla tego scenariusza został wykonany dodatkowo test dla 10000 wierszy. Porównując silniki w każdej serii znaczącą przewagę ma baza oparta na silniku PostgreSQL.



Rysunek 2: Wyniki testu funkcji getOrders.

Tabela 4: Średnie wyniki testu funkcji getOrders

Liczba wierszy	MySQL		PostgreSQL		MSSQL	
	Średni czas [ms]	Std	Średni czas [ms]	Std	Średni czas [ms]	Std
10	228,20	9,0	68,6	3,6	248,2	29,7
100	229,70	13,5	72,4	8,5	258,3	27,4
1000	255,30	10,3	99	6,0	287,7	319
10000	499,9	18,3	313,2	19,9	609,9	34,9

Tabela 5: Zużycie zasobów w trakcie testu funkcji getOrders

Liczba wierszy	MySQL		PostgreSQL		MSSQL	
	CPU [%]	RAM [MB]	CPU [%]	RAM [MB]	CPU [%]	RAM [MB]
10	91,9	541	73,6	95	93,6	1225
100	89,7	541	70,1	95	93,1	1225
1000	80,5	541	55,9	95	84,6	1225
10000	49,7	541	23,3	95	58,3	1225

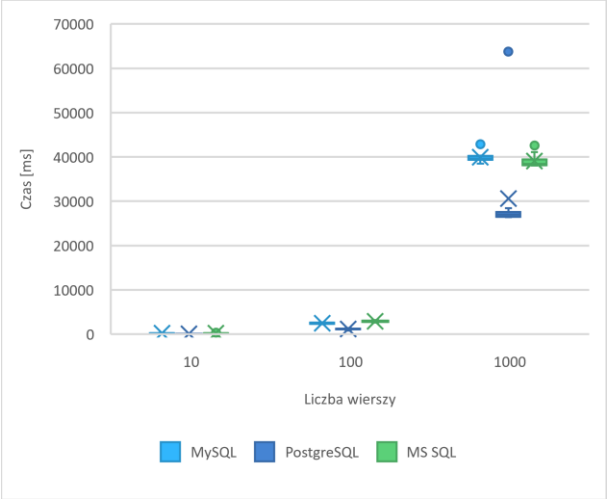
W następnym badaniu testującym scenariusz z poleceniem INSERT widać znacząco zwiększający się czas wykonania serii wraz ze zwiększającą się liczbą danych do zapisania, widać to zarówno na wykresie (Rysunek 3) jak i Tabeli 6 z uśrednionymi wartościami każdej serii. Natomiast z obserwacja zużycia zasobów (Tabela 7) pokazuje spadek użycia CPU i równoczesny przyrost użycia pamięci RAM wraz ze zwiększaniem liczby wierszy w teście. Porównując silniki ponownie dominuje PostgreSQL, jedynie przy ostatniej serii czasy wykonania były bardzo zbliżone.

Tabela 6: Średnie wyniki testu funkcji insertOrders

Liczba wierszy	MySQL		PostgreSQL		MSSQL	
	Średni czas [ms]	Std	Średni czas [ms]	Std	Średni czas [ms]	Std
10	222,6	43,1	85,1	16,9	201,2	63,4
100	2475,6	145,1	1173,4	177,7	2930,3	184,6
1000	39932,3	1183,8	30642,5	11664,9	39060,7	1512,5

Tabela 7: Zużycie zasobów w trakcie testu funkcji insertOrders

Liczba wierszy	MySQL		PostgreSQL		MSSQL	
	CPU [%]	RAM [MB]	CPU [%]	RAM [MB]	CPU [%]	RAM [MB]
10	22,4	536	28,4	103	60,3	1229
100	21,2	536	24,5	106	63,4	1251
1000	12,5	536	8,2	110	34,4	1255



Rysunek 3: Wyniki testu funkcji insertOrders.

