JOURNAL of COMPUTER SCIENCES INSTITUTE

JCSI 8 (2018) 235-240
Wysłane: 2018-01-30
Przyjęte: 2018-02-11

Porównanie technologii mapowania obiektowo-relacyjnego danych w frameworku Symfony 3

Karol Sawłuk*, Marek Miłosz

Politechnika Lubelska, Instytut Informatyki, Nadbystrzycka 36B, 20-618 Lublin, Polska

Streszczenie. W artykule przedstawiono rezultaty analizy porównawczej technologii mapowania obiektowo-relacyjnego w frameworku Symfony 3: Doctrine i Propel. Analizę przeprowadzono pod kątem szybkości wykonywania skryptu oraz zużycia pamięci podczas operacji na bazie danych. Analiza pozwoliła wskazać technologię o szybszych i wydajniejszych algorytmach. Technologia Doctrine jest nawet do trzech razy szybsza niż Propel.

Słowa kluczowe: ORM; php; symfony 3; SZRB; doctrine; propel

*Autor do korespondencji.

Adres e-mail: karol.sawluk@gmail.com

Comparison of object-relational data mapping technology in Symfony 3 framework

Karol Sawłuk*, Marek Miłosz

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

Abstract. The article presents the results of a comparative analysis of object-relation mapping technologies in the framework Symfony 3: Doctrine and Propel. The analysis was performed in terms of script execution speed and memory usage during database operations. The analysis allowed to identify the technology with faster and more efficient algorithms. Doctrine is up to three times faster than Propel.

Keywords: ORM; php; symfony 3; DBMS; doctrine; propel

*Corresponding author.

E-mail address: karol.sawluk@gmail.com

1. Wstęp

Gromadzenie informacji w dzisiejszych czasach jest czymś powszechnym. Prawie każdy system informatyczny, nie tylko webowy, korzysta z relacyjnych baz danych. Zdecydowana większość aplikacji posiada i korzysta z baz danych w sposób obiektowy - zaczynając od prostych stron firmowych i kończąc na zaawansowanych systemach zarządzania przedsiębiorstwami. Dzięki tej technice programowania, można konwertować dane między niezgodnymi systemami przy użyciu języków programowania obiektowego. Tworzy to w efekcie "wirtualną bazę danych", która może być używana w języku programowania, która jest najczęściej zwaną technologią ORM (ang. Object-relational mapping).

Mapowanie obiektowo-relacyjne (ORM) w informatyce jest techniką programowania służącą do konwersji danych pomiędzy niekompatybilnymi systemami używającymi języków programowania obiektowego i systemami zarządzania relacyjnymi bazami danych. Dostępne są darmowe i komercyjne pakiety wykonujące mapowanie obiektowo-relacyjne, które często są zawarte w podstawie wykorzystywanego frameworka. Symfony 3 posiada do wyboru dwie technologie ORM: Doctrine i Propel.

Symfony 3 jest frameworkiem, czyli abstrakcją, w której oprogramowanie dostarcza ogólne funkcjonalności i może być selektywnie zmieniane za pomocą dodatkowego kodu użytkownika, dostarczając w ten sposób specyficznego oprogramowania dla aplikacji [1]. Symfony 3 bazuje na wzorcu projektowym MVC (Model-View-Controller), który jest podzielony na trzy części [2]:

- Model - przechowuje dane, które są pobierane zgodnie z poleceniami kontrolera i wyświetlane w widoku.
- Widok - generuje nowe dane wyjściowe dla użytkownika w oparciu o zmiany w modelu.
- Kontroler - może wysyłać polecenia do modelu w celu aktualizacji stanu modelu (np. edytowania dokumentu).

Celem artykułu jest przedstawienie rezultatów testów na relacyjnej bazie danych z użyciem Doctrine i Propel, które mają określić, który ORM jest wydajniejszy.

Teza zawiera się w stwierdzeniu, że technologia Doctrine jest szybsza niż Propel.

2. Technologie mapowania obiektowo-relacyjnego

Wykorzystanie technologii mapowania obiektowo-relacyjnego w procesie tworzenia aplikacji internetowych jest teraz standardem. Kiedy system wymaga pobrania informacji z bazy danych, realizowana jest konkretna sekwencja: nawiązanie połączenia z bazą danych, wysyłanie zapytania SQL, otrzymanie wyniku zapytania oraz zamknięcie połączenia. Tradycyjne pisanie zapytań do bazy, może prowadzić do niejednego problemu np.: zmiany technologii baz danych (MySQL na PostgreSQL) czy zmiany w strukturze baz danych, które wymagają poprawy zapytań w wielu miejscach. Technologia ORM automatyzuje procedurę manipulowania danymi z bazy danych przy użyciu paradygmatu zorientowanego obiektowo [7].

