JOURNAL
COMPUTER SCIENCES INSTITUTE

JCSI 16 (2020) 285–292
Received: 23 June 2020
Accepted: 7 July 2020

Comparative analysis of selected object-relational mapping systems for the .NET platform

Analiza porównawcza wybranych systemów mapowania obiektowo-relacyjnego dla platformy .NET

Marcin Bobel*, Krzysztof Drzazga*, Maria Skublewska-Paszkowska

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

Abstract
This article is devoted to the comparison of two object-relational mapping systems supported by .NET platform - Entity Framework Core and NHibernate. The research hypothesis "framework NHibernate is more efficient than Entity Framework Core in the context of DML operations" was put forward. In order to make an efficiency analysis of ORM frameworks, a desktop application was designed and implemented to enable testing and visualization of results. The NHibernate framework turned out to be much more efficient than Entity Framework Core in single tests and slightly faster in bulk tests. The stability of both frameworks was similar.

Keywords: .NET; ORM; Entity Framework Core; NHibernate

Streszczenie
Niniejszy artykuł został poświęcony porównaniu dwóch systemów mapowania obiektowo-relacyjnego wspieranych przez platformę .NET – Entity Framework Core oraz NHibernate. Postawiono hipotezę badawczą, „szkielet NHibernate jest wydajniejszy niż Entity Framework Core w kontekście operacji DML". W celu przeprowadzenia analizy wydajności szkieletów ORM, zaprojektowano i zaimplementowano aplikację desktopową umożliwiającą wykonanie testów oraz wizualizację wyników. Szkielet NHibernate okazał się zdecydowanie wydajniejszy niż Entity Framework Core w testach pojedynczych i nieznacznie szybszy w testach masowych. Stabilność obu szkieletów kształtowała się na podobnym poziomie.

Słowa kluczowe: .NET; ORM; Entity Framework Core; NHibernate

*Corresponding author
Email addresses: marcin.bobel@pollub.edu.pl (M. Bobel), krzysztof.drzazga@pollub.edu.pl (K. Drzazga)

©Published under Creative Common License (CC BY-SA v4.0)

1. Wstęp

Na przestrzeni ostatnich lat, wraz ze wzrostem zapotrzebowania na usługi informatyczne, nastąpił rozwój narzędzi umożliwiających ich realizację. Można zaobserwować powstawanie nowych języków programowania, a także dynamiczny rozwój istniejących języków oraz szkieletów programistycznych. Wszelkie zmiany mają na celu poprawę takich aspektów rozwiązań informatycznych jak bezpieczeństwo i wydajność, jednocześnie ułatwiając programistom ich implementację.

Nowoczesne aplikacje zwykle nie powstają w oparciu o jedną technologię, lecz są zbiorem kilku współgrających ze sobą rozwiązań. Programista ma do dyspozycji coraz bardziej zaawansowane szkielety programistyczne, które pozwalają na szybszą implementację rozwiązania. Ponadto, wspomniane szkielety są odpowiedzialne za niektóre czynności, których implementacja niegdyś była zadaniem programisty. W przypadku takich rozwiązań, ważna jest równowaga pomiędzy czynnościami wykonywanymi niejawnie przez szkielet programistyczny, a czynnościami zaimplementowanymi przez twórcę aplikacji. Istotna jest również świadomość i kontrola nad szkieletem programistycznym.

Systemy mapowania obiektowo-relacyjnego znane powszechnie jako systemy ORM (ang. Object-Relational Mapping), są odpowiedzialne za mapowanie rekordów bazodanowych na obiekty w konkretnym języku programowania [1]. Takie podejście znacząco przyspiesza implementację operacji bazodanowych takich jak: wstawianie danych, aktualizacja danych, usuwanie danych jak również odczytywanie danych.

W niniejszej pracy poddano analizie systemy mapowania obiektowo-relacyjnego wspierane przez platformę .NET. Wspomniana platforma została stworzona przez firmę Microsoft i obecnie stanowi jedną z najpopularniejszych platform programistycznych. Postawiono hipotezę badawczą, „szkielet NHibernate jest wydajniejszy niż Entity Framework Core w kontekście operacji DML". Trafność hipotezy zweryfikowano na podstawie przeprowadzonych badań, których wyniki przedstawiono w dalszej części artykułu.

2. Przegląd literatury

W niniejszym rozdziale zostaną zaprezentowane źródła literaturowe, których tematyka jest zbliżona do tematu niniejszej pracy. Pierwsza ze znalezionych pozycji literaturowych to artykuł zatytułowany "Wydajność pracy z bazami danych w aplikacjach ASP.NET MVC", autorstwa Pawła Borysa i Beaty Pańczyk [2]. W tym artykule porównywane są szkielety Entity Framework 6.1.3 i NHibernate w wersji 4.6.1, dla systemów bazodano-

wych MySQL oraz MSSQL. Analiza przeprowadzona została pod kątem wydajności czasowej jak również pamięciowej dla operacji wybierania oraz wstawiania danych. W większości scenariuszy testowych wydajniejszy okazał się szkielet NHibernate.

