JOURNAL
COMPUTER SCIENCES INSTITUTE

JCSI 31 (2024) 97–102
Received: 17 March 2024
Accepted: 4 June 2024

Comparative Analysis of ORM Systems for the .NET Platform

Analiza porównawcza systemów ORM dla platformy .NET

Tomasz Wiatrowski*

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

Abstract

This article presents a comparative study of three popular ORM (Object-Relational Mapping) systems used in the .NET technology: Entity Framework Core, Dapper, and LINQ to DB. The analysis is divided into two main parts: theoretical, focusing on research found in literature, and practical, where an application was implemented to assess the performance of each system and their memory consumption. The results obtained were presented in the form of graphs. Dapper emerged as the most efficient system and consumed the least memory in the majority of test cases.

Keywords: ORM; Entity Framework Core; Dapper; LINQ to DB

Streszczenie

Niniejszy artykuł przedstawia badania porównawcze trzech popularnych systemów ORM używanych w technologii .NET jakimi są: Entity Framework Core, Dapper oraz LINQ to DB. Analiza podzielona została na dwie główne części, teoretyczną - skupiającą się na badaniach zawartych w literaturze oraz praktyczną – w ramach której zaimplementowano aplikację pozwalającą na sprawdzenie wydajności poszczególnych systemów oraz ich zużycie pamięci. Uzyskane wyniki przedstawiono w formie wykresów. System Dapper okazał się najbardziej wydajny oraz zużywał najmniej pamięci w większości przypadków testowych.

Słowa kluczowe: ORM; Entity Framework Core; Dapper; LINQ to DB

*Corresponding author

Email address: tomasz.wiatrowski@pollub.edu.pl (T. Wiatrowski)

1. Wstęp

Współczesne aplikacje wymagają efektywnych rozwiązań do zarządzania danymi w bazach danych. Wybór odpowiedniego narzędzia pełni kluczową rolę dla wydajności, efektywności i skalowalności aplikacji. Wśród rozwiązań, które są niezbędnym elementem procesu tworzenia oprogramowania, znajdują się systemy mapowania obiektowo-relacyjnego (ORM), które umożliwiają programistom pracę z bazami danych w bardziej intuicyjny sposób [1].

Badania zostały podjęte w celu sprawdzenia, które systemy ORM dla platformy .NET sprawują się najlepiej dla danych operacji. Wybór odpowiedniego systemu ORM może mieć znaczący wpływ na wydajność aplikacji. Rezultaty badań pozwoliły na określenie zalet i wad każdego z systemów oraz pomogły zweryfikować hipotezę „System ORM Dapper jest bardziej wydajny w operacjach DML niż Entity Framework Core i LINQ to DB".

W platformie .NET, istnieje wiele systemów ORM umożliwiających efektywną interakcję między bazą danych a obiektowymi modelami aplikacji. Jednym z najbardziej znanych i często stosowanych jest Entity Framework Core, stworzony przez Microsoft. Jego funkcjonalność ułatwia tworzenie zaawansowanych aplikacji wykorzystujących bazy danych [2]. Jednak oprócz tego wiodącego rozwiązania, istnieje także wiele innych popularnych systemów ORM stworzonych przez społeczność. W tej pracy oprócz Entity Framework Core, skupiono się także na systemach Dapper oraz LINQ to DB. Oba te rozwiązania cieszą się uznaniem ze względu na prostotę obsługi oraz wydajność, co przekłada się na ich coraz szersze zastosowanie w projektach.

W pracy omówiono głównie aspekty takie jak wydajność poszczególnych systemów, która obejmuje szybkość zapisu, odczytu, aktualizacji oraz usuwania danych. Zbadano również zużycie pamięci poprzez poszczególne systemy ORM. Wyniki tych badań mogą dostarczyć bardzo cennych informacji i wskazówek dla programistów, którzy poszukują odpowiedniego narzędzia ORM do swojego projektu.

