JOURNAL
COMPUTER SCIENCES INSTITUTE

JCSI 21 (2021) 309–315
Received: 07 September 2021
Accepted: 26 September 2021

Comparative analysis of connection performance with databases via JDBC interface and ORM programming frameworks

Analiza porównawcza wydajności połączeń z bazami danych poprzez interfejs JDBC i szkielety programistyczne ORM

Mateusz Żuchnik*, Piotr Kopniak

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

Abstract
The research topic of this paper was a comparative analysis of the performance of database connections using different communication methods based on the Java programming language. The investigated tools mediating communication with databases included JDBC drivers and Object-Relational Mapping (ORM) programming frameworks. The research was conducted based on 8 different criteria, in order to select the most effective method and tool for working with relational databases, when developing applications in Java. The different criteria were given weights, which were determined through a survey of Java developers and computer science students. Hibernate turned out to be the best tool without taking into account the weights obtained, and with taking into account the weights the JDBC tool.

Keywords: database connections; Java; performance; ORM framework

Streszczenie
Tematem badań niniejszego artykułu była analiza porównawcza wydajności połączeń z bazami danych za pomocą różnych metod komunikacji w oparciu o język programistyczny Java. W skład badanych narzędzi pośredniczących w komunikacji z bazami danych weszły: sterowniki JDBC i szkielety programistyczne ORM (ang. Object–Relational Mapping). Przeprowadzono badania w oparciu o 8 różnych kryteriów, w celu wyłonienia najbardziej efektywnej metody i narzędzia do pracy z relacyjnymi bazami danych, podczas tworzenia aplikacji w języku Java. Poszczególnym kryteriom przyznano wagi, które zostały określone poprzez ankietę przeprowadzoną wśród programistów języka Java i studentów informatyki. Najlepszym narzędziem bez uwzględnienia pozyskanych wag okazał się Hibernate, a z uwzględnieniem wag narzędzie JDBC.

Słowa kluczowe: połączenia z bazą danych; Java; wydajność; szkielety ORM

*Corresponding author
Email address: mateusz.zuchnik@pollub.edu.pl (M. Żuchnik)

1. Wstęp

We współczesnym świecie z roku na rok liczba osób korzystających z Internetu stale się powiększa. Jak podaje Główny Urząd Statystyczny w Polsce w roku 2020 dostęp do Internetu posiadało 90.4% gospodarstw domowych. W porównaniu z poprzednim rokiem odsetek ten był wyższy o 3,7 punktów procentowych [1]. Wzrost użytkowników korzystających z Internetu wpływa na objętość generowanych w nim danych. Przewiduje się, że w 2025 roku globalna ilość generowanych danych dziennie będzie wynosiła 463 eksabajtów [2]. Z uwagi na to, niezbędne jest posiadanie odpowiedniego systemu do gromadzenia danych i do szybkiego zarządzania nimi.

Jednym z rozwiązań, które pozwala gromadzić i zarządzać dużą ilością danych są bazy danych. Same bazy danych, a co za tym idzie systemy do zarządzania bazami danych znane jako DBMS (ang. DataBase Managment System) stanowią niemal nieodłączny element każdej aplikacji internetowej, desktopowej czy też mobilnej. Można wyróżnić takie DBMS jak: MySql, PostgreSQL, czy H2. Sam dostęp aplikacji do danych odbywa się często poprzez interfejs aplikacyjny ODBC (ang. Open Database Connectivity) [3], który pozwala na komunikację programu z bazą danych w celu przeprowadzenia operacji na tych danych. Język Java posiada swój odpowiednik interfejsu aplikacyjnego API (ang. Application Programming Interface) do łączenia się z DBMS. Nazwa odpowiednika to JDBC (ang. Java DataBase Connectivity) [4].

Język Java od momentu swojego pierwszego wydania 23 stycznia 1996 r. doczekał się bardzo dużej liczby bibliotek i narzędzi wykorzystującej interfejs JDBC. Wymienić tutaj można m.in. szkielety programistyczne takie jak: jOOQ Object Oriented Querying, MyBatis oraz Hibernate. Programiści często potrzebują wybrać konkretne rozwiązanie najbardziej pasujące do zdefiniowanych wymagań. Dotyczy to także połączenia z bazą danych. Z tej też potrzeby przeprowadzono badania mające na celu porównanie wymienionych narzędzi do komunikacji z bazami danych. Wyniki badań pomogą określić efektywność każdego z podejść i mogą ułatwić odpowiedni wybór.