Kolejną pozycją jest prelekcja pod tytułem "Fetch performance comparison of object relational mapper in .NET platform" wygłoszona przez Witoona Wiphusitphunpola oraz Thitiporna Lertrudachakula podczas czternastej międzynarodowej konferencji ECTI-CON [3]. Jednym z wykorzystanych szkieletów był Entity Framework Core w wersji 1.1.0. Zostały zbadane operacje wybierania danych w jednym wątku oraz w wielu wątkach.

Podczas trzeciej międzynarodowej konferencji dotyczącej obiektów i baz danych, która odbyła się w 2010 roku we Frankfurcie, przedstawione zostały wyniki badań Stevicy Cvetkovicia i Dragana Jankovicia na temat "Analiza porównawcza funkcji i wydajności narzędzi ORM w środowisku .NET" [4]. Autorzy w pracy użyli szkieletów NHibernate 2.0.1 i Entity Framework 4, a środowiskiem testowym był MS SQL Server 2005. Testowane były operacje wstawiania, wybierania i usuwania rekordów. Dla operacji wstawiania szybszym szkieletem okazał się Entity Framework natomiast dla operacji usuwania NHibernate.

Ostatnim źródłem konferencyjnym jest prelekcja pochodząca z Międzynarodowej konferencji Beyond Databases Architectures and Structures pod tytułem "Analiza wydajnościowa szkieletów mapowania obiektowo relacyjnego dla platformy .NET" [5]. Wykorzystane narzędzia to Entity Framework 5.0, NHibernate 3.3.1, natomiast platformy testowe to Microsoft SQL Server 2012 i Postgre SQL 9.2. Testy wydajnościowe zostały przeprowadzone dla operacji wstawiania, wybierania, modyfikacji i usuwania rekordów z tabel. W testach wstawiania, jak również usuwania, wydajniejszy okazał się szkielet NHibernate, natomiast dla operacji modyfikacji szkielet Entity Framework.

W artykule "Porównanie wydajności metod CRUD (przyp. ang. Create, Read, Update, Delete) przy użyciu systemów mapowania obiektowo relacyjnego dla .NET", porównane zostały szkielety Entity Framework Core 2.2, NHibernate 5.2.3 i Dapper 1.50.5 [6]. Platformą testową był MS SQL, a testowane operacje to wstawianie, wybieranie, modyfikacja i usuwanie rekordów dla relacji jeden do jednego oraz jeden do wielu. Testowano czas wykonania operacji i wykorzystaną pamięć. W testach wstawiania oraz w niektórych testach usuwania rekordów, wydajniejszy okazał się szkielet NHibernate, natomiast w testach modyfikacji oraz w części testów usuwania rekordów lepsze wyniki uzyskał szkielet Entity Framework Core.

3. Obiekty badań

3.1. Entity Framework Core w wersji 2.2.4

Entity Framework Core jest szkieletem programistycznym klasy ORM autorstwa firmy Microsoft [7]. Jego wykorzystanie pozwala niemalże na eliminację konieczności implementacji operacji dostępu do danych z baz danych. Obecnie istnieją dwie wersje omawianego systemu mapowania obiektowo-relacyjnego. Podstawowa wersja Entity Framework 6 jest rozwijana i stabilizowana od kilku lat, jednakże wspiera jedynie urządzenia z systemem Windows. Drugą, znacznie młodszą wersją, jest Entity Framework Core, którego platformą docelową jest .NET Core zapewniający wieloplatformowość. Cechuje go lekkość i rozszerzalność, a sam projekt jest otwartoźródłowy, co pozwala na lepsze wsparcie społeczności programistów.

3.2. NHibernate w wersji 5.1.1

NHibernate to otwartoźródłowy szkielet programistyczny dla platformy .NET [8]. Szkielet ten jest implementowany w ramach projektu Hibernate przez społeczność programistyczną. Zapewnia wsparcie dla wielu popularnych relacyjnych systemów baz danych. Pozwala na mapowanie obiektów ze strony aplikacji na rekordy i relacje po stronie systemu bazodanowego. Umożliwia tworzenie więzów integralności dla tabel oraz struktur bazodanowych. Konfiguracja odbywa się z poziomu pliku konfiguracyjnego XML lub poprzez kod C#.

4. Metoda badawcza

W celu weryfikacji poprawności postawionej hipotezy przeprowadzono badania wydajności szkieletów Entity Framework Core oraz NHibernate za pomocą zaimplementowanej w tym celu aplikacji desktopowej. Główne zadanie aplikacji stanowiło przeprowadzenie testów według wskazanych parametrów. Dodatkowo do rozwiązania zostały dodane funkcjonalności pozwalające na automatyzację wizualizacji wyników, obliczanie parametrów statystycznych oraz utrwalanie wyników.

Testy wydajności wykonane we wspomnianej wyżej aplikacji zostały przeprowadzone według przyjętych scenariuszy testowych przedstawionych w tabeli 1. Ze względu na duży narzut czasowy wykonywania pojedynczego scenariusza oraz deterministyczne zachowanie szkieletów podczas testów, zdecydowano o wykonaniu jednego powtórzenia testu dla każdego scenariusza.