2. Przegląd literatury

Pierwszą z analizowanych pozycji jest artykuł [3]. Autorzy podjęli próbę porównania trzech szkieletów jakimi są Entity Framework, LINQ to SQL oraz Transact Queries pod kątem wydajności i bezpieczeństwa. Dla każdego z trzech szkieletów przeprowadzono jedenaście różnych typów zapytań i dokładnie zbadano czas wykonania oraz zużycie pamięci dla każdego z nich. Autorzy zauważyli, że zarówno Entity Framework jak i LINQ to SQL umożliwiają znacznie szybszą implementację operacji CRUD (Create, Read, Update, Delete) w porównaniu do Transact Queries. Z drugiej strony, wyniki pokazały, że Transact Query wykazuje się mniejszym zużyciem pamięci. Ponadto, dokonano analizy bezpieczeństwa, wprowadzając różne typy ataków SQL injection w celu oceny poziomu bezpieczeństwa każdego ze szkieletów. Rezultaty wykazały, że Transact Query jest bardziej podatny na ataki w porównaniu do LINQ to SQL i Entity Framework.

W artykule [4], autorzy skoncentrowali się na analizie systemów mapowania obiektowo- relacyjnego

w środowisku .NET 6. Celem badań było zbadanie wydajności trzech narzędzi ORM: Dapper, NHibernate oraz Entity Framework Core, w tym celu zbadano nie tylko czas przetwarzania i zużycie pamięci RAM, ale także zużycie CPU. Aby przeprowadzić badania, autorzy utworzyli oprogramowanie mierzące dane aspekty podczas wykonywania operacji odczytu, dodawania, aktualizacji, usuwania, wyszukiwania i sortowania. Wyniki wykazały, że Dapper osiąga najlepsze wyniki pod względem czasu przetwarzania w operacjach odczytu, usuwania, wyszukiwania i sortowania, podczas gdy Entity Framework Core osiąga najlepsze wyniki w operacjach dodawania i aktualizacji. Dapper okazał się również najlepszy pod kątem zużycia zasobów.

W artykule [5], autorzy przeprowadzili badanie empiryczne, mające na celu porównanie wydajności pobierania danych z bazy danych używając wybranych bibliotek mapowania obiektowo- relacyjnego w platformie .NET. Autorzy zauważyli, że w miarę wzrostu popularności platformy .NET, zostało opracowanych coraz więcej systemów ORM, co sprawia, że wybór odpowiedniego systemu staje się trudnym zadaniem. Do badania wykorzystano biblioteki: ADO.NET, Dapper i Entity Framework Core. Wyniki wykazały, że ADO.NET jest najszybszy podczas zapytań o kilka wyników, choć nieznacznie różni się od pozostałych narzędzi. Jednakże, jeśli chodzi o pobieranie wielu wyników, Entity Framework Core osiąga najszybszy czas zapytania, jednak zużywa najwięcej pamięci. Bazując na swoich wynikach, autorzy zalecają używać Entity Framework Core w aplikacjach, które nie wymagają szybkiego równoczesnego dostępu do danych. Uważają, że Dapper jest odpowiedni dla aplikacji, które wymagają szybkiego równoczesnego dostępu do danych, natomiast ADO.NET jest odpowiedni dla aplikacji działających w środowisku o ograniczonej pamięci.

Kolejną analizowaną pozycją jest artykuł [6], w którym autorzy przeprowadzili analizę porównawczą wydajności trzech najczęściej używanych technologii ORM w platformie .NET: Entity Framework Core, NHibernate i Dapper. Artykuł skupia się na badaniu wpływu danego ORM na wydajność aplikacji podczas wykonywania zapytań do bazy danych. W celu wykonania badań zaprojektowano odpowiednią architekturę testową, aby zapewnić spójność testów. Dokonano oceny wydajności czasu odpowiedzi i zużycia pamięci dla każdej technologii, wykorzystując te same operacje na bazie danych. Analizując wyniki, autorzy stwierdzili, że NHibernate jest najwydajniejszy dla operacji wstawiania wierszy, Entity Framework Core cechuje się największą wydajnością dla operacji aktualizacji i usuwania wierszy, z kolei Dapper jest najbardziej wydajny dla operacji pobierania wierszy.

