JOURNAL of COMPUTER SCIENCES INSTITUTE

JCSI 12 (2019) 167-171
Wysłane: 2019-04-30
Przyjęte: 2019-05-06

Hybrydowe metody pracy z bazami danych w aplikacjach JEE

Katarzyna Jóźwicka*, Mariusz Mitrus*
Politechnika Lubelska, Instytut Informatyki, Nadbystrzycka 36B, 20-618 Lublin, Polska

Streszczenie. W artykule przedstawiono możliwości hybrydyzacji metod pracy z bazami danych w aplikacjach JEE. Do przeprowadzenia badań wykorzystano aplikacje wykonane w oparciu o interfejs JDBC, Hibernate oraz Spring. Analiza wydajności aplikacji dotyczyła czasu wykonywania oraz zużycia pamięci RAM dla podstawowych operacji CRUD w bazie danych.

Słowa kluczowe: JEE; Hibernate; Spring; JDBC; Bazy danych; wydajność

*Autor do korespondencji.

Adresy e-mail: katrzyna.jozwicka@pollub.edu.pl, mariusz.mitrus@pollub.edu.pl

Hybrid methods of working with databases in JEE applications

Katarzyna Jóźwicka*, Mariusz Mitrus*
Institute of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract. The article presents the research on hybridization of methods of working with databases in JEE applications. The test applications were made based on the JDBC interface, Hibernate and Spring framework. Application performance analysis covered the execution time and RAM usage for basic CRUD operations in the database.

Keywords: JEE; Hibernate; Spring; JDBC; Databases; performance

*Corresponding author.

E-mail addresses: katrzyna.jozwicka@pollub.edu.pl, mariusz.mitrus@pollub.edu.pl

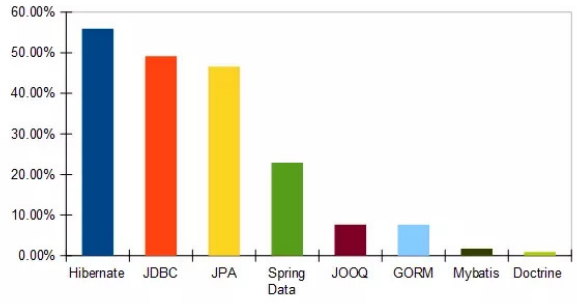
1. Wstęp

Rozwój i powszechność programowania obiektowego sprawiły, że programiści zmuszeni zostali do opracowania rozwiązania, które sprawnie konwertuje bazy danych na obiekty i odwrotnie. Do tej pory jednak nie udało się wykreować „powszechnie akceptowanej koncepcji obiektowej bazy danych" [1, 2]. Oczywiście podejmowano próby stworzenia takich baz danych np. z wykorzystaniem repozytorium XML lub dedykowanych rozszerzeń dla języka Java, czy C# [2]. Jednak pomimo upływu lat model relacyjny jest nadal podstawowym modelem organizacji danych [3].

Mapowanie obiektowo-relacyjne (ang. Object-Relational Mapping ORM) konwertuje dane między relacyjną bazą danych, a obiektami w aplikacji. Rozwiązanie oparte na ORM staje się bardziej użyteczne wraz ze wzrostem rozmiaru i złożoności projektu [4].

14 sierpnia 2010 roku została zaprezentowana biblioteka jOOQ (ang. Java Object Oriented Querying), która nie realizuje technologii mapowania. Tworząc bibliotekę jOOQ założono, że to SQL powinien być najważniejszy, jeśli chodzi o integrację z bazą danych. Dlatego też biblioteka ta nie wprowadza nowego języka zapytań (jak np. HQL w Hibernate), ale pozwala konstruować zwykły SQL z obiektów jOOQ i kodu wygenerowanego ze schematu baz danych [6].

Pomimo zalet JOOQ, najbardziej popularnym frameworkiem do pracy z bazami danych jest ciągle Hibernate. Jego największą zaletą, oprócz szybkości i możliwości pracy z wieloma bazami danych, jest fakt, że można go wykorzystać zarówno w małych, jak i dużych projektach [7]. Kolejnym według popularności interfejsów jeżyka JAVA króluje JDBC, zgodnie z rysunkiem numer 1.