Tabela 1: Scenariusze testowe

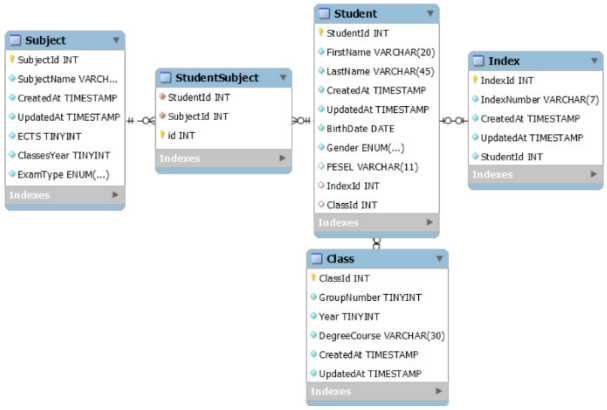
Numer scenariusza	Liczba rekordów/powtórzeń	Operacja
1.	500	Create
2.		Update
3.		Delete
4.	5000	Create
5.		Update
6.		Delete

W celu eliminacji wpływu jednego testu na drugi, każdą z trzech rodzajów operacji (create, update, delete) testowano oddzielnie. Aplikacja umożliwia wykonywanie testów masowych oraz pojedynczych. Liczby wskazane w drugiej kolumnie tabeli 1, oznaczają liczbę powtórzeń testu pojedynczego oraz liczbę rekordów dla jakiej wykonano test masowy.

Przyjęto następujące progi liczby rekordów/liczby powtórzeń:

- 500 – mała liczba rekordów/powtórzeń;
- 5000 – duża liczba rekordów/powtórzeń.

Na rysunku 1 przedstawiono strukturę bazy danych wykorzystaną w testach. Podczas projektowania bazy danych głównym założeniem było wykorzystanie wszystkich binarnych relacji bazodanowych. W celu umożliwienia testowania poszczególnych relacji oddzielnie, przyjęto pewne uproszczenia w modelu bazy danych. Jednym z nich jest opcjonalność relacji pomiędzy tabelą *Student* a *Index*. Inny przykład stanowi brak unikalności dla kolumn reprezentujących numer indeksu i numer PESEL.



Rysunek 1: Schemat testowej bazy danych

W tabeli 2 przedstawiono parametry techniczne sprzętu wykorzystanego do przeprowadzenia testów wydajności.

Tabela 2: Parametry sprzętu testowego

Nazwa modelu	Dell Inspiron 5570
System operacyjny	Windows 10 Home
Procesor	Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz 1.80GHz
Pamięć RAM	16 GB
Typ systemu	x64
Dysk	SSD – 256GB, HDD – 1 TB

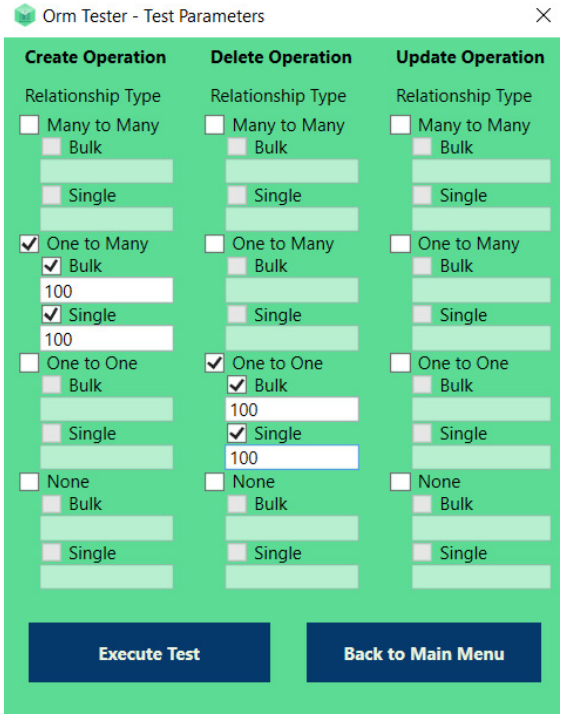
5. Aplikacja wspomagająca analizę wydajności

Aplikacja wspomagająca analizę wydajności powstała jako aplikacja desktopowa WPF dla platformy .NET Framework. Zostały w niej wykorzystane trzy utworzone wcześniej biblioteki: biblioteka zawierająca główny moduł testujący oraz dwie biblioteki implementujące operacje bazodanowe według określonego interfejsu dla Entity Framework Core i NHibernate. Baza danych została utworzona za pomocą silnika bazodanowego SQL Server Express LocalDB w wersji 13.0.4001, wbudowanego w narzędzie Visual Studio.

Aplikacja posiada takie funkcjonalności jak:

- parametryzacja testu;
- obliczenie parametrów statystycznych;
- zapis/odczyt parametrów statystycznych do/z pliku;
- eksport pomiarów do arkusza kalkulacyjnego;
- generowanie wykresów;
- zapis wykresów do pliku.