Autorzy zapoznali się z podobnymi przeprowadzonymi badaniami. Wskazać należy artykuł pt.: „Analiza porównawcza technologii odwzorowania obiektowo-relacyjnego dla aplikacji Java”, autorstwa Piotra Błocha i Marka Wojciechowskiego. Celem autorów było zba-

danie możliwości i wydajności narzędzi do mapowania obiektowo-relacyjnego. Autorzy w swoich badaniach skupili się na takich narzędziach jak Hibernate, Oracle Toplink oraz JDO (ang. Java Data Objects). W przeprowadzonych badaniach udało się wykazać, że narzędzia Hibernate oraz Toplink są lepszym rozwiązaniem niż JDO [5].

Kolejnym przeanalizowanym artykułem była praca pt.: „Hybrydowe metody pracy z bazami danych w aplikacjach JEE", autorstwa Katarzyny Jóźwickiej i Mariusza Mitrusa. Celem artykułu była analiza narzędzi w kontekście operacji typu CRUD. W skład badanych narzędzi weszły: JDBC, Hibernate oraz szkielet aplikacyjny Spring. Przeprowadzone badania wykazały, iż najlepszym rozwiązaniem okazała się aplikacja hybrydowa łącząca JDBC i Hibernate. Takie połączenie narzędzi pozwoliło zmniejszyć czas na wykonanie zapytań SQL oraz zoptymalizować wykorzystanie pamięci RAM [6].

Artykuł naukowy pt.: „JEE database applications performance" autorstwa Magdaleny Grzesińskiej, Magdaleny Waszczyńskiej oraz Beaty Pańczyk, prezentuje porównanie wydajności bazodanowych aplikacji JEE z wykorzystaniem różnych interfejsów programistycznych
(JDBC, Hibernate, JOOQ). Wnioskami autorek z przeprowadzonych badań było to, że najlepszym rozwiązaniem dla operacji bazodanowych będzie podejście hybrydowe. W prostych operacjach typu CRUD sugerują, aby wykorzystać narzędzie Hibernate, a dla bardziej złożonych zapytań korzystać z rozwiązań jakie dostarcza jOOQ [7].

2. Cel i problem badawczy

Celem opisywanych badań była analiza porównawcza wydajności połączeń z bazami danych poprzez interfejs JDBC i szkielety programistyczne ORM. Do badań porównawczych wybrano następujące narzędzia: JDBC, jOOQ, MyBatis oraz Hibernate. Wpływ na wybór powyższych narzędzi miała ich rosnąca liczba wyszukiwań w serwisie Google [8] oraz gotowa konfiguracja dostarczana wraz z projektem Spring Data [9]. Rysunek 3 przedstawia charakterystykę popularności każdego z badanych narzędzi.

Problemem badawczym było wyznaczenie, które z wymienionych narzędzi jest najbardziej wydajne pod kątem określonych kryteriów. Uwzględniono kryteria, które w sposób bezpośredni lub pośredni wpływają na pracę programisty z danym narzędziem. Badania zostały przeprowadzone z wykorzystaniem relacyjnego systemu bazodanowego MySQL w wersji 8.0 [10].

W skład kryteriów wchodziły: czas wykonywania zapytania SQL (ang. Structured Query Language), w przypadku operacji typu CRUD, średnie zużycie pamięci RAM, średnie procentowe wykorzystanie procesora, popularność danego narzędzia, potrzeba znajomości SQL, niepodatność na ataki typu „SQL Injection", wsparcie dla dialektów języka SQL oraz sprawdzanie składni SQL. Wagi dla poszczególnych kryteriów zostały określone na podstawie badań ankietowych przeprowadzonych wśród programistów języka Java. Wartości uzyskanych wag zostały przedstawione w Tabeli 3.

3. Wybrane narzędzia

W ramach przygotowań do badań został przeprowadzony przegląd literatury dla badanych narzędzi do komunikacji z bazami danych. Podstawowe cechy każdego z nich przedstawiono poniżej.

3.1. JDBC

JDBC jest interfejsem programowania stworzonym przez ówczesnego właściciela języka Java, czyli firmę Sun Microsystems [11]. JDBC jest odpowiednikiem standardu ODBC i jego zasada działania jest bardzo zbliżona. Rozwiązanie to umożliwia na bezpośrednie połączenie z serwerem baz danych przy wykorzystaniu sterownika. JDBC umożliwia wysyłanie gotowych zapytań SQL, a w wyniku ich przetworzenia otrzymujemy zbiór rezultatów w formie tablicy danych.