Zautomatyzowanie procesu, pozwoli na rozwiązanie wyżej wymienionych problemów.

2.1. Doctrine

Doctrine to zestaw bibliotek PHP, skupiających się przede wszystkim na trwałości usług i relacyjnych funkcjonalnościach. Jest domyślnie używany w Symfony 3. Doctrine bazuje na wzorcu Data Mapper, który oddziela atrybuty obiektów od pól tabeli, które są w nich utrzymywane. Istotą wzorcu Data Mapper jest klasa, która odwzorowuje lub tłumaczy atrybuty obiektów domeny i/lub metody na pola tabeli bazy danych i odwrotnie. Zadaniem tego wzorca jest zarówno przedstawianie informacji, tworzenie nowych obiektów domeny w oparciu o informacje w bazie danych i aktualizowanie lub usuwanie informacji w bazie danych przy użyciu informacji z obiektów domeny [3].

Model biblioteki Doctrine oparty jest na mechanizmie encji. Encja jest lekkim, trwałym obiektem domeny. Encja opisuje powiązane ze sobą elementy (np. typy danych) wewnątrz modelu biznesowego aplikacji. Model Entity składa się z typów podmiotów i określa związki, które mogą istnieć między przypadkami tych typów podmiotów [9] [11].

W Doctrine relacje między klasami obiektów określa się poprzez asocjacje. Asocjacje umożliwiają wystąpienie jednej instancji obiektu, powodując kolejną czynność w jego imieniu. Ten związek jest strukturalny, ponieważ określa, że obiekty jednego rodzaju są połączone z obiektami innego i nie reprezentują tego samego zachowania [4]. Zamiast pracować z obcymi kluczami w kodzie, programista pracuje z odwołaniami do obiektów, a Doctrine przekonwertuje te odniesienia do kluczy obcych. Odniesienie do pojedynczego obiektu jest reprezentowane przez klucz obcy. Zbiór obiektów reprezentowany jest przez wiele obcych kluczy, wskazując na obiekt posiadający kolekcję.

Asocjacje dzielą się na cztery typy [5]:

- OneToMany - jedna instancja obecnego obiektu ma wiele instancji (odwołań) do odesłanej encji.
- ManyToOne - wiele wystąpień obecnej encji odnosi się do jednego wystąpienia odesłanej encji.
- OneToOne - jedna encja bieżącego podmiotu odnosi się do jednego wystąpienia odesłanej encji.
- ManyToMany - wiele wystąpień obecnej encji odnosi się do wielu wystąpień odesłanej encji.

2.2. Propel

Propel jest wolnym, otwartym źródłem na licencji MIT (ang. Massachusetts Institute of Technology) obiektowo-relacyjnym narzędziem mapującym napisanym w PHP. Jest to również integralna część systemu Symfony [12].

Propel bazuje na wzorcu Active Record, który jest najprostszym wzorcem projektowania bazy danych. Istotą Active Record jest model domeny (ang. Domain Model), w którym klasy ściśle pasują do struktury rekordu bazy danych. Każdy aktywny rekord jest odpowiedzialny za zapisywanie i ładowanie do bazy danych, a także za logikę domeny, która działa na danych. Może to być całość logiki aplikacji, lub część tej logiki, która jest przechowywana w skryptach transakcyjnych (ang. Transaction Scripts) ze wspólnym kodem zorientowanym na dane. Struktura danych Active Record powinna dokładnie odpowiadać strukturze bazy danych - jedno pole w klasie dla każdej kolumny w tabeli [7].

Propel generuje inteligentne klasy Active Record w oparciu o definicję schematów tabel. Obiekty Active Record oferują potężne API do intuicyjnego zarządzania danymi bazy danych. Dla każdej tabeli znajdującej się w schemacie XML, Propel generuje jedną klasę Active Record - nazywaną również klasą Model. Przykłady klas Active Record reprezentują pojedynczy wiersz z bazy danych, zgodnie z regułami wzoru projektowania [7]. To ułatwia tworzenie, edytowanie, wstawianie lub usuwanie pojedynczego wiersza.