Rys. 1. Popularność interfejsów bazodanowych [8]

2. Cel i teza badań

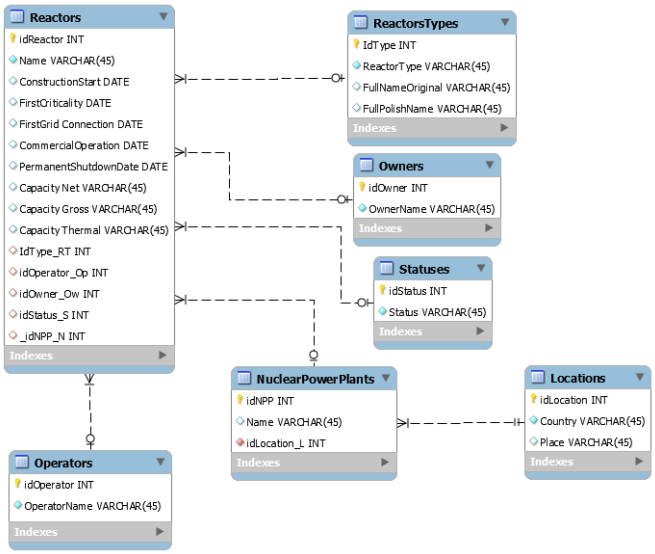
Celem badań była analiza możliwości hybrydyzacji metod pracy z bazami danych w celu zwiększenia wydajności pracy z danymi w aplikacjach JEE.

Za tezę badawczą przyjęto:

Hybrydyzacja metod pracy z bazami danych w aplikacjach JEE pozwoli przyspieszyć pracę z danymi w stosunku do rozwiązań standardowych.

3. Scenariusze i metoda badań

Na potrzeby badań utworzono bazę reaktorów uzupełnioną rzeczywistymi informacjami na temat wszystkich światowych 786 reaktorów (Rys. 2). Wszystkie dane pobrano ze strony http://www.world-nuclear.org/ [9]. Następnie dodano ponad 1000000 nowych, wygenerowanych losowo rekordów.



Rys.2. Diagram związków encj (ERD) bazy danych

Platforma oraz serwer zostały utworzone na dwóch stanowiskach pomiarowych, których parametry prezentuje tabela 1. Na serwerze została zainstalowana relacyjna baza danych MySql.

Tabela 1. Parametry stanowisk pomiarowych

Nazwa stanowiska	CPU	Pamięć RAM	Karta Graficzna	System
Stanowisko numer 1	Intel i7-3612QM, 4x2.1GHz	8.00GB	Intel Graphics 4000 + Nividia Geforce GT630M	Windows 7, 64 bitowy
Stanowisko numer 2	Intel i5 3337U	8.00GB	Intel Graphics 4000	Windows 7, 64 bitowy

Do badań wykorzystane zostały aplikacje testowe, które wykonywały klasyczne operacje CRUD zaimplementowane z wykorzystaniem JDBC, Hibernate i Spring. Badania odbyły się z użyciem zapytań: INSERT INTO, UPDATE, DELETE oraz SELECT, których użycie w scenariuszach prezentuje tabela 2. Po każdym teście przeprowadzonym zgodnie z danym scenariuszem, baza była przywrócona do stanu wcześniejszego. Dzięki temu dostępna była identyczna baza dla każdego scenariusza, a wyniki były w większym stopniu miarodajne. Każdy ze scenariuszy powtórzono 4 razy.

Tabela 2. Przykładowe scenariusze wykorzystane do badań

Numer scenariusza	Rodzaj zapytania	Opis
S1.1	SELECT	Odczyt nazw 5000 najstarszych reaktorów
S1.2	SELECT	Odczyt nazw 50000 najstarszych reaktorów
S1.3	SELECT	Odczyt nazw 100000 najstarszych reaktorów
S1.4	SELECT	Odczyt nazw 300000 najstarszych reaktorów
S1.5	INSERT INTO	Zapis 5000 reaktorów o do tabeli REACTORS z nazwą 'TEST'
S1.6	INSERT INTO	Zapis 50000 reaktorów do tabeli REACTORS z nazwą 'TEST'
S1.7	INSERT INTO	Zapis 100000 reaktorów do tabeli REACTORS
S1.8	INSERT INTO	Zapis 300000 reaktorów do tabeli REACTORS z nazwą 'TEST'

Metoda pomiaru wydajności opierała się na pomiarze zużycia pamięci RAM oraz czasu realizacji danego scenariusza (Przykład 1).

