JOURNAL
COMPUTER SCIENCES INSTITUTE

JCSI 17 (2020) 326–331
Received: 05 July 2020
Accepted: 24 September 2020

Impact of the persistence layer implementation methods on application performance

Wpływ metod implementacji warstwy persystencji na wydajność aplikacji

Kamil Siebyła*, Maria Skublewska-Paszkowska

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

Abstract

There are various methods for creating web applications which have different levels of performance. The way the data access will be programmed at a specific endpoint, therefore, determines the performance of the entire application. There are many programming methods that are often time-consuming to implement. This paper presents a comparison of the available methods of handling the persistence layer in relation to the efficiency of their implementation. There are few methods for Entity Framework enviroment : Linq To Entity, Explicite Loading, Eager Loading, Raw SQL oraz Stored Procedure. While executing particular test scenarios, it was found that working on pure sql code in the case of working with the persistence layer is more efficient than using Object-Relational Mapper.

Keywords: time performance of the queries; entity framework core; sql

Streszczenie

Istnieją różne metody tworzenia aplikacji internetowych. Każda z tych metod charakteryzuje się różnym poziomem wydajności. Sposób, w jaki zostanie zaprogramowany dostęp do danych na konkretnym punkcie końcowym, uwarunkowuje więc wydajność całej aplikacji. Niniejszy artykuł przedstawia porównanie dostępnych sposobów obsługi warstwy persystencji w stosunku do wydajności ich implementacji. Sposobami tymi w środowisku Entity Frameworka są: Linq To Entity, Explicite Loading, Eager Loading, Raw SQL oraz Stored Procedure.. Wykonując poszczególne scenariusze testowe ustalono, że działanie na czystym kodzie sql w przypadku pracy z warstwą persystencji jest wydajniejsze niż korzystanie z maperów obiektowo relacyjnych (ang. *Object-Relational Mapper*).

Słowa kluczowe: wydajność czasowa zapytań; entity framework core; sql

*Corresponding author

Email address: **kamil.siebyla@gmail.com** (K. Siebyła)

1. Wstęp

Przez kilka ostatnich lat oblicze tworzenia aplikacji internetowych przeszło prawdziwą rewolucję. Klienci końcowi rzeczonych aplikacji oczekują wysokiej jakości wizualnej, jak i wydajności. Warstwa persystencji w przypadku tej drugiej właściwości odgrywa kluczową rolę. Sytuacja, w której niepoprawnie zostaną zaimplementowane mechanizmy związane z obsługą bazy danych, może mieć negatywne skutki dla działania aplikacji. Negatywnie może to również wpłynąć na wydajność warstwy wizualnej, bowiem w momencie, kiedy brakuje danych dla kontrolek mogą one zachowywać się w sposób niepożądany. Implementacja określonych rozwiązań i ich rozbudowa od strony programistycznej jest zróżnicowana pod kątem trudności implementacji i zarządzania. Programista często jest odgórnie ograniczony czasowo, przez co należy wybrać metody implementacji, które są wydajne dla danego typu aplikacji oraz które można w łatwy sposób rozbudować przy jednocześnie możliwie najkrótszym czasie tworzenia kodu. Badania dotyczące interfejsów ORM (ang. Object Relational Mapping) zostały poruszone w kilku pracach. Wykorzystywane są przez autorów tych prac różne technologie i podejścia, jednak celem badań poruszanym w tych publikacjach jest wydajność aplikacji. Autor artykułu pod tytułem „Comparsion of performance beetwen Raw SQL and Eloquent ORM in Laravel" [1] wykorzystuje do badań wydajności aplikacji internetowych framework języka PHP (ang. Hypertext Preprocessor), jakim jest Laravel. Bada trzy różne techniki obsługi bazy danych, Eloquent ORM, Query Builder oraz Raw SQL. Autor publikacji stawia tezę w swojej pracy, mówiącą, że Eloquent ORM intuicyjnie powinien być wolniejszy niż kod napisany w czystym SQL – co potwierdza wynikami przeprowadzonych badań. Kolejny artykuł nosi tytuł „Performance evaluation of java object-relational mapping tools" [2]. Autorzy publikacji skupiają się na porównaniu różnych narzędzi ORM w ekosystemie Javy. Porusza on bardzo obszernie wszystkie możliwe narzędzia badając ich wydajność wykorzystując odpowiednie zapytania. Autorzy sugerują, że gdy używane są standardy JPA (ang. Java Persistance API) wydajność konkretnych narzędzi wzrasta. Potwierdza to fakt, iż Hibernate okazał się być najwydajniejszym narzędziem w środowisku Javy. Nie przypadkowo oparty jest on właśnie na JPA. Zdecydowanie najsłabiej wypadł Open JPA w testach zaprezentowanych przez autora publikacji.