3. Metoda badawcza

Celem pracy badawczej było sprawdzenie postawionej następującej tezy:

Technologia Doctrine jest szybsza niż Propel.

Eksperyment polegał na przeprowadzeniu badań na dwóch aplikacjach testowych napisanych w frameworku Symfony 3 - jedna dla Doctrine ORM, druga dla Propel ORM. Aplikacje testowe mają za zadanie zrealizować te same scenariusze testowe. Każdy scenariusz, realizowany w Doctrine i Propel, pozwala na odmierzenie czasu w milisekundach jaki potrzebuje aplikacja w celu wykonania zadania oraz zmierzenie ilości wykorzystywanej pamięci w megabajtach. Opisy scenariuszy zostały przedstawione w tabeli 1.

Tabela 1. Opis scenariuszy testowych

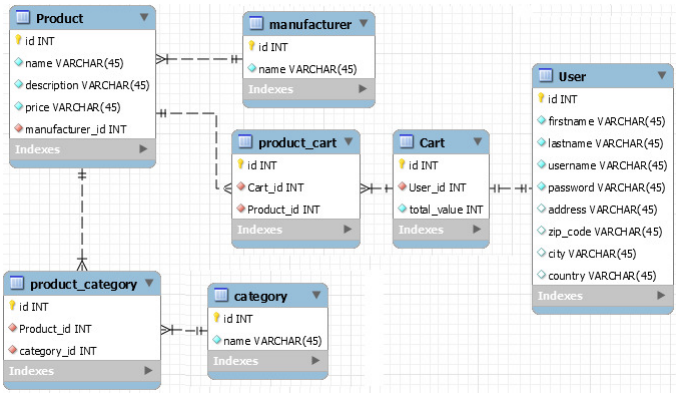
Lp	Opis
1.	Zapis 50, 100, 1000 i 10000 rekordów do relacyjnej bazy danych poprzez utworzenie obiektu reprezentującego poszczególny rekord.
2.	Zapis 50, 100, 1000 i 10000 rekordów powiązanych relacyjnie z trzema obiektami do relacyjnej bazy danych poprzez utworzenie obiektów reprezentujących poszczególny rekord.
3.	Odczyt 50, 100, 1000 i 10000 rekordów z relacyjnej bazy danych.
4.	Odczyt 50, 100, 1000 i 10000 rekordów powiązanych relacyjnie z trzema obiektami z relacyjnej bazy danych.

Do testów obu aplikacji została użyta ta sama baza danych, stworzona na potrzebę zrealizowania testów. Schemat bazy został przedstawiony na rysunku 1. Schemat bazy danych stworzony w programie MySQL WorkBench został przedstawiony na rysunku 1. Testowa baza danych obejmuje tabele:

- User - przechowuje informacje dotyczące użytkowników,
- Cart - pozwala na dodanie produktów do koszyka,
- Product - informacje na temat produktów,
- ProductCategory - tabela łącząca kategorie z produktem,
- Category - zawiera informacje na temat kategorii,
- Manufacturer - zawiera informacje o producencie.

Do pomiaru pamięci oraz czasu, w aplikacji używającej Doctrine, wykorzystano klasę do debugowania (DebugStack), która mierzy czas w mikrosekundach wykonanego skryptu.

W aplikacji z Propel, niestety nie ma tak rozbudowanego narzędzia do debugowania, więc użyto funkcji microtime(), która mierzy czas w mikrosekundach.



Rys. 1. Schemat bazy danych użytej do testów

Obie metody bazują na tej samej funkcji (microtime()), więc wyniki są mierzone w ten sam sposób. Do pomiaru zużytej pamięci poprzez pojedynczy skrypt użyto metody memory_get_usage(), która zwraca w bajtach wielkość pamięci zaalokowanej skryptu.

Opis środowiska testowego na którym zostały przeprowadzone testy, znajduje się w tabeli 2.