Przykład 1. Pomiar wydajności

```
long beforeUsedMem=Runtime.getRuntime().totalMemory()-
Runtime.getRuntime().freeMemory();//pomiar zużycia RAM
long elapsedTime=0;
long afterUsedMem=Runtime.getRuntime().totalMemory()-
Runtime.getRuntime().freeMemory();//Pomiar RAM
long actualMemUsed=afterUsedMem-beforeUsedMem;
out.println("Zużycie RAM[MB]: "+actualMemUsed/1000000 );
//konwersja na milisekundy:
float elapsedTimeMili
   =TimeUnit.NANOSECONDS.toMillis(elapsedTime);
out.println("<p>Czas w milisekundach: "+elapsedTimeMili);
```

W celu utworzenia aplikacji hybrydowych, wykonano najpierw scenariusze z wykorzystaniem standardowych metod. Na podstawie otrzymanych wyników okazało się, że dla rozważanej bazy danych, najlepszym rozwiązaniem pod względem wydajności dla scenariuszy od 1.1 do 1.4 oraz od 1.16 do 1.20 jest interfejs JDBC, a dla innych okazał się Hibernate. Jeśli chodzi o złożone zapytania, sytuacja przedstawiła się podobnie (tabela 3 i 4).

Tabela 3. Zestawienie najlepszych rozwiązań standardowych dla scenariuszy od 1.1 do 1.32

Liczba rekordów\ Rodzaj zapytania	5000	50000	100000	300000
SELECT	JDBC	JDBC	JDBC	JDBC
INSERT INTO	Hibernate	Hibernate	Hibernate	Hibernate
UPDATE	Hibernate	Hibernate	Hibernate	Hibernate
DELETE	Hibernate	Hibernate	Hibernate	Hibernate

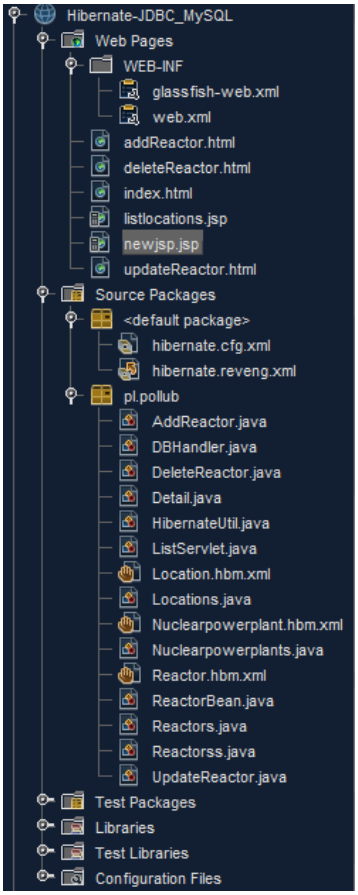
Tabela 4. Zestawienie najlepszych rozwiązań standardowych dla scenariuszy od 2.1 do 2.32

Liczba rekordów\ Rodzaj zapytania	5000	50000	100000	300000
SELECT	Spring+JDBC/JDBC	Spring+JDBC/JDBC	Spring+JDBC/JDBC	Spring+JDBC/JDBC
INSERT INTO	Hibernate	Hibernate	Hibernate	Hibernate
UPDATE	Hibernate	Hibernate	Hibernate	Hibernate
DELETE	Hibernate	Hibernate	Hibernate	Hibernate

Na podstawie wyników przedstawionych w tabeli 3 i 4 zaproponowano rozwiązania hybrydowe.

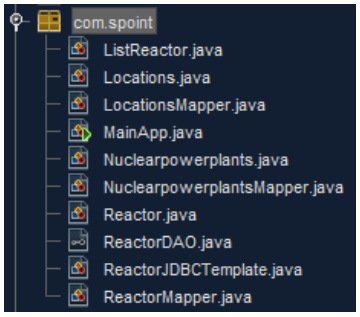
4. Programy hybrydowe

Hybrydyzacja JDBC i Hibernate (Rys. 3) została zaimplementowana w postaci aplikacji internetowej z możliwością wyboru żądanego typu zapytania na stronie *index.html*. Dodatkowe akcje na bazie danych obsługują odpowiednio strony *updateReactor.html*, *deleteReactor.htlm* oraz *addReactor.html*. *SelectReactor.html* jest stroną z formularzem wykorzystującym klasę *ListServlet*, która wywołuje metody *loadReactors* klasy *DBHandler,* dokładnie jak w przypadku programu korzystającego tylko z połączenia JDBC. Modyfikacja zapytań typu SELECT polegała na zmianie kodu w metodzie *loadReactors*.