Ostatnią analizowaną pozycją jest materiał konferencyjny [7]. Podczas konferencji, autorzy zaprezentowali przegląd technologii do pracy z relacyjnymi bazami danych w środowisku .NET. Następnie omówili badania mające na celu ocenę czasu wykonywania operacji wstawiania, wybierania, aktualizacji i usuwania danych w bazie danych, przy użyciu narzędzi ORM ADO.NET, Dapper i Entity Framework. Uzyskane wyniki wskazały na to, że ADO.NET jest najszybszy, jeśli chodzi o wstawianie, aktualizację i usuwanie wierszy, z kolei Dapper jest najbardziej wydajny, jeśli chodzi o pobieranie wierszy.

3. Obiekty badań

3.1. Entity Framework Core

Entity Framework Core (EF Core) jest otwartym szkieletem programistycznym opracowanym przez firmę Microsoft, służy do mapowania obiektowo-relacyjnego w aplikacjach .NET. Główny cel Entity Framework Core to ułatwienie interakcji z bazami danych w aplikacjach .NET poprzez eliminację potrzeby korzystania w sposób bezpośredni z języka SQL i skomplikowanych zapytań. Zamiast tego, programiści mogą pracować z obiektami reprezentującymi dane a EF Core zajmuje się przekształcaniem tych obiektów na odpowiednie struktury bazy danych [2].

3.2. Dapper

Dapper to biblioteka ORM napisana dla platformy .NET, która umożliwia łatwe i efektywne mapowanie zapytań SQL na obiekty w języku C#. Dapper oparty jest na prostocie i wydajności. Jego główną cechą jest to, że nie maskuje kodu SQL, co pozwala zachować większą kontrolę nad wykonywanymi zapytaniami. Dapper jest popularnym wyborem dla programistów preferujących szybkie operacje bazodanowe w aplikacjach .NET [8].

3.3. LINQ to DB

LINQ to DB to narzędzie ORM umożliwiające wykorzystanie języka zapytań LINQ do interakcji z bazami danych w języku C#. LINQ to DB stanowi swego rodzaju most pomiędzy kodem aplikacji a bazą danych, co umożliwia programistom wykonywanie zapytań do bazy danych w sposób bardziej czytelny i deklaratywny. Dodatkowo, LINQ to DB dostarcza mechanizmy optymalizacji zapytań, które starają się optymalizować zapytania SQL generowane przez LINQ, co przyczynia się do poprawy wydajności aplikacji [9].

4. Metody badawcze

Aby porównać wydajność systemów Entity Framework Core, Dapper oraz LINQ to DB, zaimplementowano aplikację w szkielecie .NET, której celem jest zbadanie średnich czasów wykonywania poszczególnych operacji bazodanowych z wykorzystaniem danego systemu ORM. W celu zapewnienia rzetelnych i wiarygodnych wyników dla każdego badanego systemu zastosowano identyczne warunki testowe:

- Użyto tego samego serwera bazy danych, a mianowicie Microsoft SQL Server, aby uniknąć wpływu różnych silników na wyniki badań;
- Wykorzystano identyczną strukturę tabel w bazie danych, co pozwoliło na porównywanie wydajności danych operacji w identycznych warunkach;
- Użyto takich samych danych dla każdego testu, eliminując tym samym wpływ różnych zestawów danych na wyniki;

- Do przeprowadzenia badań użyto identycznego sprzętu.