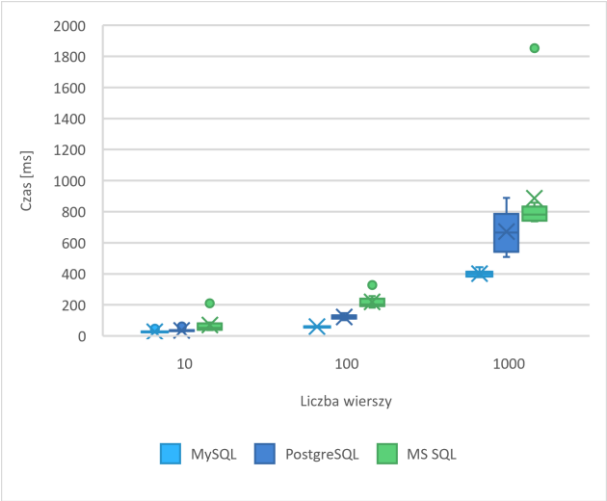
W ostatnim ze scenariuszy testowano polecenie UPDATE. Wyniki przedstawione na Rysunku 4 pokazują duży rozrzut między pomiarami w serii 1000 wierszy dla bazy PostreSQL. Tabela 8 przedstawia uśrednione wyniki serii, które także pokazują przyrost czasu między seriami. Jednocześnie wraz ze zwiększaniem liczby wierszy rosło wykorzystanie CPU i pamięci RAM dla każdej z baz (Tabela 9). W tym scenariuszu w każdej serii najniższe czasy osiągał silnik MySQL z niewielką przewagą nad silnikiem PostgreSQL jak i MSSQL.

Tabela 8: Średnie testu funkcji updateOrders

Liczba wierszy	MySQL		PostgreSQL		MSSQL	
	Średni czas [ms]	Std	Średni czas [ms]	Std	Średni czas [ms]	Std
10	27,7	7,4	35,6	9,8	69,8	52,3
100	59,5	6,6	122,6	13,4	219,1	44,4
1000	401	19,8	671,7	130,2	886,9	341,6

Tabela 9: Zużycie zasobów w trakcie testu funkcji updateOrders

Liczba wierszy	MySQL		PostgreSQL		MSSQL	
	CPU [%]	RAM [MB]	CPU [%]	RAM [MB]	CPU [%]	RAM [MB]
10	35,4	522	33,4	347	47,1	1226
100	47,9	523	54,8	379	69,0	1259
1000	73,6	527	68,4	453	84,5	1318



Rysunek 4: Wyniki testu funkcji updateOrders.

7. Wnioski

W niniejszym artykule przeprowadzone zostały badania wydajności trzech najpopularniejszych baz danych dostępnych na rynku, czyli MySQL, PostgreSQL oraz MSSQL. Dla zapewnienia neutralnych warunków każdy z silników bazodanowych został uruchomiony w oddzielnym kontenerze oprogramowania Docker z wykorzystaniem oficjalnych, dedykowanych obrazów. Analizując wyniki przyprowadzonych badań jedynie w przypadku testu sprawdzającego wydajność wykonywania polecenia SELECT można zaobserwować znaczącą przewagę bazy danych opartej o silnik PostgreSQL. Przewaga jest widoczna zwłaszcza w pierwszych trzech seriach, gdzie testy bazy PostgreSQL uzyskiwały trzykrotnie krótsze czasy w porównaniu do pozostałych testowanych baz. W pozostałych dwóch testach poleceń INSERT i UPDATE nie została odnotowana tak znacząca przewaga którejkolwiek z testowanych baz. W teście polecenia INSERT także najlepsze wyniki osiągnął PostgreSQL lecz wraz ze wzrostem liczby przetwarzanych wierszy czasy każdej z baz były zbliżone. W ostatnim teście z poleceniem UPDATE z niewielką przewagą najszybciej poradził sobie silnik MySQL. Pod względem zużycia zasobów maszyny, baza oparta o silnik PostgreSQL także wypada najlepiej spośród testowanych. Bez obciążenia baza zużywa znikomy procent CPU jak i znikomą ilość pamięci RAM. Podczas wykonywania testów wartości te znacząco wzrastają, lecz nie osiągają wartości obserwowanych na pozostałych bazach. Teza, która została postawiona w artykule: „Baza danych oparta o MSSQL jest najwydajniejsza spośród testowanych systemów” zosta-