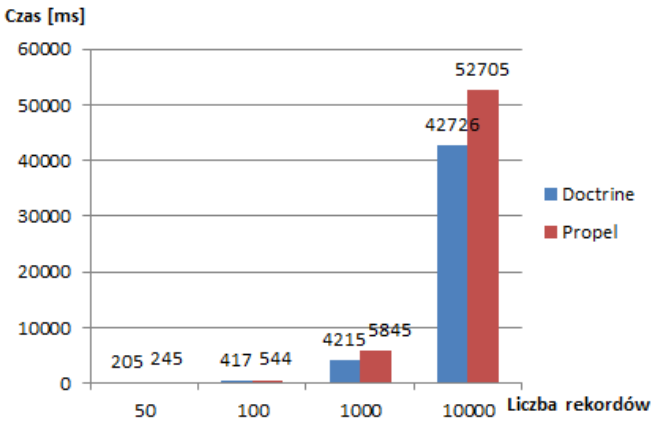
Tabela 2. Dane środowiska testowego

Nazwa	Wartość
System operacyjny	Windows 10
Procesor	Intel i5 4440 3.1GHz
Pamięć RAM	8GB DDR3 1600MHz
Serwer Apache	2.4.27
MySQL	5.7.19
PHP	5.6.30

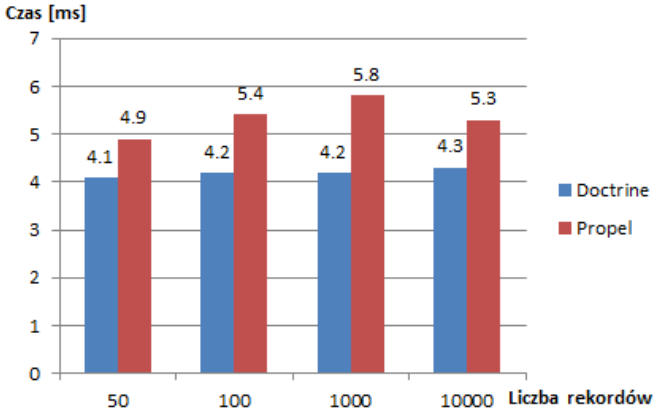
4. Rezultaty badań

- **Scenariusz 1**

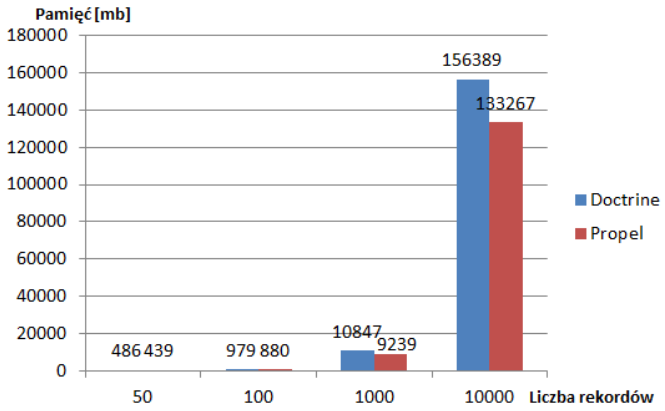
Pierwszy scenariusz polegał na zapisie 50, 100 oraz 1000 rekordów niepowiązanych do relacyjnej bazy danych.



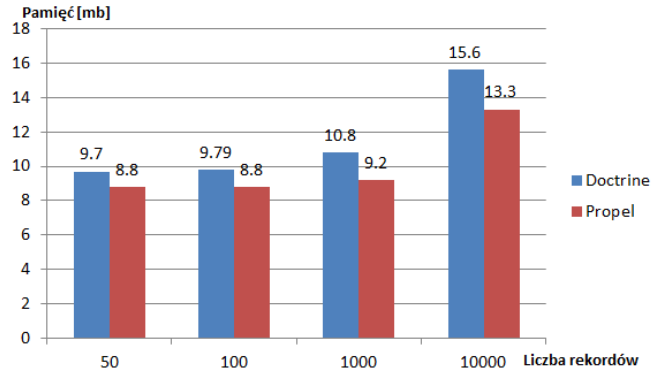
Rys. 2. Czas wykonywania skryptu w milisekundach