Celem niniejszego artykułu jest porównanie metod implementacji warstwy persystencji oraz zbadanie ich wydajności przy użyciu autorskiego narzędzia testującego punkty końcowe API (Application Programing Interface) oraz autorskiej aplikacji internetowej, pełnią-

cej rolę warstwy wizualnej dla bazy danych. Ponadto analizie zostanie poddana trudność implementacji poszczególnych metod dostępu do danych w stosunku do rozmiaru badanej aplikacji.

W artykule postawiono hipotezę badawczą o brzmieniu: *Operacje na czystym kodzie SQL są wydajniejsze od operacji realizowanych przez Entity Frameowrk Core*.

2. Metody dostępu do danych w .NET

2.1. Entity Framework Core

Jest to kompaktowa, rozszerzalna, międzyplatformowa wersja klasycznego Entity Framework, który jest narzędziem pozwalającym na dostęp do warstwy danych aplikacji. Dzięki Entity Framework Core [3] istnieje możliwość pracy z bazą danych przy użyciu klasycznych obiektów. Oznacza to, że chcąc odwoływać się do bazy danych, choćby w klasycznym CRUD (ang. *Create Read Update Delete*), programista nie jest zmuszony dopisania złożonego kodu sql. Dzięki Entity Framework jest w stanie wykonywać operacje na bazie danych w taki sposób jakby pisał kod obiektowy. Deweloper korzysta z abstrakcji jaką oferuje Entity Framework Core. Bazowym pojęciem w tym przypadku jest model, który symuluje encje zawarte w bazie danych. Wraz z kontekstem zawartym w Entity Framework Core stanowi integralną część sesji połączenia z bazą danych.

2.2. Procedury Składowane

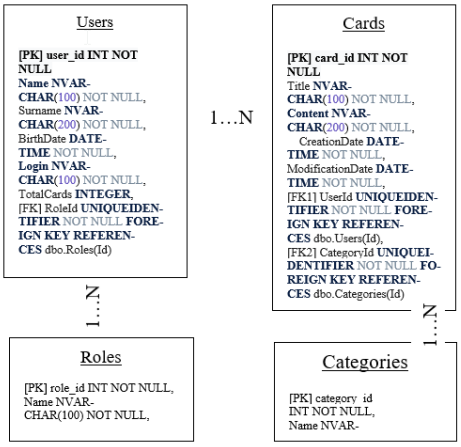
Procedury składowane są znane programistom od kilkudziesięciu lat. Pisanie procedur rozszerza nieco wachlarz możliwości warstwy danych, pod kątem zaawansowania zapytań. Dzięki napisaniu zaawansowanej procedury, programista jest w stanie wyciągnąć skomplikowany zestaw danych. Procedury posiadają parametry, które w praktyce są przekazywane przez kod serwerowy. Wydajność procedur w praktyce powinna prezentować wysoki poziom, gdyż przesyłane są parametry do procedury w jednym żądaniu. Dodatkowo nie jest przesyłany cały kod SQL, a jedynie nazwa procedury, a silnik bazodanowy mając nazwę wykona żądane operacje. Niniejszy artykuł będzie prezentował procedury składowane w środowisku napisane w języku Transact SQL w środowisku MSSQL firmy Microsoft [4].