Rys 3. Struktura programu hybrydowego Hibernate+JBDC

Hybrydyzacja JDBC, Spring + JBDC i Hibernate polegała na dodaniu do kopii hybrydy JDBC i Hibernate metod z aplikacji Spring + JDBC. Wszystkie klasy i metody JDBC i Hibernate są identyczne jak w przypadku wcześniejszej hybrydy. Służą one do zapytań typu SELCECT w scenariuszach od 1.1 do 1.32, oraz typu INSERT INTO, DELETE i UPDATE dla scenariuszy od 2.1 do 2.32. Pliki programu Spring + JDBC zostały dodane w oddzielnym pakiecie *com.spoint* (Rys. 4) oraz do pliku konfiguracyjnego *Beans.xml*. W celu wykonania powiązanych zapytań typu SELECT i wyświetlenia ich wyników, konieczne było utworzenie dodatkowej klasy *ListReactor*. Zmiana zapytania została wykonana w metodzie *loadReactor()* klasy *ReactorJDBCTemplate.*



Rys. 4. Pakiet *com.spoint* dla Spring+JDBC

5. Wyniki testów

Średnie wartości pomiarów dla programów hybrydowych przedstawiają tabele 5, 6 oraz rysunki 5, 6 dla scenariuszy z zapytaniem typu SELECT. Program hybrydowy JDBC i Hibernate okazał się najbardziej efektywny.

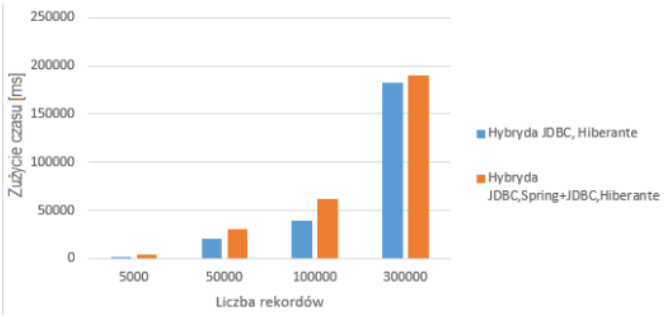
Tabela 5. Średnie wyniki testów programów hybrydowych dla scenariuszy od 1.1 do 1.32

Rodzaj zapytania	Liczba rekordów	Hybryda JDBC, Hiberante		Hybryda JDBC,Spring+JDBC,Hiberante	
		RAM[MB]	Czas[ms]	RAM[MB]	Czas[ms]
Select	5000	19	2002	46	3546
	50000	31	20363	150	31012
	100000	56	38770	222	61513
	300000	113	182536	386	190562
Insert into	5000	27	2674	27	2867
	50000	95	24943	96	24820
	100000	77	44855	76	45159
	300000	103	158848	107	156538
Update	5000	71	8014	72	7713
	50000	93	42391	98	42106
	100000	105	54207	110	54797
	300000	115	124693	128	126575
Delete	5000	44	4731	43	5655
	50000	34	34855	35	34624
	100000	85	75506	85	80413
	300000	92	281957	91	289255

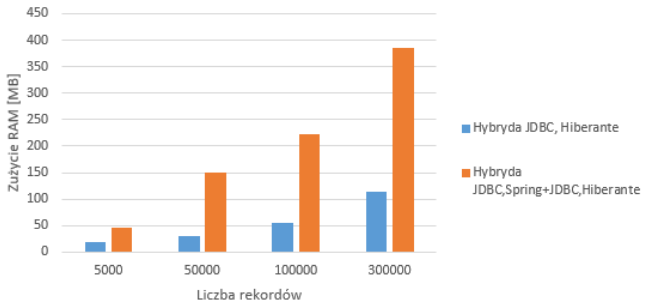
Tabela 6. Średnie wyniki testów dla programów hybrydowych dla scenariuszy od 2.1 do 2.32

Rodzaj zapytania	Liczba rekordów	Hybryda JDBC, Hiberante		Hybryda JDBC,Spring+JDBC,Hiberante	
		RAM[MB]	Czas[ms]	RAM[MB]	Czas[ms]
Select	5000	23	6302	264	6664
	50000	40	56299	729	57077
	100000	65	161474	1170	168300
	300000	149	716576	2194	730326
Insert into	5000	29	3172	27	3207
	50000	93	27406	91	27477
	100000	82	49294	77	49098
	300000	104	174231	104	175689
Update	5000	78	8184	77	7636
	50000	95	45361	93	46025
	100000	103	59102	97	60617
	300000	113	137886	118	137697
Delete	5000	51	27617	51	33076
	50000	43	204162	43	190210
	100000	102	412749	113	424652
	300000	120	1572076	119	1582120