3.2. jOOQ

Narzędzie jOOQ posiada mechanizm generowania kodu, który pozwala na budowanie bezpiecznych składniowo zapytań SQL i uzyskanie pełnej kontroli nad generowanym SQL poprzez czyste i wydajne API. Rozwiązanie to daje programiście możliwość uzyskania kontroli nad wygenerowanym kodem [12]. Narzędzie jOOQ w porównaniu do JDBC nie wymaga obsługiwania wszystkich wyjątków (ang. exceptions) w sposób jawny związanych z komunikacją z bazą danych.

3.3. MyBatis

MyBatis jest szkieletem programistycznym z otwartym kodem źródłowym (ang. Open Source). Zapewnia on wsparcie dla operacji typu CRUD (ang. Create Read Update Delete), gdzie logika samego mapowania SQL na obiekty jest oddzielona od kodu odpowiedzialnego za logikę biznesową. Dodatkowo zapewnia dwa sposoby konfiguracji: poprzez pliki konfiguracyjne XML, jak i przy użyciu adnotacji języka Java [13].

3.4. Hibernate

Ostatnim z badanych podejść do komunikacji z relacyjnymi bazami danych jest szkielet Hibernate. Hibernate jest przykładem technologii typu ORM. Dodatkowo szkielet ten w pełni zapewnia implementację dla specyfikacji JPA (ang. Java Persistence API), która została opracowana przez firmę Sun Microsystems [14]. Narzędzie to jest najbardziej rozbudowane i szeroko stosowane przez programistów Java [15].

4. Środowisko i metoda badawcza

Ze względu na charakter prowadzonych badań, niezbędnym było zdefiniowanie środowiska badawczego, na którym zostały przeprowadzone badania. Zdecydowana większość aspektów wykorzystywanych przy badaniu musiała zostać dobrana w taki sposób, aby była powtarzalna. Pozyskane wyniki dla każdej innej konfiguracji środowiska mogłyby znacząco odbiegać od

uzyskanych w niniejszej pracy. Biorąc pod uwagę fakt, iż głównym narzędziem potrzebnym do przeprowadzania badań był komputer, to samo środowisko badawcze zastosowane do wszystkich badań, zostało zdefiniowane poprzez sekcje sprzętową oraz programową.

4.1. Środowisko badawcze

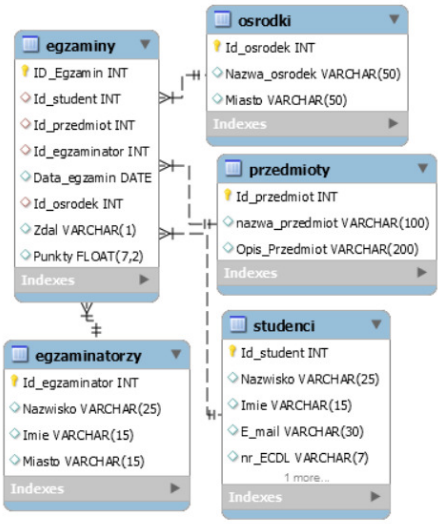
Specyfikacja sprzętowa została przedstawiona w tabeli 1, a specyfikacja oprogramowania została przedstawiona w tabeli 2. Baza danych składała się z 5 tabel: egzaminy, egzaminatorzy, studenci, ośrodki i przedmioty. Schemat struktury bazy danych przedstawiono na rysunku 1. Aplikacja testowa była uruchamiana z wykorzystaniem maszyny wirtualnej HotSpot JVM (ang. Java Virtual Machine) dostarczaną wraz z darmowym odpowiednikiem JDK 11 (ang. Java Development Kit), czyli OpenJDK w wersji 11.0.12+7. Wirtualna maszyna Javy została poddana dodatkowej konfiguracji. Początkowy rozmiar sterty ustawiono na 512MB, czego odpowiednikiem jest flaga (-Xms512m) oraz całościowy rozmiar sterty ustawiono na 2048MB (-Xmx2048m) dla JVM. Dla aplikacji testowej został wykorzystany domyślny GarbageCollector (GC) tzn. „G1 GC'. Dla potwierdzenia wykorzystania tego właśnie GarbageCollector'a, wykorzystano flagę „-XX:+UseG1GC''. Wersje narzędzi JDBC, jOOQ, MyBatis i Hibernate, były dostarczane wraz z wersją Spring Boota 2.4.5. Wykorzystany sterownik JDBC do połączeń z bazą danych MySQL, był również dostarczany w ramach projektu Spring Boota.