Rys. 3. Średni czas wykonywania skryptu w milisekundach



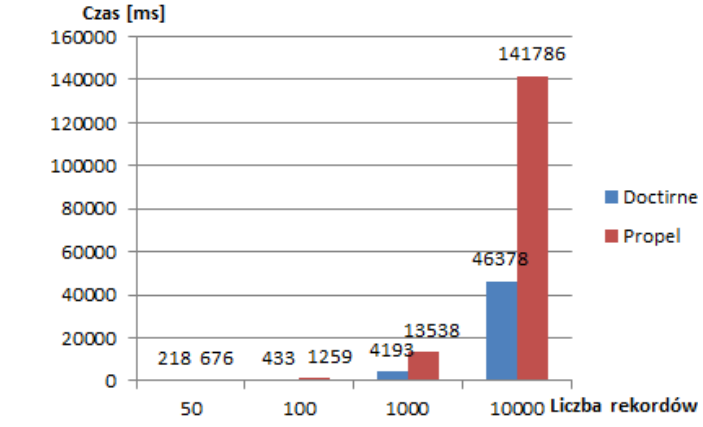
Rys. 4. Całkowite zużycie pamięci w megabajtach



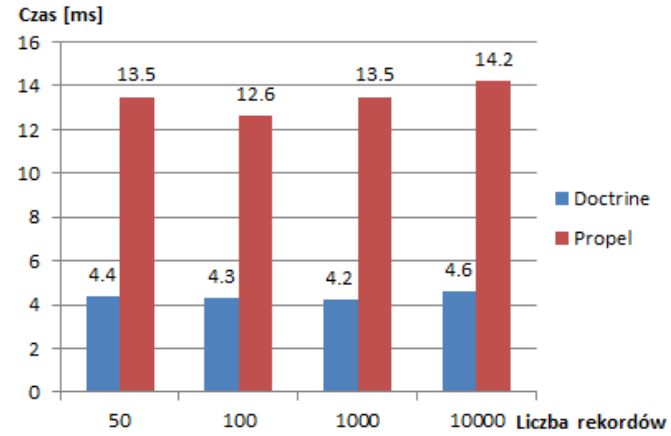
Rys. 5. Średnie zużycie pamięci w megabajtach

- **Scenariusz 2**

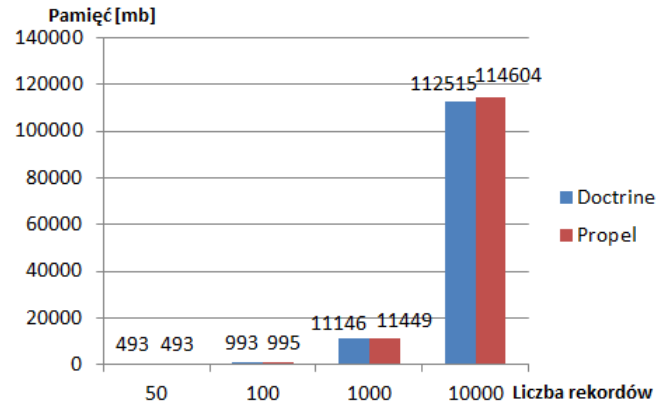
Drugi scenariusz polegał na zapisie 50, 100 oraz 1000 rekordów powiązanych do relacyjnej bazy danych.



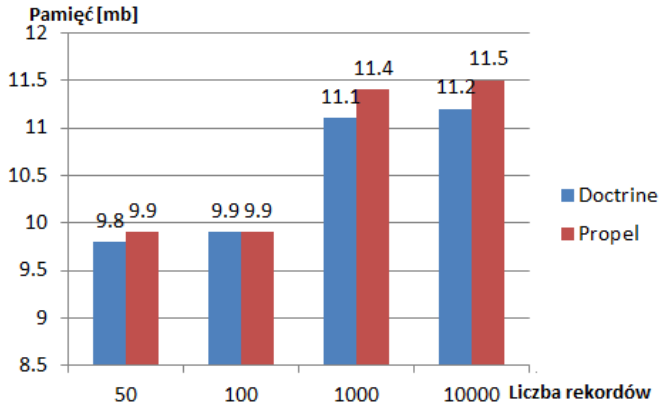
Rys. 6. Czas wykonywania skryptu w milisekundach



Rys. 7. Średni czas wykonywania skryptu w milisekundach



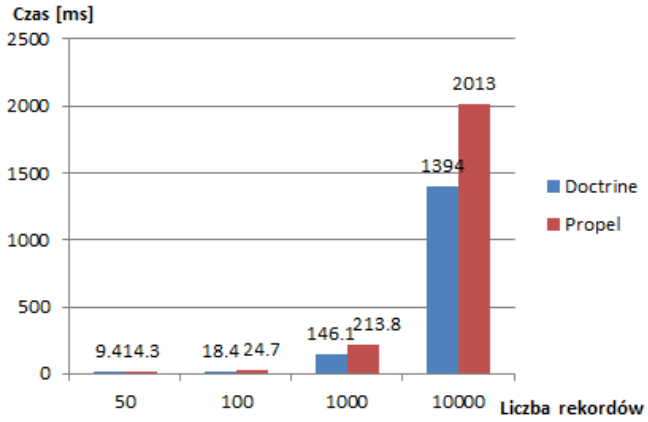
Rys. 8. Całkowite zużycie pamięci w megabajtach