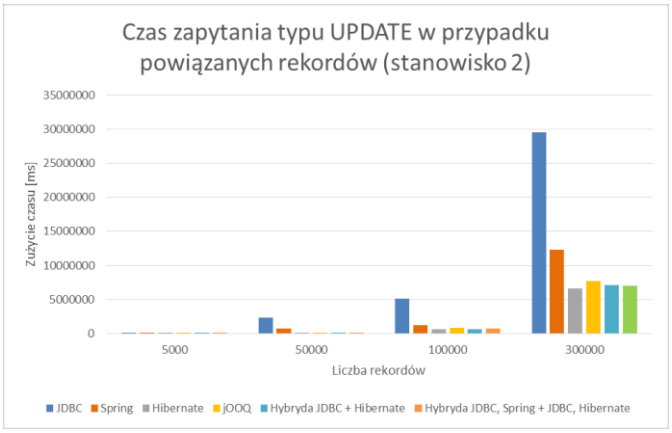
Rozwiązania hybrydowe są znacząco lepsze od standardowych metod pracy z bazami danymi, co przedstawiają rysunki 7 i 8. Średnie wyniki programu hybrydowego JDBC+Hibernate w porównaniu do średniej standardowych metod prezentuje tabela 7. Dla hybrydy Spring+JDBC i Hibernate wyniki przedstawia tabela 8.



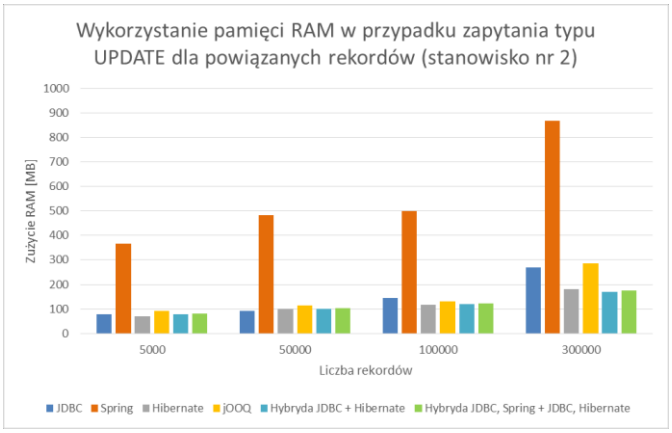
Rys. 5. Średni czas zapytań typu SELECT dla scenariuszy 1.1-1.32



Rys. 6. Średnie wykorzystanie pamięci RAM w przypadku scenariuszy 1.1-1.32



Rys. 7. Średni czas zapytania typu UPDATE w przypadku scenarisuzy 2.1-2.32 (Stanowisko numer 2)



Rys. 8. Średnie wykorzystanie pamięci RAM w przypadku zapytania typu UPDATE dla scenarisuzy 2.1-2.32(Stanowisko numer 2)

Tabela 7. Poprawa (w %) średnich wyników dla JDBC+Hibernate w porównaniu ze średnimi wynikami standardowych metod

Typ zapytania	Zużycie RAM [MB]	Zużycie czasu [ms]
SELECT	34%	6%
INSERT INTO	2%	44%
UPDATE	13%	39%
DELETE	-18%	37%

Tabela 8. Poprawa (w %) średnich wyników dla Spring+JDBC+Hibernate w porównaniu ze średnimi standardowych metod

Typ zapytania	Zużycie RAM [MB]	Zużycie czasu [ms]
SELECT	55%	6%
INSERT INTO	3%	43%
UPDATE	11%	39%
DELETE	88%	33%

6. Wnioski

Przeprowadzone badania wykazały, że w pracy z bazami danych dużą rolę w optymalizacji, zwiększeniu szybkości dostępu do rekordów może odgrywać hybrydyzacja.