Tabela 1: Wykaz specyfikacji sprzętowej, na której zostały przeprowadzone badania

Producent	Lenovo
Model	ideapad 700-15isk
Procesor	Intel Core i5-6300HQ CPU 2,30Ghz
Pamięć RAM	8GB (SO-DIMM DDR4, 2133MHz)
Karta graficzna	NVIDIA GeForce GTX950M Intel HD Graphics 530
Dysk	Samsung SSD 970 EVO PLUS 1TB

Tabela 2: Wykaz oprogramowania. na którym zostały przeprowadzone badania

System operacyjny	Windows 10 Education
Java SDK	OpenJdk 11.0.12+7
Środowisko programistyczne	Intellij IDEA 2021.1
Baza danych	MySQL 8.0
Silnik bazodanowy	InnoDB
Spring Boot	2.4.5



Rysunek 1: Schemat bazy danych wykorzystany podczas przeprowadzania badań.

Każde z narzędzi dostępu do baz danych badano pod kątem 8 różnych kryteriów. Z uwagi na fakt, iż kryteria nie muszą być rozpatrywane na tym samym poziomie ważności tzn. jedno kryterium może być ważniejsze od drugiego, koniecznym było wyznaczenie ich wag. W związku z tym zdecydowano o przeprowadzeniu ankiety wśród programistów Java i studentów informatyki ostatniego roku studiów magisterskich. Do samej ankiety przystąpiło 26 osób. Ankieta składała się z 12 pytań, z czego dwa pierwsze dotyczyły doświadczenia programowania w języku Java. Pozostałe pytania dotyczyły bezpośrednio badanych kryteriów. Celem osoby wypełniającej ankiety było zaznaczenie jednej odpowiedzi w skali od 1 do 5, gdzie 1 oznaczało kryterium mało istotne, a 5 oznaczało kryterium bardzo ważne. Ankieta zawierała jedno pytanie kontrolne, w celu wyeliminowania nierzetelnych odpowiedzi. Samo pytanie kontrolne wymagało od użytkownika zaznaczenia konkretnej odpowiedzi, zdefiniowanej w pytaniu np.: w skali odpowiedzi od 1 do 5, użytkownik powinien zaznaczyć opcję 4. Wszyscy ankietowani odpowiedzieli na pytanie kontrolne poprawnie, zatem uwzględniono wszystkie wyniki zebrane z ankiety. Każde z kryteriów podlegało ocenie. Sposobem obliczania wagi danego kryterium była suma wszystkich odpowiedzi (uzyskanych punktów dla danego kryterium), dzielona przez liczbę ankietowanych. W taki sposób wagę stanowiła średnia arytmetyczna. Wyniki ankiety zostały przedstawione w sekcji „Wyniki".

4.2. Sposób punktowania kryteriów

Sposobem na wyznaczenie, które z badanych narzędzi jest najbardziej efektywne była ich weryfikacja pod kątem opracowanych kryteriów porównawczych wraz z uwzględnieniem wyznaczonych ankietowo wag. Analiza została podzielona na dwie części określone za pomocą aspektów ilościowych i jakościowych. Kryteria ilościowe to takie, których wynikiem była konkretna wartość np. czas wykonania zapytania wyrażony w milisekundach [ms] lub średnie wykorzystanie pa-

mięci RAM wyrażone w megabajtach [MB]. Kryteria jakościowe to takie, których nie da się zmierzyć, np. wsparcie dla dialektów. Uzyskanie wyników względem kryteriów jakościowych było możliwe dzięki analizie dokumentacji technicznej. Sam sposób punktowania był zależny od tego, czy dane kryterium było ilościowe lub jakościowe. Dla kryteriów ilościowych można na podstawie otrzymanych wyników każdemu narzędziu przyporządkować „uzyskane miejsce''. Jeśli dana metoda wypadła najlepiej w kontekście danego kryterium to otrzymywała 3 punkty, drugie najlepsze podejście 2 punkty itp. Najgorzej wypadające rozwiązanie otrzymywało 0 punktów. W przypadku kryterium jakościowego narzędzia mogły go spełniać lub nie, dlatego wynik był jedną z dwóch możliwych wartości. W zależności od tego czy narzędzie spełniało dane kryterium otrzymywało 2 punkty bądź 0 punktów. W końcowym procesie oceniania uzyskane wagi stanowiły mnożnik do uzyskanych punktów dla danego kryterium.