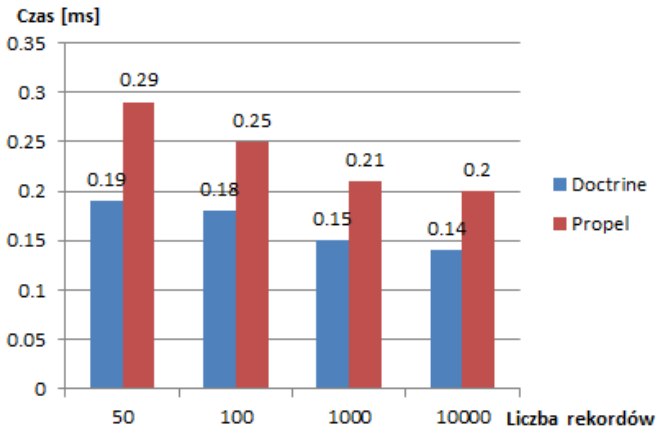
Rys. 9. Średnie zużycie pamięci w megabajtach

- **Scenariusz 3**

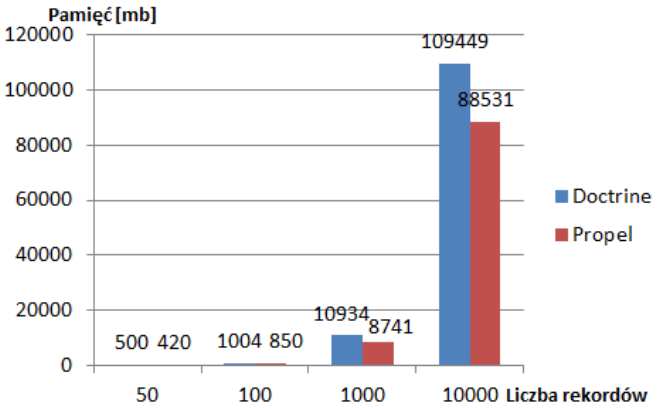
Trzeci scenariusz polegał na odczycie 50, 100 oraz 1000 rekordów niepowiązanych do relacyjnej bazy danych.



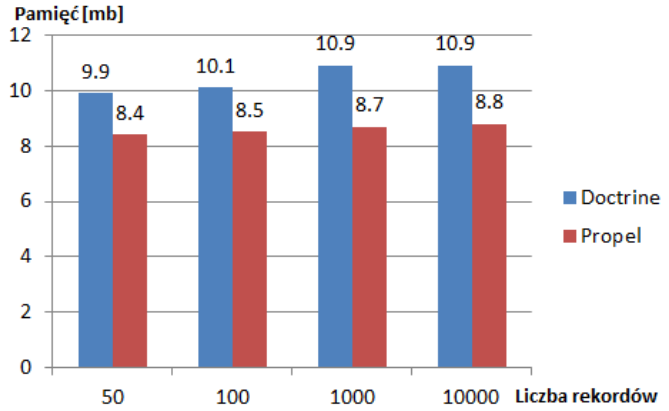
Rys. 10. Czas wykonywania skryptu w milisekundach



Rys. 11. Średni czas wykonywania skryptu w milisekundach



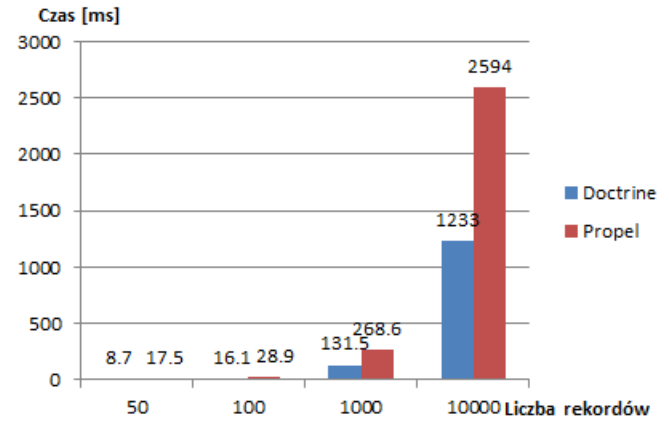
Rys. 12. Całkowite zużycie pamięci w megabajtach