3. Architektura środowiska pracy

Badania zostały przeprowadzone w środowisku .NET Framework z wykorzystaniem frameworka Entity Framework Core. Interfejs ten został poddany analizie w kontekście realizacji podstawowych operacji na bazie danych w zależności od liczby danych. Dodatkowo został wykorzystany skrypt napisany w Powershell [5], który testuje punkty końcowe serwisu REST (ang *Representational State Tranfer*) [6] mierząc czas wykonania żądania przy wygenerowanych wcześniej danych. Testy zostały przeprowadzone na sprzęcie o podanych poniżej parametrach: MSI GE70, 64 bitowy system operacyjny Windows 10 PRO, Procesor Intel(R) Core(TM) i5-4200M CPU @ 2.50 GHz, karta graficzna NVIDIA GeForce GTX 765M, 8 GB pamięci RAM DDR3 o częstotliwości 1600 MHz oraz dysku SSD o pojemności 128 GB. Baza danych została uruchomiona lokalnie przy użyciu konteneryzacji. W tym celu został wykorzystany popularny kontener o nazwie docker. Uruchomienie bazy danych w kontenerze sprowadzało się instalacji klienta dockera na lokalnej maszynie, pobraniu obrazu rzeczonej bazy danych a następnie wykonaniu kilku komend umożliwiających utworzenie kontenera z działającym pobranym wcześniej obrazem bazy danych.

4. Aplikacja testowa oraz narzędzia

Środowisko testowe składa się ze skonfigurowanej bazy danych, aplikacji internetowej, serwisu typu REST oraz skryptu Powershell. Aplikacja jest środowiskiem, które służy do testowania wydajności warstwy persystencji. Jest interfejsem, który jest wywoływany asynchronicznie przez skrypt Powershell w celu przetestowania wydajności konkretnych metod dostępu do danych. Na rysunku 1 schemat bazy danych aplikacji testowej.



Rysunek 1: Schemat bazy danych aplikacji testowej.

5. Scenariusze testowe

Scenariusze testowe zostały skonstruowane tak, aby jak najwierniej przedstawić wady i zalety konkretnych metod implementacyjnych. Tabele 1-4 przedstawiają scenariusze badawcze. W ostatniej kolumnie podana jest liczba rekordów z bazy danych. Znak „/" w poniższych tabelach oddziela liczbę obsługiwanych rekordów dla danej metody w tysiącach dla kolejnych prób.

Tabela 1: Scenariusze badawcze dla odczytu bez śledzenia zmian

Numer scenariusza badawczego	Rodzaj implementacji	Liczba obsługiwanych rekordów w tysiącach
1	L2E	10/60/160
2	Explicite Loading	10/60/160
3	Eager Loading	10/60/160
4	Raw SQL	10/60/160
5	Procedure	10/60/160

Tabela 2: Scenariusze badawcze dla odczytu z uruchomionym śledzeniem zmian

Numer scenariusza badawczego	Rodzaj implementacji	Liczba obsługiwanych rekordów w tysiącach
6	L2E	10/60/160
7	Explicite Loading	10/60/160
8	Eager Loading	10/60/160
9	Raw SQL	10/60/160
10	Procedure	10/60/160

Tabela 3: Scenariusze badawcze dla zapisu bez śledzenia zmian

Numer scenariusza badawczego	Rodzaj implementacji	Liczba obsługiwanych rekordów w tysiącach
11	Kontekst Globalny	3/10/12
12	Kontekst Lokalny	3/10/12
13	Automatyczna detekcja zmian	3/10/12

Tabela 4: Scenariusze badawcze dla zapisu z uruchomionym śledzeniem zmian

Numer scenariusza badawczego	Rodzaj implementacji	Liczba obsługiwanych rekordów w tysiącach
14	Kontekst Globalny	3/10/12
15	Kontekst Lokalny	3/10/12
16	Automatyczna detekcja zmian	3/10/12

Każdy z powyższych scenariuszy został wykonany jednokrotnie przy użyciu przygotowanej infrastruktury technicznej.

6. Wykonywane operacje podczas realizacji scenariuszy testowych

Podczas realizacji scenariuszy badawczych wyróżnione zostały dwa typy operacji zapisu oraz odczytu.