4.3. Kryteria ilościowe

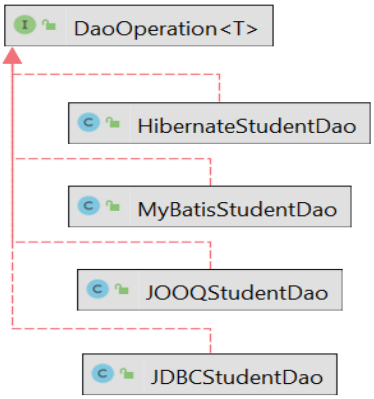
Sposobem pozyskiwania wyników dla poszczególnych kryteriów ilościowych było przeprowadzenie eksperymentu mającego na celu zagregowanie wyników dla poszczególnych przypadków testowych. W skład tych kryteriów wchodziły następująco: czas wykonania zapytania SQL (operacje CRUD), średnie zużycie pamięci RAM, średnie procentowe wykorzystanie procesora oraz popularność danego podejścia mierzona poprzez liczbę zapytań w wyszukiwarce Google. Dla każdego kandydata z grupy przygotowano zapytania SQL typu CRUD i przeprowadzono eksperyment dla 5 zestawów danych. Były to odpowiednio: 1 rekord, 10 rekordów, 100 rekordów, 1000 rekordów i 10 000 rekordów. W zależności od typu operacji dane te były odczytywane, dodawane, aktualizowane lub usuwane z bazy danych. Wszystkie zapytania podlegały mechanizmowi „grupowania" (ang. batch), w celu zwiększenia wydajności dla dużych porcji danych.

4.4. Kryteria jakościowe

W skład kryteriów jakościowych wchodziły następująco: wymaganie znajomości przez programistę języka SQL, zabezpieczenie przed atakami typu „SQL Injection" [16], wsparcie dla dialektów SQL oraz wspomaganie programisty poprzez sprawdzanie składni SQL. Kryteria tego typu najczęściej wynikały bezpośrednio z dokumentacji technicznej dostępnej na stronach autorów.

5. Aplikacja testowa

W celu zbadania kryteriów ilościowych, konieczne było napisanie aplikacji do komunikacji z bazą danych. Zadaniem aplikacji było realizowanie przypadków testowych, mierzenie czasu wykonania danej operacji SQL, mierzenie wykorzystania pamięci RAM oraz obserwacja procentowego wykorzystania procesora przy komunikacji z bazą danych. W pierwszym kroku stworzono interfejs dla języka Java, który definiował, jakie metody mają zostać zaimplementowane przez badane narzędzia do komunikacji z bazą danych. Metody z interfejsu Java reprezentowały każdą z operacji typu CRUD. Struktura implementacji dla stworzonego interfejsu przedstawia Rysunek 2.



Rysunek 2: Diagram przedstawiający klasy implementujące interfejs DaoOperation.

Cała aplikacja została napisana w języku Java z wykorzystaniem szkieletu programistycznego Spring Boot w wersji 2.4.5 [17]. Dane testowe były generowane w aplikacji z wykorzystaniem biblioteki JavaFaker w wersji 1.0.2. Kod aplikacji testowej mierzący czas wykonania operacji nie powinien wpływać na wynik samego testu, więc wykorzystano podejście programowania aspektowego dzięki projektowi Spring AOP (ang. Aspect-Oriented Programming). Aby całość działała poprawnie, stworzono dodatkową adnotację języka Java (measureExecutionTime), która była wykorzystywana do oznaczania metod, podlegających pomiarowi. Zadaniem stworzonej adnotacji był pomiar czasu wykonania danej metody. Każda z metod dla danego badanego narzędzia odpowiedzialnego za komunikację została oznaczona taką adnotacją na poziomie dostępu do danych (ang. Data Access Object - DAO). Zadaniem omawianej adnotacji było pobranie czasu systemowego przed wykonaniem operacji na bazie danych i bezpośrednio po wykonaniu. Obliczając różnice zmierzonych czasów otrzymano wynik w postaci czasu wykonania samej operacji na bazie danych. Dla zmierzenia średniego procentowego wykorzystania procesora oraz średniego wykorzystania pamięci RAM, wykorzystano oprogramowanie JProfiler [18]. Oprogramowanie to umożliwia śledzenie aktywności GC, dzięki czemu można wykluczyć te pomiary podczas których GC rozpoczął swoje działanie. W badaniu uwzględniono tylko te pomiary dla których GC rozpoczął i zakończył swoje działanie bezpośrednio przed pomiarem. Dodatkowo GC nie uruchamiał się, aż do zakończenia samego pojedynczego pomiaru. Wszystkie dane pozyskane podczas badań były zapisywane do pliku, który później posłużył do obliczenia wartości średnich arytmetycznych. Dzięki zewnętrznemu oprogramowaniu możliwe było poznanie charakterystyki wykorzystania procesora w przypadku większych zapytań SQL.

6. Wyniki

W pierwszej kolejności zostały wyliczone wagi dla każdego z kryterium na podstawie danych pozyskanych poprzez przeprowadzenie ankiety. Wagi zostały przedstawione w Tabeli 3. Tabela 4 przedstawia wyniki 3 pomiarów dla operacji odczytywania z bazy danych. Tabela 5 przedstawia te same pomiary tym razem dla operacji dodawania rekordów do bazy danych. Tabela 6 przedstawia pomiary dla aktualizowania rekordów, a Tabela 7 przedstawia pomiary dla usuwania rekordów z bazy danych.