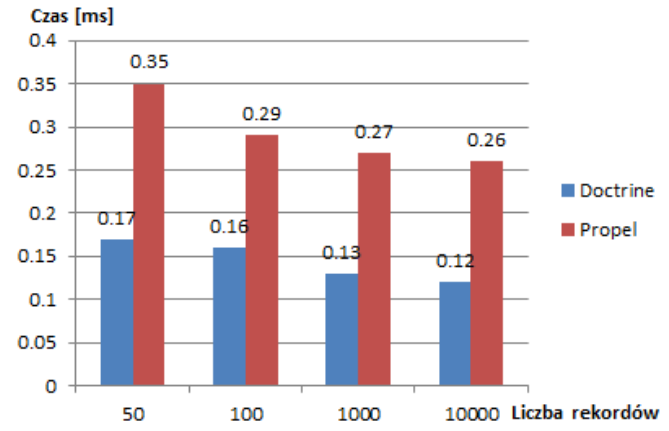
Rys. 13. Średnie zużycie pamięci w megabajtach

- **Scenariusz 4**

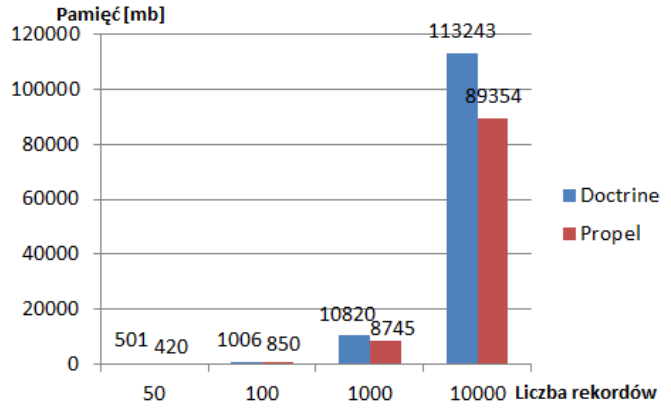
Czwarty scenariusz polegał na odczycie 50, 100 oraz 1000 rekordów powiązanych do relacyjnej bazy danych.



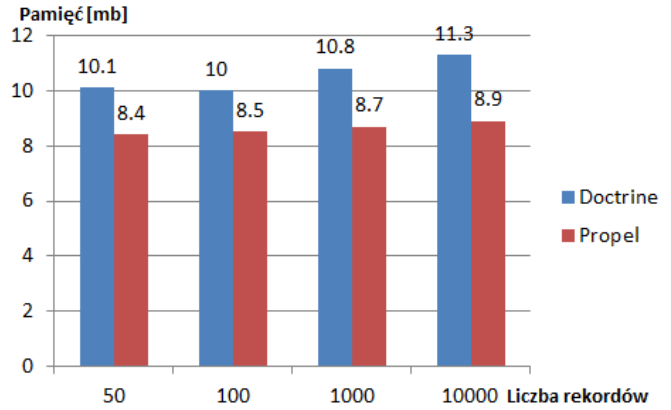
Rys. 14. Czas wykonywania skryptu w milisekundach



Rys. 15. Średni czas wykonywania skryptu w milisekundach



Rys. 16. Całkowite zużycie pamięci w megabajtach



Rys. 17.Średnie zużycie pamięci w megabajtach

5. Wnioski

Pierwszy scenariusz badał zapisanie pojedynczego rekordu bez asocjacji. W każdej sytuacji Doctrine przodował względem Propel w czasie wykonywania skryptu. W przypadku scenariusza pierwszego, Doctrine był szybszy o 17-28%, zależnie od ilości utrwalonych rekordów (Rysunek 2, Rysunek 3). Pomiar zużycia pamięci wykazał, że Propel zużywa od 10% do 17% mniej pamięci względem Doctrine (Rysunek 4, Rysunek 5).

Utrwalenie rekordów powiązanych wykazało, że Doctrine potrafi być trzy razy szybszy niż Propel. Wynik ten, można zaobserwować dla próby 50, 100 i 1000 rekordów (Rysunek 6, Rysunek 7). W przypadku zużycia pamięci, Propel potrafił zaoszczędzić do 6% pamięci (Rysunek 8, Rysunek 9).