Na rysunku 2 przedstawiono okno parametryzacji testów. Składa się ono z trzech kolumn, z których każda zawiera parametry testów określonej operacji. Możliwe jest wykonanie testu dla następujących rodzajów relacji: jeden do jednego, jeden do wielu, wiele do wielu oraz dla pojedynczej tabeli. Za pomocą przycisków typu *checkbox* wskazywana jest również informacja czy ma zostać wykonany test masowy czy też pojedynczy. W polach tekstowych określana jest liczba powtórzeń/rekordów wykorzystanych w danym teście.



Rysunek 2: Okno parametryzacji pomiaru

Na listingu 1 przedstawiono implementację pomiaru czasu wykonania operacji tworzenia rekordu w relacji jeden do jednego dla Entity Framework Core. Na początku generowane są dane testowe, następnie uruchamiany jest pomiar czasu, po czym obiekty dodawane są do kontekstu. Wywołanie metody *SaveChanges()* powoduje utworzenie transakcji wstawiającej rekordy do bazy danych. Na końcu następuje zatrzymanie pomiaru czasu i zwracany jest wynik. Pozostałe operacje zostały zaimplementowane w sposób analogiczny.

Na listingu 2 przedstawiono implementację identycznej operacji jak w poprzednim akapicie z tym, że dla szkieletu NHibernate. Na początku, podobnie jak w przypadku Entity Framework Core, tworzone są dane testowe i uruchamiany jest pomiar czasu. Następnie w sposób jawny tworzona jest transakcja i w niej obiekty dodawane są do sesji. Transakcja zatwierdzana jest za pomocą metody *Commit()*. Na końcu zatrzymywany jest pomiar czasu oraz następuje zwrócenie wyniku.

Listing 1: Implementacja pomiaru czasu w Entity Framework Core

```
public TimeSpan SingleCreateOneToOne()
{
    Index index = TestDataFactory.GetIndex();
    Student student = TestDataFactory.GetStudent();

    Stopwatch stopwatch = new Stopwatch();
    stopwatch.Start();

    db.Indexes.Add(index);
    db.Students.Add(student);
    student.Index = index;
    db.SaveChanges();

    stopwatch.Stop();

    return stopwatch.Elapsed;
}
```

Listing 2: Implementacja pomiaru czasu operacji w NHibernate

```
public TimeSpan SingleCreateOneToOne()
{
    var student = NHibernateDataGenerator.GetStudent();
    var index = NHibernateDataGenerator.GetIndex(1);

    var watch = new Stopwatch();
    watch.Start();

    using (var transaction = session.BeginTransaction())
    {

        student.IndexId = index;
        session.Save(index);
        session.Save(student);

        transaction.Commit();
    }

    watch.Stop();
    return watch.Elapsed;
}
```

6. Wyniki badań

Wyniki badań z podziałem na relacje przedstawiono w kolejnych podrozdziałach zgodnie z podziałem na operacje. Parametry, które zaprezentowano w tabelach to średni czas wykonania operacji dla pojedynczego rekordu oraz odchylenie standardowe dla testów pojedynczych. Wykresy ilustrują współczynnik zmienności.

6.1. Operacja wstawiania rekordów

W tabeli 3 został zaprezentowany średni czas wykonania pojedynczej operacji dla scenariusza wstawiania pięciuset rekordów. W przedstawionym scenariuszu w większości testów wydajniejszym szkieletem okazał się NHibernate. Jedynym wyjątkiem jest test masowy dla relacji wiele do wielu. Warto zauważyć, że w testach masowych różnice pomiędzy szkieletami były nieduże.

W tabeli 4 zaprezentowano średni czas wykonania pojedynczej operacji w scenariuszu wstawiania pięciu tysięcy rekordów. W przedstawionych wynikach widoczny jest fakt osiągnięcia przewagi wydajności przez szkielet NHibernate w testach pojedynczych. W testach masowych wartości te są porównywalne. Można zauważyć również drastyczną różnicę pomiędzy testami pojedynczymi a masowymi, które w skrajnym przypadku są kilkaset razy mniejsze w testach masowych.

W przedstawionych scenariuszach testowych widoczny jest wzrost średnich wartości pomiędzy scenariuszami w testach pojedynczych, natomiast w przypadku testów masowych wartości średnie zmalały. Dodatkowo wartości odchylenia standardowego są większe w przypadku szkieletu Entity Framework Core niż szkieletu NHibernate.

Tabela 3: Czas wykonania pojedynczej operacji dla scenariusza wstawiania pięciuset rekordów