6.1. Operacje odczytu

Entity Framework Core udostępnia dwie główne grupy metod, które realizują operacje odczytu. Pierwszą grupą rozwiązań są te, które, cechują się znacznym poziomem obiektowości oraz abstrakcji. Dla EF (ang. *Entity Framework*) nazywane są „Linq To Entities". Drugą grupą metod są rozwiązania implementujące język SQL. W środowisku testowym wykorzystywanym w niniejszej pracy jest to odpowiednio „EntitySQL" [7]. Każda z tych metod umożliwia konstruowanie złożonych zapytań do bazy danych. Różni je sposób w jaki dane z bazy są otrzymywane. Obydwie metody zwracają zmaterializowane dane. Oznacza to, że programista w kodzie otrzymuje instancję encji. Z punktu widzenia programisty jest to bardzo wygodne rozwiązanie. Jednakże L2E (ang. Linq To Entity) wprowadza dodatkową warstwę abstrakcji do zbioru wynikowego poprzez typy anonimowe, które w znaczący sposób upraszczają implementację mechanizmów związanych z encjami. Niesie to jednak za sobą również negatywne skutki w postaci niemożności tworzenia silnie dynamicznych oraz złożonych zapytań. EntitySQL z kolei zwraca dane w postaci kolekcji. Takie kolekcje jest trudniej przeszukiwać, a programista nie jest w stanie zbudować warstwy abstrakcji, która ułatwiłaby dostęp do zwróconych przez zapytanie danych.

6.2. Operacje zapisu

Operacje zapisu są realizowane przy udziale kontekstu jakim programista dysponuje w środowisku aplikacji. Każda instancja kontekstu posiada mechanizm śledzący zmiany (ang. *Change Tracker*). Przechowywane są w nim informacje o zmianach jakie zostały dokonane na instancji konkretnej encji. Kontekst śledzi encję i w momencie wywołania metody „SaveChangesAsync" zapisuje informacje do bazy danych. Domyślnie zapytania, które zwracają encje konkretnego typu, są śledzone. Z drugiej strony programista ma do dyspozycji zapytania, pozbawione śledzenia zmian. Kontekst umożliwia ręczne wyłączenie śledzenia zmian. Takie zapytania są szczególnie przydatne, kiedy dane zwracane przez zapytanie znajdują zastosowanie w elementach, w których są wykorzystywane jako dane tylko do odczytu. Istnieją dwa sposoby na wyłączenie tej funkcjonalności. Pierwszym z nich jest wyłączenie kontekstu globalnie z poziomu instalacji kontekstu. Drugim sposobem jest użycie metody „AsNoTracking" przy wykonaniu samego zapytania. Wartą nadmienia kwestią w przypadku mechanizmu śledzenia zmian w Entity Framework Core jest rozwiązywanie identyfikatorów encji (ang. *Identity Resolution*) [8]. Chodzi o to, że dopóki zapytanie jest śledzone, Entity Framework Core potrafi w momencie materializacji encji rozpoznać, czy dana encja jest śledzona i jeżeli tak w istocie jest, zwrócić instancję dokładnie tej śledzonej encji. Jeżeli w zbiorze wynikowym jest wiele wystąpień danej encji, to zostanie zwrócona ta sama instancja, tylko wielokrotnie. W przypadku braku śledzenia zmian za każdym razem będzie tworzona nowa instancja encji dla każdego wystąpienia jej w zbiorze wynikowym. Mechanizm ten jest kluczowy z punktu widzenia wydajności aplikacji. Zostanie zbadane jakie są różnice w wydajności dla konkretnych zbiorów danych z włączonym lub wyłączonym śledzeniem zmian. Dodatkowym pojęciem, jakie wiąże się z zapisem, jest współbieżność dostępu do danych. Sytuacja taka ma miejsce kiedy dwóch użytkowników próbuje zmodyfikować tą samą encję. Jeden użytkownik jest w trakcie modyfikacji, natomiast drugi zdążył zapisać już dane do bazy danych. Popularnym i zarazem najlepszym rozwiązaniem jest wprowadzenie pojęcia właściwości o nazwie „rowversion" [10]. Jest to po prostu dodatkowa kolumna w bazie danych, która przechowuje aktualną wersję wiersza. W środowisku SQL Server stworzono specjalny typ zmiennej o nazwie „rowversion". Jest to zmienna liczbowa, która przy