Tabela 3: Uzyskane wagi dla poszczególnych kryteriów

Nr	Kryterium	Waga
1	Szybkość wykonywania zapytań SQL	4,65
2	Stopień wykorzystania procesora	3,69
3	Stopień wykorzystania pamięci RAM	3,46
4	Popularność technologii	3,08
5	Potrzeba znajomości SQL	2,77
6	Niepodatność na ataki typu „SQL Injection”	4,19
7	Wykorzystanie dialektów	3,12
8	Sprawdzanie składni SQL	2,77

w procesie ankietowania

Tabela 4: Pomiary dla operacji odczytu z bazy danych (liczba powtórzeń 50)

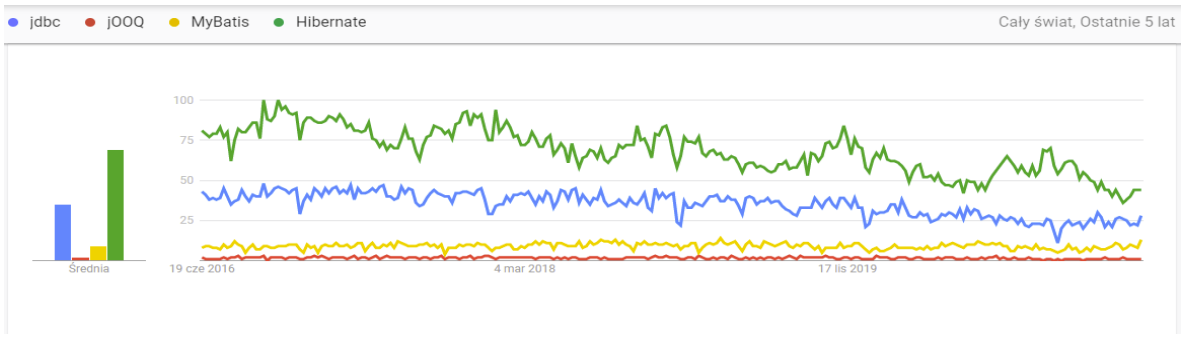
Zapytanie SQL	SELECT * FROM Studenci			
Mierzone parametry	Średni czas wykonania operacji [ms] Średnie zużycie pamięci RAM [MB] Średnie wykorzystanie procesora [%]			
Liczba rekordów	JDBC	jOOQ	MyBatis	Hibernate
1	1,4191 29,25 0,80	2,8537 30,55 2,32	2,0039 29,32 2,32	4,4142 30,52 3,21
10	1,8799 29,26 0,77	3,0520 30,82 1,94	2,3297 29,81 3,43	5,0645 30,56 3,90
100	2,7873 29,80 1,17	4,9258 30,89 2,35	3,9625 29,82 4,52	7,3024 30,77 4,27
1 000	7,1629 30,10 1,17	10,4747 31,57 2,75	11,2639 30,93 5,85	14,4853 32,10 4,29
10 000	28,0151 38,38 2,34	43,8685 46,28 5,07	41,3272 43,34 7,02	54,0078 49,02 7,04

Tabela 5: Pomiary dla operacji dodawania rekordów do bazy danych (liczba powtórzeń 20)

Zapytanie SQL	INSERT INTO Studenci VALUES (…)			
Mierzone parametry	Średni czas wykonania operacji [ms] Średnie zużycie pamięci RAM [MB] Średnie wykorzystanie procesora [%]			
Liczba rekordów	JDBC	jOOQ	MyBatis	Hibernate
1	2,3616 30,27 0,37	2,8759 30,66 1,17	3,009 30,32 0,37	4,9857 30,48 0,37
10	8,7373 30,28 0,40	15,332 31,18 1,94	9,3764 30,38 1,17	23,3646 31,02 0,49
100	34,2451 30,38 0,49	52,0805 32,76 1,94	35,1963 32,42 1,94	132,9326 33,11 5,85
1 000	202,2956 33,69 1,59	247,1499 50,99 7,42	222,75 47,58 5,1	580,8474 56,97 12,5
10 000	1793,546 79,85 10,10	2058,063 237,4 16,80	1893,3111 204,2 11,23	4573,8852 290,40 17,57

Tabela 6: Pomiary dla operacji aktualizowania rekordów w bazie danych (liczba powtórzeń 20)