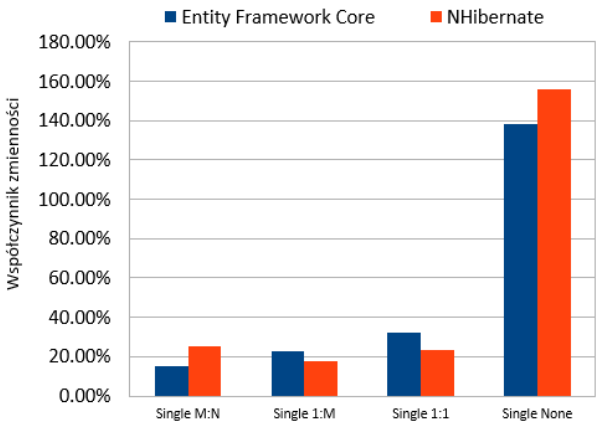
Nazwa testu	Czas wykonania pojedynczej operacji - Entity Framework Core (ms)	Czas wykonania pojedynczej operacji -NHibernate (ms)
Pojedynczy M:N	79,371 ± 12,14	31,243 ± 7,91
Pojedynczy 1:N	53,417 ± 12,07	24,03 ± 4,27
Pojedynczy 1:1	32,33 ± 10,4	13,175 ± 3,07
Pojedynczy brak relacji	10,835 ± 14,99	5,599 ± 8,72
Masowy M:N	1,184	1,231
Masowy 1:N	1,622	1,282
Masowy 1:1	3,078	1,972
Masowy brak relacji	1,371	0,986

Tabela 4: Czas wykonania pojedynczej operacji dla scenariusza wstawiania pięciu tysięcy rekordów

Nazwa testu	Czas wykonania pojedynczej operacji - Entity Framework Core (ms)	Czas wykonania pojedynczej operacji –NHibernate (ms)
Pojedynczy M:N	678,461 ± 90,42	243,566 ± 45,27
Pojedynczy 1:N	405,371 ± 67,73	117,22 ± 28,59
Pojedynczy 1:1	185,509 ± 59,13	50, 371 ± 14,18
Pojedynczy brak relacji	41,521 ± 23,56	13,359 ± 7,26
Masowy M:N	0,7	0,795
Masowy 1:N	0,992	0,723
Masowy 1:1	1,288	1,122
Masowy brak relacji	0,632	0,577

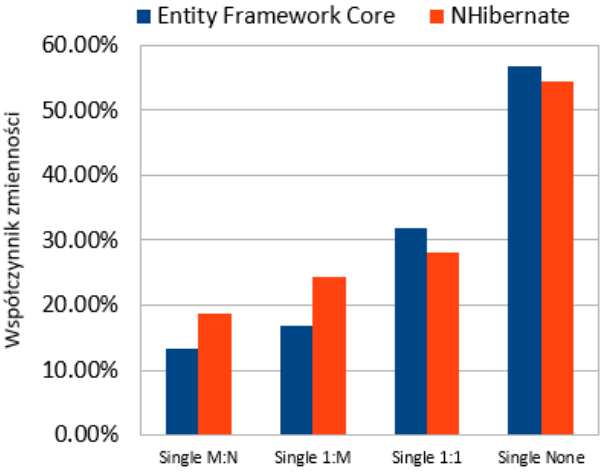
Na rysunku 3 przedstawiony został wykres wartości współczynnika zmienności w scenariuszu wstawiania pięciuset rekordów. Współczynnik ten jest to miara względna odchylenia standardowego oraz średniego czasu wstawienia pojedynczego rekordu i może być interpretowana jako stabilność. Uzyskane wyniki dla tego scenariusza są zbliżone dla obu szkieletów. Najmniejszą stabilnością charakteryzuje się operacja wsta-

wiania przy braku relacji. W pozostałych testach wartości są dużo mniejsze



Rysunek 3: Wartość współczynnika zmienności w testach pojedynczych dla scenariusza wstawiania pięciuset rekordów

Na rysunku 4 został przedstawiony współczynnik zmienności dla przeprowadzonych testów pojedynczych w scenariuszu wstawiania pięciu tysięcy rekordów. Wartości dla testów relacji wiele do wielu i jeden do wielu są mniejsze dla szkieletu Entity Framework Core, w pozostałych przypadkach mniejsze wartości tego współczynnika osiągnął szkielet NHibernate. W ogólności oba szkielety uzyskały podobny poziom stabilności.



Rysunek 4: Wartość współczynnika zmienności w testach pojedynczych dla scenariusza wstawiania pięciu tysięcy rekordów

6.2. Operacja modyfikacji rekordów

W tabeli 5 został przedstawiony średni czas wykonania pojedynczej operacji dla scenariusza modyfikacji pięciuset rekordów. W tym scenariuszu średni czas wykonania operacji dla szkieletu NHibernate okazał się dużo mniejszy niż dla szkieletu Entity Framework Core. Różnice te są najbardziej widoczne w testach masowych.

Tabela 5: Czas wykonania pojedynczej operacji dla scenariusza modyfikacji pięciuset rekordów

Nazwa testu	Czas wykonania pojedynczej operacji - Entity Framework Core (ms)	Czas wykonania pojedynczej operacji –NHibernate (ms)
Pojedynczy M:N	80,503 ± 12	33,355 ± 5,47
Pojedynczy 1:N	53,191 ± 11,9	20,691 ± 4,02
Pojedynczy 1:1	32,093 ± 10	11,519 ± 2,78
Pojedynczy brak relacji	9,856 ± 3,98	4,721 ± 1,54
Masowy M:N	1,305	0,152
Masowy 1:N	3,995	0,354
Masowy 1:1	3,292	0,24
Masowy brak relacji	3,468	0,142