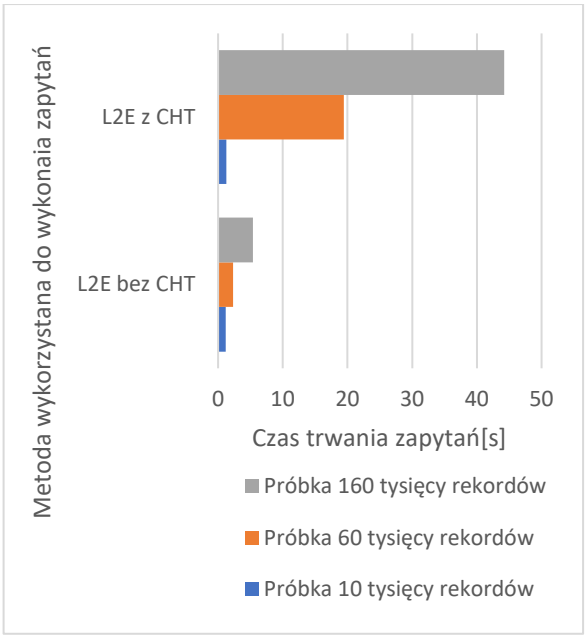
każdej modyfikacji wiersza inkrementuje swoją wartość. Dzięki temu mechanizmowi, kiedy jeden użytkownik zacznie modyfikować wiersz, podczas gdy inny zapisał nową wartość wierszu w czasie kiedy pierwszy wprowadzał modyfikację, użytkownik który modyfikował zawartość wiersza w momencie wywołania metody „SaveChanges" zostanie wyświetlony wyjątek „DbUpdateConcurencyException".

7. Porównanie graficzne wyników

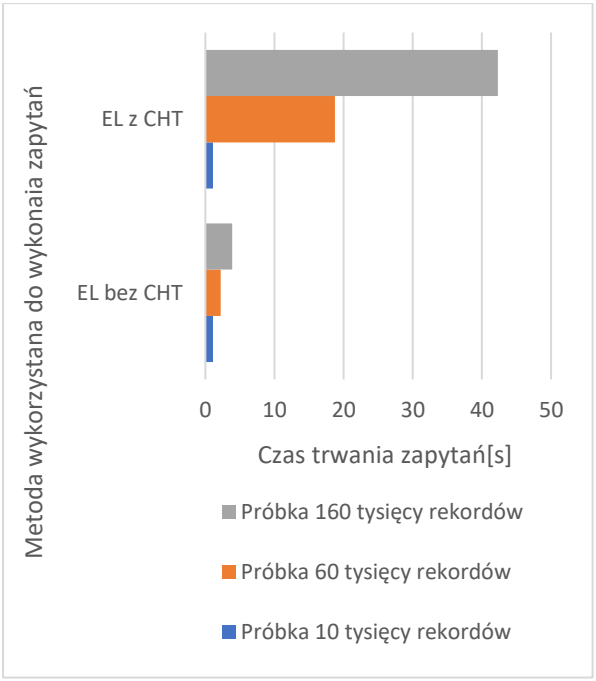
Każdy z powyższych scenariuszy został wykonany jednokrotnie przy użyciu przygotowanej infrastruktury technicznej. Wyniki jakie zostały otrzymane po wykonaniu konkretnych scenariuszy zostały przedstawione w tabelach: 1, 2, 3 oraz 4.