Tabela 1: Scenariusze testowe

Numer scenariusza testowego	Rodzaj operacji	Rodzaj relacji tabel	Liczba rekordów
1	Pobieranie danych	Bez relacji	1
2			1000
3		1:1	1
4			1000
5		1:N	1
6			1000
7		N:M	1
8			1000
9	Wstawianie danych	Bez relacji	1
10			1000
11		1:1	1
12			1000
13		1:N	1
14			1000
15		N:M	1
16			1000
17	Aktualizowanie danych	Bez relacji	1
18			1000
19		1:1	1
20			1000
21		1:N	1
22			1000
23		N:M	1
24			1000
25	Usuwanie danych	Bez relacji	1
26			1000
27		1:1	1
28			1000
29		1:N	1
30			1000
31		N:M	1
32			1000

5. Aplikacja wspomagająca analizę

W celu dokładnej oceny możliwości oraz efektywności poszczególnych systemów mapowania obiektowo-relacyjnego, konieczne jest posiadanie narzędzia umożliwiającego przeprowadzenie rzetelnych porównań i analiz. W tym celu, zaprojektowana została aplikacja wspomagająca analizę. Aplikacja umożliwia przeprowadzenie testów wydajnościowych dla systemów Entity Framework Core, Dapper oraz LINQ to DB.

Na potrzeby aplikacji zaprojektowano prostą bazę danych składającą się z tabeli bez relacji, tabel w relacji 1:1, tabel w relacji 1:N oraz tabel w relacji N:M. W celu zapewnienia tego, aby dane w bazie danych były zawsze takie same dla każdej testowanej operacji, zaimplementowano generator danych. Generator ten wykorzystuje bibliotekę Bogus, pozwalającą na generowanie przykładowych danych [10]. Przed każdym testem baza jest przywracana do pierwotnego stanu, dla każdego testu baza ma identyczne dane.

Do pomiaru czasu wykonywania poszczególnych operacji oraz zużycia pamięci zastosowano popularny w platformie .NET szkielet BenchmarkDotNet, który wielokrotnie powtarza każdą operację i oblicza średni czas wykonywania oraz średnie zużycie pamięci. Co więcej szkielet ten generuje tabelę z porównaniem czasów wykonywania danych operacji oraz zużycia pamięci, co ułatwia analizę wyników. W BenchmarkDotNet liczba iteracji jest dobierana dynamicznie w celu zapewnienia wiarygodnych i stabilnych wyników pomiarów. Jest to realizowane poprzez tzw. auto-skalowanie, które analizuje czasy wykonania poszczególnych metod w trakcie pierwszych kilku iteracji i na tej podstawie dostosowuje liczbę iteracji, aby uzyskać odpowiednio dokładne wyniki [11].

Aplikacja minimalizuje wpływ pamięci podręcznej (cache) na wyniki pomiarów poprzez wykonanie każdej iteracji w odseparowanej sesji, co zapobiega kumulowaniu efektu ciepłego startu. Oznacza to, że każda iteracja jest wykonywana w "zimnym starcie", tj. z pustą pamięcią podręczną. Dzięki temu zapewniana jest większa dokładność wyników, ponieważ nie ma efektu kumulacji pamięci podręcznej, który mógłby wpłynąć na pomiary.

Listing 1: Implementacja testów wydajnościowych przy użyciu BenchmarkDotNet

```
/// <summary>
///     Tests getting multiple rows from table with no relation
///     using entity framework core.
/// </summary>
[Benchmark]
0 references
public void GetMultipleRowsFromTableWithNoRelationUsingEntityFrameworkCore()
{
    using var dbContext = new AppDbContext(DbContextOptions);

    var movies = dbContext.Movies.ToList();
}

/// <summary>
///     Tests getting multiple rows from table with no relation using dapper.
/// </summary>
[Benchmark]
0 references
public void GetMultipleRowsFromTableWithNoRelationUsingDapper()
{
    const string query = "SELECT * FROM Movies";

    using IDbConnection connection = new SqlConnection(ConnectionString);

    connection.Open();

    var movies = connection.Query<Movie>(query).ToList();
}

/// <summary>
///     Tests getting multiple rows from table with no relation using Linq2Db.
/// </summary>
[Benchmark]
0 references
public void GetMultipleRowsFromTableWithNoRelationUsingLinq2Db()
{
    using var dataConnection = new AppDataConnection();
    var movies = dataConnection.Movie.ToList();
}
```