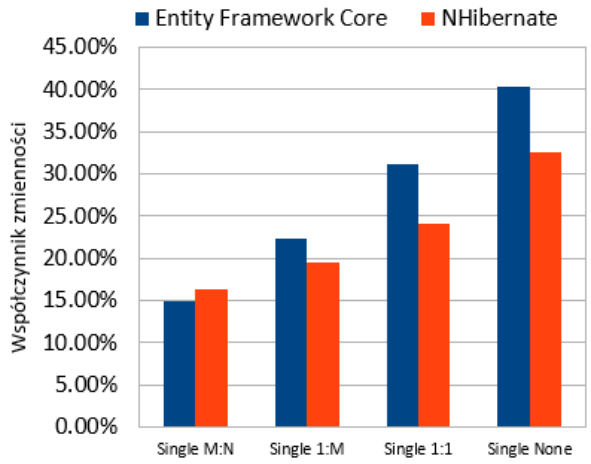
W tabeli 6 przedstawiono średni czas wykonania pojedynczej operacji w scenariuszu modyfikacji pięciu tysięcy rekordów. Wydajniejszym szkieletem w niniejszym teście okazał się szkielet NHibernate, jednak w testach masowych różnice są nieduże.

W obu scenariuszach średnie czasy wykonania operacji w testach pojedynczych okazały się zdecydowanie większe od testów masowych

Tabela 6: Czas wykonania pojedynczej operacji dla scenariusza modyfikacji pięciu tysięcy rekordów

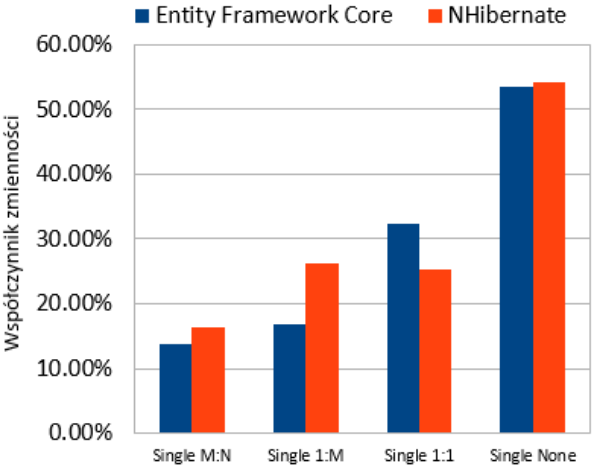
Nazwa testu	Czas wykonania pojedynczej operacji - Entity Framework Core (ms)	Czas wykonania pojedynczej operacji –NHibernate (ms)
Pojedynczy M:N	694,553 ± 96,43	269,033 ± 43,72
Pojedynczy 1:N	406,224 ± 68,58	140,966 ± 36,98
Pojedynczy 1:1	184,931 ± 59,99	64,783 ± 16,37
Pojedynczy brak relacji	42,112 ± 22,51	25,168 ± 13,66
Masowy M:N	0,388	0,321
Masowy 1:N	1,325	0,426
Masowy 1:1	0,904	0,152
Masowy brak relacji	0,84	0,143

Na rysunku 5 przedstawiony został wykres wartości współczynnika zmienności dla scenariusza modyfikacji pięciuset rekordów. Poza testem dla relacji wiele do wielu stabilniejszym szkieletem okazał się NHibernate. Różnice pomiędzy szkieletami nie są jednak duże, a wartość współczynnika zmienności nie przekracza 40%.



Rysunek 5: Wartość współczynnika zmienności w testach pojedynczych dla scenariusza modyfikacji pięciuset rekordów

Na rysunku 6 zostały przedstawione wartości współczynnika zmienności dla testów pojedynczych w scenariuszu modyfikacji pięciu tysięcy rekordów. Dla trzech z czterech relacji stabilniejszym szkieletem okazał się Entity Framework Core – wyniki te zostały osiągnięte dla relacji wiele do wielu, jeden do wielu oraz przy modyfikacji niepowiązanego rekordu.



Rysunek 6: Wartość współczynnika zmienności w testach pojedynczych dla scenariusza modyfikacji pięciu tysięcy rekordów

6.3. Operacja usuwania rekordów

W tabeli 7 został przedstawiony średni czas wykonania pojedynczej operacji w scenariuszu usuwania pięciuset rekordów. W większości testów niniejszego scenariusza wydajniejszym szkieletem okazał się NHibernate. Entity Framework Core był szybszy w pojedynczym teście usuwania przy braku relacji oraz w teście masowym dla relacji jeden do wielu.

W tabeli 8 przedstawiono średni czas wykonania pojedynczej operacji w scenariuszu usuwania pięciu tysięcy rekordów. Zdecydowanie wydajniejszym szkieletem ponownie okazał się NHibernate, jednakże różnice w testach masowych nie okazały się duże. W niniejszym scenariuszu można zauważyć duże różnice wartości odchylenia standardowego między szkieletami w pojedynczych testach relacji jeden do wielu oraz wiele do wielu. W pozostałych dwóch testach wartości są zbliżone.

Ponownie średni czas usuwania pojedynczego rekordu w obu scenariuszach w testach masowych jest mniejszy niż w testach pojedynczych. W testach masowych wraz ze wzrostem liczby rekordów wydajność operacji również rośnie.