7.1. Wydajność odczytu

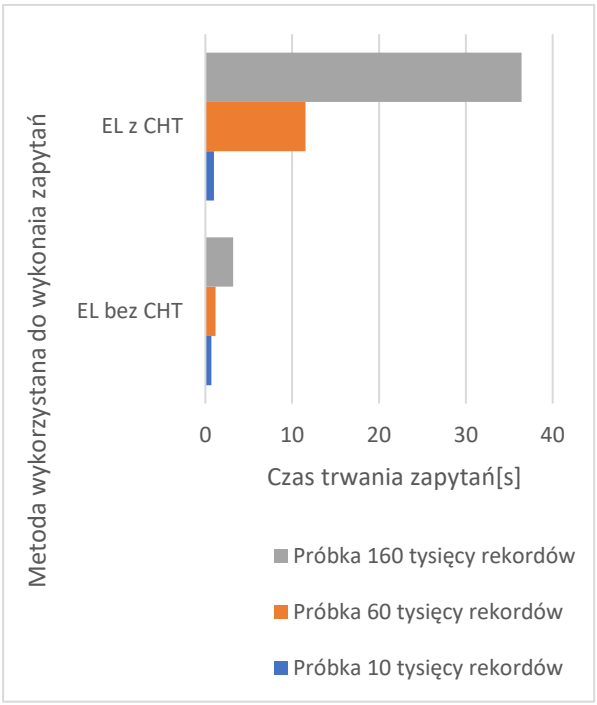
Na rysunkach 2-9 przedstawiono wyniki wykonania wszystkich scenariuszy testowych zawartych w tabelach 1-4. Rysunki te przedstawiają za pomocą wykresów słupkowych wartości czasowe jakie zostały zarejestrowane dla każdej z badanych metod.



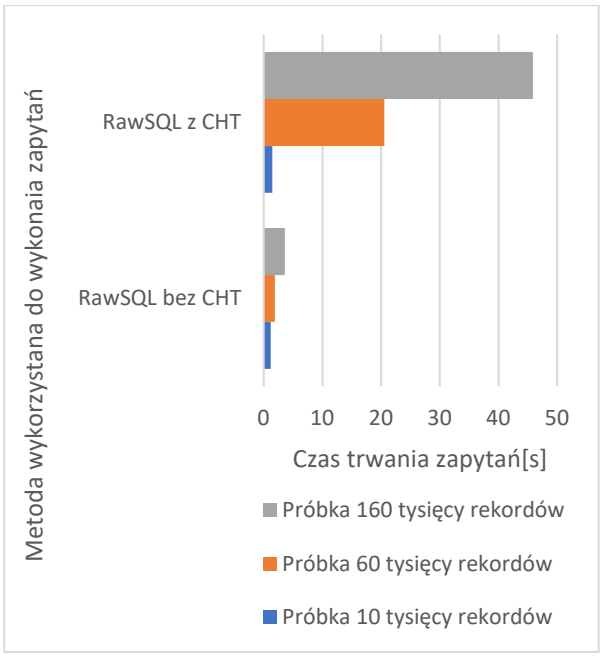
Rysunek 2: Porównanie wydajności odczytu dla L2E z włączonym (L2E z CHT) oraz wyłączonym śledzeniem zmian (L2E bez CHT)



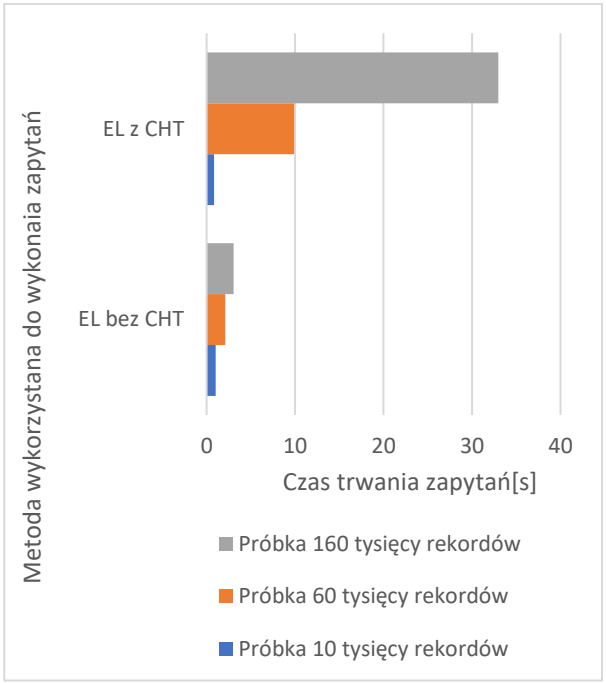
Rysunek 3: Porównanie wydajności odczytu dla Explicite Loading z włączonym oraz wyłączonym śledzeniem zmian



Rysunek 4: Porównanie wydajności odczytu dla Eager Loading z włączonym oraz wyłączonym śledzeniem zmian