Scenariusz trzeci badał odczyt rekordów bez relacji. Doctrine po raz kolejny był lepszy pod względem czasu wykonywania skryptu. Wspomniany zwycięzca był szybszy o 35-39%, zależnie od liczby utrwalonych rekordów (Rysunek 10, Rysunek 11). Propel tym razem także potrafił wykorzystać znaczne mniejszą ilość pamięci, bo aż do 25%.

W przypadku odczytu rekordów powiązanych, czas wykonywania skryptu potrafił być dwa razy większy w przypadku Propela w stosunku do Doctrine. Propel, jak i w poprzednich scenariuszach potrzebował zaalokować mniejszą ilość pamięci. Różnica w wykorzystania pamięci oscylowała w granicach 20-25%.

6. Podsumowanie

W ramach pracy przebadano i porównano obie technologie ORM pod względem wydajności, co pozwoliło sformułować wniosek, która technologia mapowania obiektowo-relacyjnego ma mniejszy czas wykonywania skryptu.

Z otrzymanych wyników jednoznacznie można stwierdzić, że Doctrine wykonuje operacje na bazie danych szybciej niż Propel, więc teza badawcza się potwierdziła. Badania wykazują, że obie technologie mają swoje mocne oraz słabe strony. W przypadku Doctrine, mocną stroną jest czas wykonywania skryptu, słabą zaś zużycie pamięci. Przy obserwacji wyników Propela, można zauważyć, że jest na odwrót – mniejsze zużycie pamięci i długi czas wykonywania skryptu.

Testy zapisu oraz odczytu rekordów dały jednoznaczne wyniki - Doctrine potrafi wykonać ten sam skrypt nawet trzy razy szybciej niż Propel. Zależnie od scenariusza, Doctrine był szybszy od 17% do nawet 300%. Badanie zużycia pamięci serwera wykazało, że Propel potrafi użyć mniej zasobów w przedziale 6%-25%.

W testach zostały przebadane dwa ważne czynniki, które jak się okazało potrafią być mocną oraz słabą stroną obu ORM. Szybszy czas wykonywania skryptu jest zawsze korzystniejszy dla użytkownika końcowego. Zużycie pamięci serwera jest kwestią podrzędną, gdyż pamięć RAM zawsze można zwiekszyć małym kosztem, a czas wykonywania skryptu nie da się żadnym prostym sposobem zmniejszyć, tym bardziej, kiedy różnica potrafi być trzy raz większa na korzyść Doctrine.

Podsumowując Doctrine jest wydajniejszą technologią mapowania obiektowo-relacyjnego w Symfony 3. W pracy zostały przeanalizowane jedynie dwa parametry. Ciekawym kierunkiem dalszych badań może być zbadanie innych parametrów takich jak zarządzanie transakcjami biznesowymi, konflikty zapytań, pojemność oraz konfiguracja [10][13].

Literatura

[1] Matt Zandstra: PHP Objects, Patterns and Practice, 5th Edition, 2016.

[2] Chris Pitt: Pro PHP MVC, 2012

[3] O'Reilly Media: Learning PHP Design Patterns, 2013

[4] Kevin Dunglas: Persistance in PHP with Doctrine ORM, 2013

[5] http://docs.doctrine-project.org/projects/doctrine-orm/en/latest/reference/association-mapping.html [18.08.2017]

[6] Jason E. Sweat: PHP Architect's Guide to PHP Design Patterns, 2005

[7] Martin Fowler: Patterns of Enterprise Application Architecture, 2002

[8] http://docs.doctrine-project.org/projects/doctrine-orm/en/latest/reference/working-with-objects.html [3.12.2017]

[9] http://www.vertabelo.com/blog/technical-articles/side-by-side-doctrine2-and-propel-2-comparison [20.09.2017]

[10] https://blog.appdynamics.com/engineering/top-6-database-performance-metrics-to-monitor-in-enterprise-applications/ [4.12.2017]

[11] http://docs.doctrine-project.org/projects/doctrine-orm/en/latest/reference/architecture.html [4.12.2017]

[12] https://en.wikipedia.org/wiki/Propel_(PHP) [4.12.2017]

[13] O'Reilly Media: High Performance MySQL, 3rd Edition, 2012