Tabela 7: Czas wykonania pojedynczej operacji dla scenariusza usuwania pięciuset rekordów

Nazwa testu	Czas wykonania pojedynczej operacji - Entity Framework Core (ms)	Czas wykonania pojedynczej operacji – NHibernate (ms)
Pojedynczy M:N	35,361 ± 10,67	14,202 ± 3,94
Pojedynczy 1:N	10,988 ± 4,75	6,061 ± 1,89
Pojedynczy 1:1	3,626 ± 1,06	3,432 ± 0,54
Pojedynczy brak relacji	2,374 ± 0,5	2,473 ± 0,23
Masowy M:N	1,665	0,156
Masowy 1:N	0,207	0,533
Masowy 1:1	6,876	1,915
Masowy brak relacji	4,112	0,169

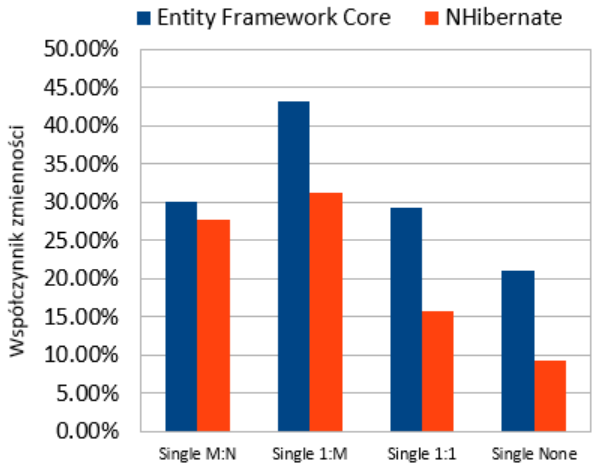
Tabela 8: Czas wykonania pojedynczej operacji dla scenariusza usuwania pięciu tysięcy rekordów

Nazwa testu	Czas wykonania pojedynczej operacji - Entity Framework Core (ms)	Czas wykonania pojedynczej operacji – NHibernate (ms)
Pojedynczy M:N	221,002 ± 60,37	62,924 ± 21,4
Pojedynczy 1:N	53,169 ± 31,98	14,622 ± 7,51
Pojedynczy 1:1	2,913 ± 0,99	2,179 ± 0,4
Pojedynczy brak relacji	2,124 ± 0,55	1,606 ± 0,56
Masowy M:N	0,539	0,088
Masowy 1:N	0,224	0,173
Masowy 1:1	2,425	2,838
Masowy brak relacji	0,945	0,086

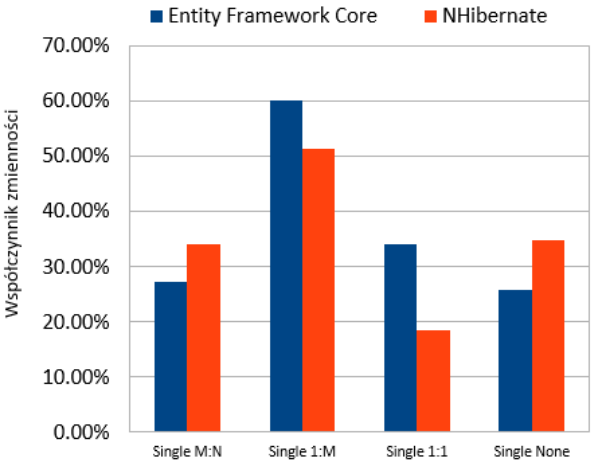
Na rysunku 7 przedstawiono wartość współczynnika zmienności w scenariuszu usuwania pięciuset rekordów. We wszystkich testach stabilniejszy okazał się szkielet NHibernate, jednakże w tym scenariuszu wartości te znacząco różnią się pomiędzy szkieletami dla wszystkich relacji poza relacją wiele do wielu.

Na rysunku 8 przedstawiono wartości współczynnika zmienności dla scenariusza usuwania pięciu tysięcy rekordów. W testach relacji wiele do wielu oraz dla rekordów niepowiązanych, stabilniejszym szkieletem okazał się szkielet Entity Framework Core, natomiast

dla relacji jeden do wielu i jeden do jednego mniejszą wartość współczynnika osiągnął szkielet NHibernate.



Rysunek 7: Wartość współczynnika zmienności w testach pojedynczych dla scenariusza usuwania pięciuset rekordów



Rysunek 8: Wartość współczynnika zmienności w testach pojedynczych dla scenariusza usuwania pięciu tysięcy rekordów

7. Analiza porównawcza

Konfiguracja szkieletów programistycznych wygląda podobnie w obu porównywanych szkieletach programistycznych [7, 8]. Zarówno w Entity Framework Core jak i NHibernate, zależności do projektu można dodać za pomocą managera pakietów NuGet. Oba szkielety wspierają obsługę najpopularniejszych systemów bazodanowych takich jak Sql Server, Sql Lite, PostgreSQL, MySql, Oracle DB, jednakże dostęp do niektórych z nich w Entity Framework Core jest płatny.