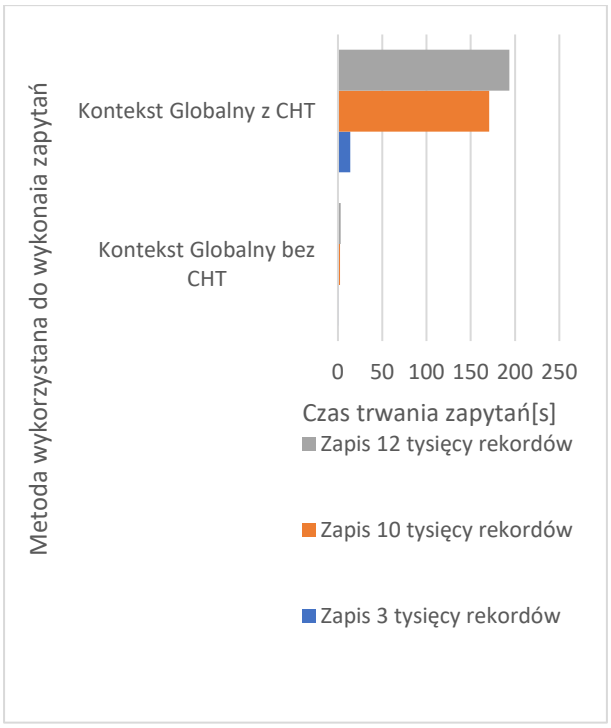
Rysunek 5: Porównanie wydajności odczytu dla Raw SQL z włączonym oraz wyłączonym śledzeniem zmian



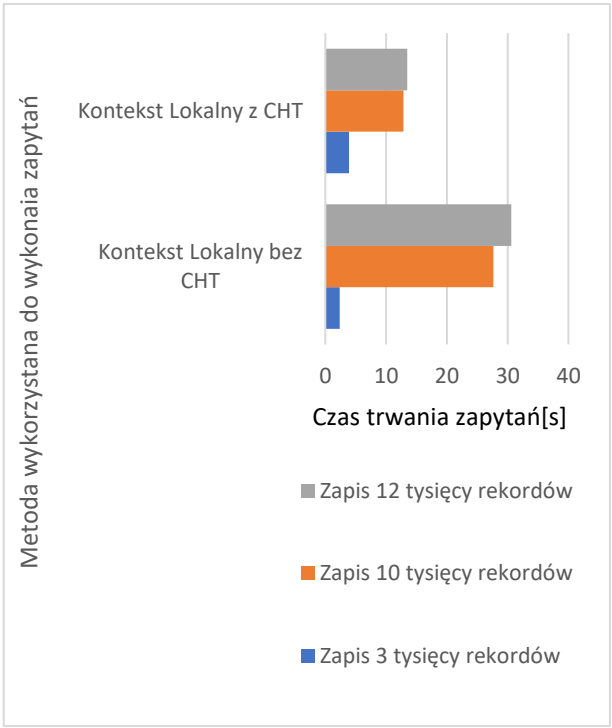
Rysunek 6: Porównanie wydajności odczytu dla procedury składowanej z włączonym oraz wyłączonym śledzeniem zmian

7.2. Wydajność zapisu

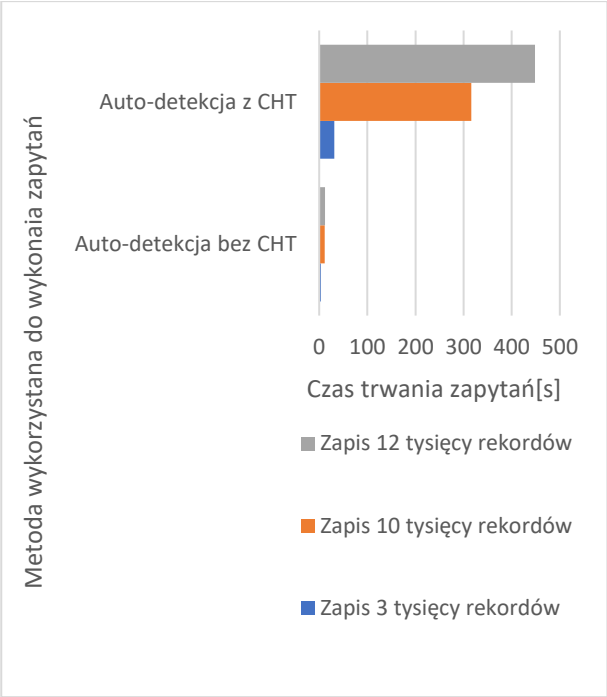
Rysunki 7-9 przedstawiają wydajność związaną z zapisem.