Na Listingu 1 przedstawiono metody testujące wydajność operacji pobierania wielu wierszy na raz z tabeli bez relacji przy użyciu systemów Entity Framework Core,

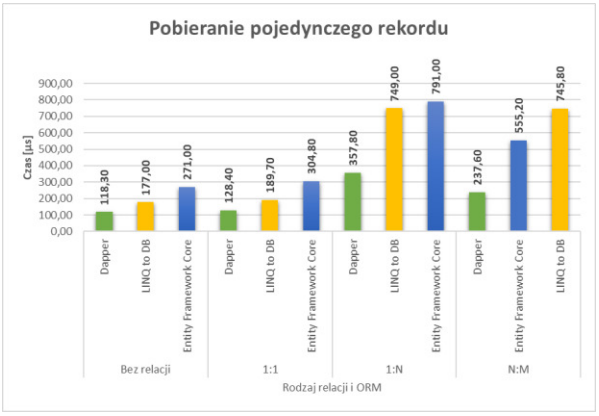
Dapper oraz LINQ to DB. Metody te oznaczono specjalnym atrybutem [Benchmark] dzięki czemu biblioteka BenchmarkDotNet wie, które metody należy poddać testom. W analogiczny sposób stworzone zostały klasy i metody testujące dla innych scenariuszy testowych.

6. Wyniki badań

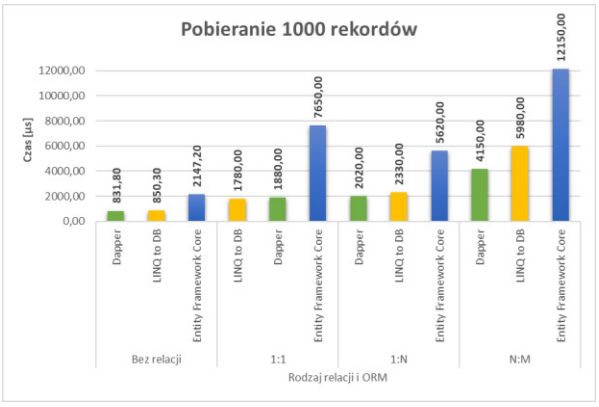
W poniższym rozdziale przedstawiono wykresy słupkowe obrazujące średni czas wykonywania poszczególnych operacji w danym typie relacji oraz średnie zużycie pamięci dla danej operacji.

6.1. Średnie czasy wykonywania

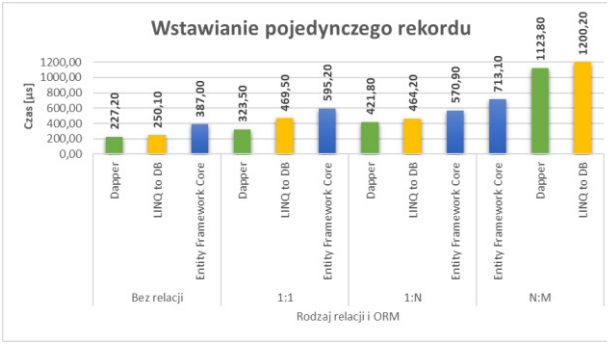
Średnie czasy wykonywania przedstawiono na Rysunkach 1-8.



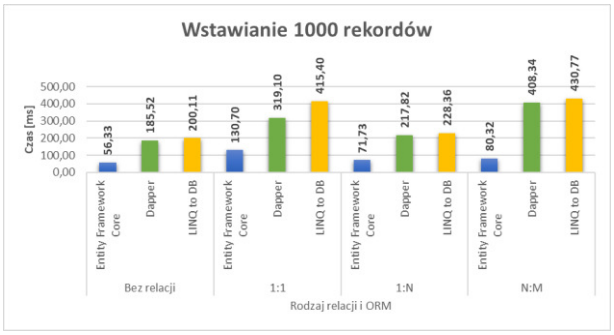
Rysunek 1: Średni czas wykonywania operacji pobierania pojedynczego rekordu.