W Entity Framework Core modele definiowane są jako klasy składające się z właściwości (ang. *properties*) [7, 8]. Modele opisywane są w dwojaki sposób – za pomocą atrybutów lub tak zwanego *fluent API*. W NHibernate definiowanie modeli przebiega w podobny sposób z taką różnicą, że właściwości muszą zostać oznaczone słowem kluczowym *virtual*. Modele opisywane są za pomocą plików XML lub z wykorzystaniem biblioteki FluentNHibernate.

Implementacja operacji bazodanowych jest mniej złożona w szkielecie Entity Framework Core [7, 8]. W NHibernate operacje te implementowane są w bardziej niskopoziomowy sposób – zarządzanie transakcjami spoczywa na programiście podczas, gdy w Entity Framework Core operacje te wykonywane są niejawnie po stronie szkieletu. Ponadto, wartą zauważenia cechą jest brak wsparcia dla operacji masowych przez NHibernate.

Zarządzenie schematem bazy danych w Entity Framework Core odbywa się za pomocą mechanizmu migracji, który pozwala na generowanie schematu na podstawie modeli i klasy kontekstu (klasy dziedziczącej po *DbContext*) [7, 8]. Powyższa funkcjonalność wykorzystywana jest poprzez wiersz poleceń, a wygenerowane pliki mogą aktualizować strukturę bazy danych zarówno podczas implementacji jak i w czasie wykonywania programu. W NHibernate schemat bazy danych tworzony jest na podstawie plików mapujących, których konfiguracje zawarto w kodzie aplikacji.

Oba szkielety cechuje nowoczesność oraz wspieranie typowych dla szkieletów ORM funkcjonalności takich jak [7, 8, 9, 10]:

- *Forward Engineering*;
- *Reverse Engineering*;
- Surowe zapytania SQL;
- Zapytania asynchroniczne;
- Ładowanie leniwe (ang. *Lazy Loading*);
- Ładowanie chciwe (ang. *Eager Loading*);
- Śledzenie zmian (ang. *Change Tracking*);
- Zapytania LINQ.

NHibernate nie wspiera popularnego mechanizmu wstrzykiwania zależności (ang. *Dependency Injection*).

8. Podsumowanie

Podsumowując badania wydajności, w oparciu o wykresy i tabele zaprezentowane w rozdziale szóstym, zdecydowanie lepszy wynik uzyskał szkielet NHibernate, co udowadnia postawioną hipotezę badawczą - "szkielet NHibernate jest wydajniejszy niż Entity Framework Core w kontekście operacji DML". W testach pojedynczych różnice pomiędzy szkieletami okazały się znaczące, natomiast w testach masowych znikome. Warta odnotowania jest również duża różnica w wynikach pomiędzy testami masowymi oraz pojedynczymi. Współczynnik zmienności prezentuje się na zbliżonym poziomie we wszystkich testach pojedynczych, co wskazuje na podobny poziom stabilności obu porównywanych szkieletów ORM.

Literatura

[1] Object-relational mapping, https://en.wikipedia.org/wiki/Object-relational_mapping, [16.06.2020]

[2] P. Borys, B. Pańczyk: Wydajność pracy z bazami danych w aplikacjach ASP.NET MVC. Journal of Computer Science Institute 6, 2018.

[3] D. Zmaranda, L-L. Pop-Fele, C. Győrödi, R. Győrödi, G. Pecherle: Performance Comparison of CRUD Methodsusing NET Object Relational Mappers: A Case

Study (IJACSA) International Journal of Advanced Computer Science and Applications,Vol. 11, No.1, 2020.

[4] W. Wiphusitphunpol, T. Letrusdachakul: Fetch performance comparison of object relational mapper in .NET platform. [W]: 14th International Conference on Electrical Engineering/Electronics, IEEE, Computer, Telecommunications and Information Technology (ECTI-CON), Phuket, Tajlandia 7 listopada 2017.

[5] S. Cvetković, D. Janković: A Comparative Study of the Features and Performance of ORM Tools in a .NET Environment. [W]: Objects and Databases ICOODB 2010. Lecture Notes in Computer Science, vol 6348. Springer, Berlin, Heidelberg. Frankfurt, Niemcy. 28-30 września 2010.

[6] A. Gruca, P. Podsiadło: Performance Analysis of .NET Based Object–Relational Mapping Frameworks. [W]: Beyond Databases, Architectures, and Structures. BDAS 2014. Communications in Computer and Information Science, vol 424. Springer, Cham. Ustroń Polska, 27-30 maja 2014.

[7] Dokumentacja szkieletu programistycznego Entity Framework Core, https://docs.microsoft.com/en-us/ef/core/, [22.04.2020].

[8] Dokumentacja szkieletu programistycznego NHibernate, https://nhibernate.info/, [29.03.2020].

[9] Dokumentacja biblioteki FluentNHibernate https://github.com/FluentNHibernate/fluent-nhibernate/wiki, [10.05.2020].

[10] Entity Framework Core Tutorial, https://www.entityframeworktutorial.net/efcore/entity-framework-core.aspx, [25.04.2020].