Zapytanie SQL	UPDATE Studenci Studenci SET imie = -[…] WHERE id = […]			
Mierzone parametry	Średni czas wykonania operacji [ms] Średnie zużycie pamięci RAM [MB] Średnie wykorzystanie procesora [%]			
Liczba rekordów	JDBC	jOOQ	MyBatis	Hibernate
1	3,6718 30,25 0,37	4,9383 30,42 0,37	4,6767 30,30 0,37	4,3478 30,22 1,17
10	10,3425 30,26 0,40	12,0667 30,66 0,77	12,7007 30,32 0,77	13,6078 31,25 2,73
100	41,4706 30,85 1,17	39,4382 32,25 1,17	45,0012 30,87 0,79	123,5245 34,60 3,54
1 000	217,625 34,20 3,90	214,275 47,96 9,77	222,6127 35,79 4,30	621,0228 67,74 17,30
10 000	1667,94 78,66 8,90	1812,87 196,40 17,57	1781,577 91,05 14,62	4690,511 323,50 25,34



Rysunek 3: Wykres przedstawiający popularność wyszukiwania danego narzędzia w przeglądarce Google.

Tabela 7: Pomiary dla operacji usuwania rekordów z bazy danych (liczba powtórzeń 20)

Zapytanie SQL	DELETE FROM Studenci WHERE id = […]			
Mierzone parametry	Średni czas wykonania operacji [ms] Średnie zużycie pamięci RAM [MB] Średnie wykorzystanie procesora [%]			
Liczba rekordów	JDBC	jOOQ	MyBatis	Hibernate
1	5,1196 29,25 0,39	6,6558 30,15 0,40	6,6676 30,31 0,40	10,3639 30,48 0,34
10	10,1224 29,25 0,77	13,1132 30,67 0,40	12,9217 30,33 1,17	36,5221 30,50 0,36
100	34,4795 30,31 0,79	43,0083 30,72 0,77	36,1948 30,38 2,72	126,9967 30,57 0,39
1 000	226,9031 33,69 1,94	259,0934 33,58 4,29	245,3846 34,77 5,06	567,6597 36,07 5,74
10 000	2010,299 62,36 11,50	2208,017 133,90 14,30	2049,5607 80,29 11,87	4247,6016 285,8 17,80

Tabele od 8 do 11 przedstawiają wyniki badań pod względem kryteriów jakościowych. Tabela 8 przedstawia czy dane narzędzie wymaga znajomości SQL od programisty. Tabela 9 przedstawia czy dane rozwiązanie zabezpiecza przed atakami typu „SQL Injection”.

Tabela 8: Potrzeba znajomości SQL

Potrzeba znajomości SQL			
JDBC	jOOQ	MyBatis	Hibernate
TAK	NIE	TAK	NIE

Tabela 9: Zapewnienie mechanizmu ochrony przed atakami typu „SQL Injection"

Ochrona przed atakami typu „SQL Injection”			
JDBC	jOOQ	MyBatis	Hibernate
NIE	TAK	TAK	TAK

Tabela 10: Wsparcie dla dialektów

Wsparcie dla dialektów			
JDBC	jOOQ	MyBatis	Hibernate
NIE	TAK	NIE	TAK

Tabela 11: Sprawdzanie poprawności składni SQL

Sprawdzanie składni SQL			
JDBC	jOOQ	MyBatis	Hibernate
NIE	TAK	NIE	TAK

Tabela 10 przedstawia czy dane narzędzie wprowadza wsparcie dla dialektów. Tabela 11 przedstawia, które z rozwiązań sprawdza poprawność składni SQL.

6.1. Punktacja

Tabela 12 prezentuje punktację, którą uzyskały badane narzędzia w kontekście poszczególnych, określonych wcześniej kryteriów. Do ostatecznej oceny na poziomie kryterium zostały uwzględnione dodatkowo wagi. Po zsumowaniu wszystkich punktów w ramach danego rozwiązania otrzymano finalną punktację, na podstawie której wskazano najbardziej efektywne podejście do pracy z bazami danych w języku Java.

Tabela 12: Punktacja technologii bez uwzględnienia wag

Nr kryterium	JDBC	jOOQ	MyBatis	Hibernate
1	3	1	2	0
2	3	0	2	1
3	3	1	2	0
4	2	0	1	3
5	0	2	0	2
6	0	2	2	2
7	0	2	0	2
8	0	2	0	2
Suma	11	10	9	12