Rysunek 7: Porównanie wydajności zapisu dla kontekstu globalnego z włączonym oraz wyłączonym śledzeniem zmian



Rysunek 8: Porównanie wydajności zapisu dla kontekstu lokalnego z włączonym oraz wyłączonym śledzeniem zmian



Rysunek 9: Porównanie wydajności zapisu dla auto-detekcji zmian z włączonym oraz wyłączonym śledzeniem zmian

8. Wnioski

Przeprowadzone badania związane z monitorowaniem wydajności konkretnych metod implementacji warstwy persystencji pokazały, że czysty kod SQL efektywniej razi sobie z przetwarzaniem zapytań. Wyniki uzyskane w pierwszym eksperymencie (tabele: 1 i 2 oraz rysunki: 2-6) pozwalają zauważyć różnice pomiędzy odpowiednimi metodami implementacji Entity Framework Core, a kodem SQL. Spowodowane jest to tym, że Entity Framework Core jest obudowany warstwą abstrakcji w postaci obiektów klas implementujących dostęp do operacji na bazie danych. Dodatkowo możliwość manipulacji kontekstem oraz mechanizm śledzenia zmian powodują, że czasy operacji zapisu jak i odczytu wzrastają bądź zostają skrócone w zależności od zastosowanej techniki. Można zauważyć, że autodetekcja zmian znacząco wydłuża czas wykonania zapytań – zarówno dla zapisu jak i dla odczytu. Procedura składowana dla zestawu 160 tysięcy rekordów z włączoną detekcją zmian wykonała się w przybliżeniu w 33 sekundy. Jest to najlepszy wynik. Jednak ta sama procedura dla takiej samej porcji danych z wyłączoną detekcją zmian wykonała się w niecałe 10 sekund. Jest to kilka rzędów wielkości. Dla porównania metoda RawSQL użyta z Entity Framework wykonywała się średnio 10 sekund dłużej dla każdego z przypadków – co świadczy o lepszej wydajności procedury składowanej. Można więc powiedzieć, że im bliżej źródła danych programista operuje, tym efektywniej działają operacje związane z warstwą persystencji. Przeprowadzone badania (rys. 2-9) pozwalają potwierdzić, że postawiona teza *Operacje na czystym kodzie SQL są wydajniejsze od operacji realizowanych przez Entity Frameowrk Core została udowodniona.* Operacje na czystym kodzie sql są wydajniejsze niż narzędzia ORM jakimi dysponuje programista w codziennej pracy.

Literatura

[1] I. Jound, H. Halimi, M. Svahnberg, Comparsion of performance beetwen Raw SQL and Eloquent ORM in Laravel, Faculty of Computing Blekinge Institute of Technology SE-371 79 Karlskrona Sweden.

[2] H. Yousaf, Performance evaluation of java object-relational mapping tools, Georgia: University of Georgia, 2012.

[3] P. Anbazhagan, Mastering Entity Framework Core 2.0, Wydawnictwo Helion SA, Gliwice 2017.

[4] Oficjalna dokumentacja MSSQL, https://docs.microsoft.com/en-us/sql, [06.2020].

[5] Oficjalna dokumentacja PowerShell, https://docs.microsoft.com/en-us/powershell, [06.2020].

[6] G. Arorra, T. Dash, Building RESTful Web Services with .NET Core, Wydawnictwo Helion SA, Gliwice 2019.

[7] Dokumentacja EntitySQL, https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/ef/language-reference/entity-sql-overview, [06.2020].

[8] O. Mehboob, A. Khan, C# 7 i.NET Core 2.0 Programowanie wielowątkowych i współbieżnych aplikacji, Wydawnictwo Helion SA, Gliwice 2019.

[9] Dokumentacja Entity Framework, https://docs.microsoft.com/en-us/dotnet/framework/ data/adonet/ef, [06.2020].