Na podstawie uzyskanych wyników wybrano najlepsze rozwiązania i stworzono dwie aplikacje hybrydowe: JDBC i Hibernate, a także JDBC, Spring+JDBC i Hibernate. Poprzez analizę uzyskanych rezultatów stwierdzono, że najlepszym rozwiązaniem jest aplikacja hybrydowa łącząca JDBC i Hibernate. Zastosowanie JDBC dla zapytań typu SELECT, Hibernate dla INSERT, UPDATE i DELETE pozwoliło ograniczyć zużycie pamięci RAM, a także zmniejszyć czas potrzebny do zwrócenia wyników zapytań.

Wykorzystując najlepsze rozwiązania dla poszczególnych typów zapytań, można więc znacząco poprawić wydajność aplikacji, umożliwić użytkownikowi sprawniejszy dostęp do bazy, niezależnie od tego, jakiego rodzaju danych potrzebuje. Oczywiście takie rozwiązanie wymaga od programisty dogłębnego zapoznania się z dostępnymi na rynku rozwiązaniami oraz samą bazą danych, wykonania odpowiednich badań i ich późniejszej analizy. Jednak włożony na początku wysiłek, może później znaczącą skrócić pracę nad późniejszą optymalizacją gotowej aplikacji.

W prowadzeniu badań bardzo pomocne są wyniki uzyskiwane przez innych badaczy. Niniejszy artykuł stanowi rozwinięcie badań prowadzonych w publikacji [10, 11], ale wyniki nie mogą być bezpośrednio porównane. W niniejszej pracy zwiększono liczbę scenariuszy oraz liczbę rekordów. W obu pracach, w przypadku stosowania rozwiązań standardowych, Hibernate okazał się najszybszą metodą dla zapytań INSERT a JDBC dla SELECT. Proponowanym (ale nie testowanym) rozwiązaniem w pracy [10] jest zastosowanie aplikacji hybrydowej z Hibernate dla większości zapytań, lecz z użyciem jOQQ, zamiast JDBC, jak w pracy niniejszej.

Istotnym elementem jest struktura samej bazy danych, serwerów baz danych oraz parametrów środowiska. Na przykład w pracy [10] testy prowadzono dla JBOSS oraz MSSQL. W niniejszej pracy wykorzystano WampServer

z MySQL, i najbardziej optymalne wyniki uzyskano dla JDBC i Hibernate.

Wyniki przedstawione w pracy pozwoliły potwierdzić, postawiona na wstępie tezę.

Hybrydyzacja metod pracy z bazami danych w aplikacjach JEE pozwoli przyspieszyć pracę z danymi w stosunku do rozwiązań standardowych.

Literatura

[1] Oracle, Java™ EE at a Glance, https://www.oracle.com/ technetwork/ java/javaee/overview/index.html, [maj 2018].

[2] Rosiek Z.: Mapowanie obiektowo-relacyjne (ORM) – czy to dobra idea? Zeszyty Naukowe Rok 4 Zeszyt 4, Warszawa 2010, s. 99-112.

[3] Teamquest, Relacyjne bazy danych – dlaczego warto je znać?, https://teamquest.pl/blog/461_relacyjne-bazy-danych-dlaczego-warto-je-znac, [czerwiec 2018].

[4] Object Oriented Application Cooperation Methods with Relational Database (ORM) based on J2EE Technology, P. Ziemniak , B. Sakowicz , A. Napieralski, 2007 9th International Conference - The Experience of Designing and Applications of CAD Systems in Microelectronics, 2007.

[5] NoSQL2: SQL to NoSQL Databases, J. Adriana, M. Holanda, World Conference on Information Systems and Technologies, 2018.

[6] Working with JOOQ, K. Siva Prasad Reddy, 2017.

[7] Javapipe, 10 Best Java Web Frameworks to Use in 2019, https://javapipe.com/blog/best-java-web-frameworks/, [maj 2018].

[8] Bill William, Are Spring and Hibernate still popular?, https://www.quora.com/Are-Spring-and-Hibernate-still-popular, [czerwiec 2018].

[9] World Nuclear Association and IAEA Power Reactor Information System, Information Library - World Nuclear Association , http://www.world-nuclear.org/information-library/facts-and-figures/reactordatabase-search.aspx, [maj 2018].

[10] Grzesińska M., Waszczyńska M.: Analiza porównawcza technologii ORM stosowanych w internetowych aplikacjach JEE. Praca magisterska, Politechnika Lubelska, 2016.

[11] Grzesińska M., Waszczyńska M., Pańczyk B. Wydajność pracy z bazami danych w aplikacjach JEE. IAPGOŚ- 2016, nr 4, vol. 6, s. 73-76.