Tabela 13: Finalna suma punktów z uwzględnieniem wag

Tech.	JDBC	jOOQ	MyBatis	Hibernate
Suma	41,56	33,81	35,06	38,63

7. Wnioski

Celem badań było wykazanie, które narzędzie do komunikacji z relacyjną bazą danych jest najefektywniejsze. Rezultat ten udało się osiągnąć przeprowadzając niniejsze badania. Należy zauważyć, że wprowadzenie wag uzyskanych w procesie ankietowania, znacząco zmienia finalną punktację i końcową ocenę badanych technologii. Jeżeli kryteria zostałyby potraktowane jako równoważne, najefektywniejszym narzędziem do pracy z bazami danych zostałby Hibernate, następne w kolejności byłby JDBC, potem jOOQ i na końcu MyBatis. Przy wprowadzeniu systemu wag, wyniki narzędzi zostały rozpatrzone pod kątem najważniejszych kryteriów i to one w głównej mierze wpływały na końcową ocenę. Przy uwzględnieniu wag zgodnie z tabelą 13 najefektywniejszym rozwiązaniem zostaje JDBC, następnie Hibernate, potem MyBatis i jOOQ. Dla ankietowanych najważniejszymi kryteriami były: szybkość wykonywania zapytań SQL oraz niepodatność na ataki typu „SQL Injection". Za to najmniej ważnymi kryteriami zostały: potrzeba znajomości SQL oraz sprawdzanie składni SQL.

Dodatkowo analizując wyniki stwierdzono, że operacją potrzebującą najwięcej zasobów komputera, w którego skład wchodzą: wykorzystanie procesora oraz pamięci RAM jest operacja dodawania rekordów do bazy danych. Za to operacją potrzebującą najmniej zasobów jest operacja pobierania rekordów z bazy danych.

Literatura

[1] Główny urząd statystyczny, społeczeństwo informacyjne w Polsce w 2020 roku, https://stat.gov.pl/obszary-tematyczne/nauka-i-technika-spoleczenstwo-informacyjne/spoleczenstwo-informacyjne/spoleczenstwo-informacyjne-w-polsce-w-2020-roku,1,14.html , [17.09.2021].

[2] J. Desjardins, How much data is generated each day?, World Economic Forum 2019, https://www.weforum.org/agenda/2019/04/how-much-data-is-generated-each-day-cf4bddf29f/ , [17.09.2021].

[3] Dokumentacja programistyczna ODBC, https://docs.microsoft.com/en-us/sql/odbc/reference/odbc-programmer-s-reference?view=sql-server-ver15 , [19.09.2021].

[4] Dokumentacja programistyczna JDBC, https://docs.oracle.com/cd/E11882_01/java.112/e16548/toc.htm , [19.09.2021].

[5] P. Błoch, M. Wojciechowski, Analiza porównawcza technologii odwzorowania obiektowo-relacyjnego dla aplikacji Java. XIII Konferencja PLOUG: Systemy informatyczne. Projektowanie, implementowanie, eksploatowanie, Zakopane (2007).

[6] K. Jóźwicka, M. Mitrus, Hybrydowe metody pracy z bazami danych w aplikacjach JEE. Journal of Computer Sciences Institute, (2019) 12.

[7] M. Grzesińska, M. Waszczyńska, B. Pańczyk, JEE DATABASE APPLICATIONS PERFORMANCE. Informatyka, Automatyka, Pomiary W Gospodarce I Ochronie Środowiska, 6(4) (2016) 73-76.

[8] Liczba wyszukiwań badanych narzędzi w serwisie Google, https://trends.google.pl/trends/explore?q=jdbc,jooq,mybatis,hibernate , [20.09.2021].

[9] Dokumentacja techniczka Spring Data, https://docs.spring.io/spring-data/jpa/docs/current/reference/html/#reference ,[20.09.2021].

[10] Dokumentacja techniczna systemu bazodanowego MySQL, https://dev.mysql.com/doc/ , [17.09.2021].

[11] G. Reese, Database Programming with JDBC and JAVA, O'Reilly Media Inc, 2000.

[12] K. Siva Prasad Reddy, Working with JOOQ. In: Beginning Spring Boot 2, Apress, Berkeley CA (2017) 71-82.

[13] K. Siva Prasad Reddy, Java Persistence with MyBatis3, Packt Publishing Ltd, 2013.

[14] Dokumentacja Java Persistence API, https://javadoc.io/doc/javax.persistence/javax.persistence-api/latest/index.html , [20.09.2021].

[15] P. T. Fisher, B. D. Murphy, Spring persistence with Hibernate, Apress (2010).

[16] J. Clarke-Salt, SQL injection attacks and defense, Elsevier, 2009.

[17] C. Walls, Spring Boot in action, Manning Publications, 2016.

[18] N. Kumari, R. Kumar, Profiling JVM for AI Applications Using Deep Learning Libraries, In Machine Learning for Predictive Analysis, Springer, Singapore (2021) 395-404.