ła obalona. W wyniku analizy badań przeprowadzonych na interfejsie REST API stworzonym na szkielecie aplikacji Symfony, wykorzystującym bibliotekę Doctrine, najbardziej wydajnym systemem okazał się PostgreSQL. Jednakże w przypadku badań polecenia SELECT i INSERT zużycie procesora spada wraz ze wzrostem liczby wierszy co świadczy o większym zapotrzebowaniu na moc obliczeniową przez aplikację REST API. Jest to znak, iż badania symulujące większe systemy powinny być wykonywane na więcej niż jednej maszynie, aby zapewnić takie sam dostęp do mocy obliczeniowej każdemu elementowi podlegającemu badaniu.

Wydajność baz danych jest niezwykle istotna zwłaszcza w przypadku przechowywania gigabajtów danych w postaci relacyjnych baz. Z testu polecenia INSERT i UPDATE wynika jasno, że różnice w czasach wykonania dla poszczególnych baz są niewielkie. Warto również zaznaczyć, że wraz ze wzrostem wielkości zapytań i ilości przetwarzanych danych czasy wykonania wzrastają. Nawiązując do wstępu, system CRM ma usprawniać pracę z danymi, dlatego zapytania trwające prawie 40 sekund - w przypadku zapytania zapisującego 10 000 zamówień - zbliża się do bariery „przepływu myśli użytkownika" [12], nie uwzględniając czasu potrzebnego na przesłanie danych między serwerem a użytkownikiem i czasu potrzebnego do wygenerowania widoku. Biorąc pod uwagę powyższe stwierdzenie nasuwa się wniosek, iż na czasy wykonania zapytań składają się nie tylko wydajność bazy, ale także optymalizacja wykonywanych zapytań poprzez zmniejszenie wyników za pomocą klauzuli LIMIT i paginacji wyników za pomocą klauzuli OFFSET.

Literatura

[1] Czym jest system CRM, https://www.oracle.com/pl/cx/what-is-crm/, [2014-02-19].

[2] Dokumentacja Postman, https://learning.postman.com/docs/getting-started/introduction/, [2022-07-05].

[3] Dokumentacja biblioteki Doctrine, https://www.doctrine-project.org/, [2019-06-24].

[4] B. Jose, S. Abraham, Performance analysis of NoSQL and relational databases with MongoDB and MySQL, Materials Today: Proceedings 24, (2020) 2036-2043, https://doi.org/10.1016/j.matpr.2020.03.634.

[5] W. Truskowski, R. Klewek, M. Skublewska-Paszkowska, Comparison of MySQL, MSSQL, PostgreSQL, Oracle databases performance, including virtualization, Journal of Computer Sciences Institute 16 (2020) 279-284, https://doi.org/10.35784/jcsi.2026.

[6] B. Nejman, B. Pańczyk, Efficiency of databases in Django-based applications, Journal of Computer Sciences Institute 11 (2019) 82-85, https://doi.org/10.35784/jcsi.142.

[7] Dokumentacja Django, https://docs.djangoproject.com/en/4.1/, [02-08-2022].

[8] G. Dziewit, J. Korczyński, M. Skublewska-Paszkowska, Performance analysis of relational databases Oracle and MS SQL based on desktop application, Journal of Computer Sciences Institute 8 (2018) 263-269, https://doi.org/10.35784/jcsi.693.

[9] REST API, https://www.ibm.com/pl-pl/cloud/learn/rest-apis, [06-04-2021].

[10] Dokumentacja Symfony, https://symfony.com/doc/5.4/index.html, [09-12-2021].

[11] Dokumentacja Docker, https://docs.docker.com/compose/, [2021-05-10].

[12] J. Nielsen, Usability Engineering, AP Professional, San Francisco, 1993.