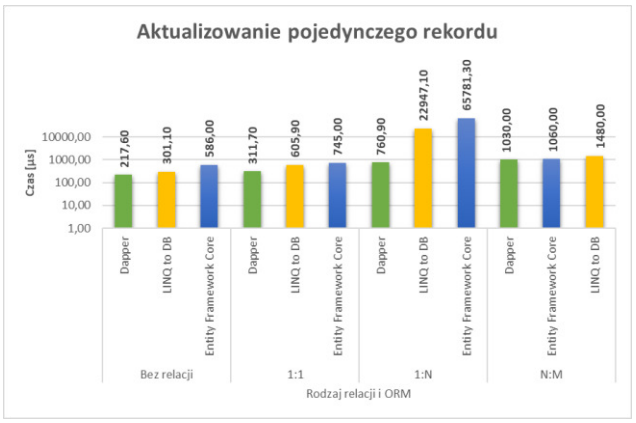
Rysunek 2: Średni czas wykonywania operacji pobierania 1000 rekordów.



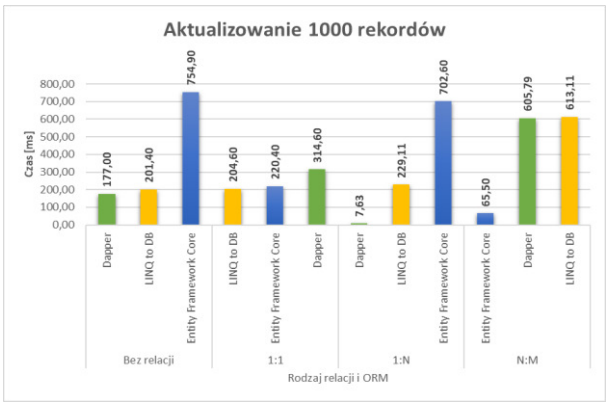
Rysunek 3: Średni czas wykonywania operacji wstawiania pojedynczego rekordu.



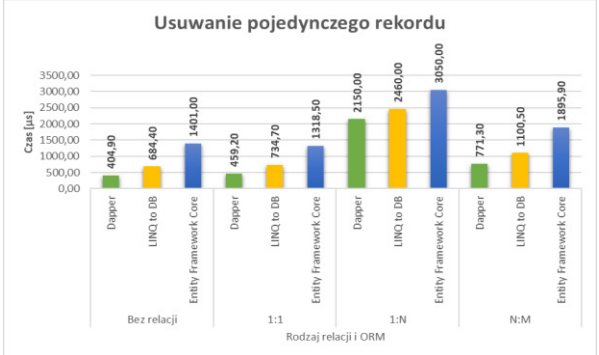
Rysunek 4: Średni czas wykonywania operacji wstawiania 1000 rekordów.



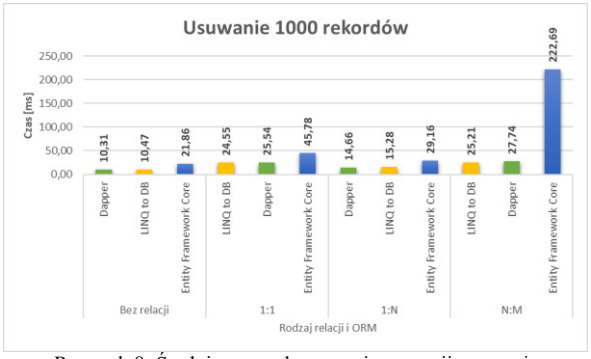
Rysunek 5: Średni czas wykonywania operacji aktualizacji pojedynczego rekordu.



Rysunek 6: Średni czas wykonywania operacji aktualizacji 1000 rekordów.



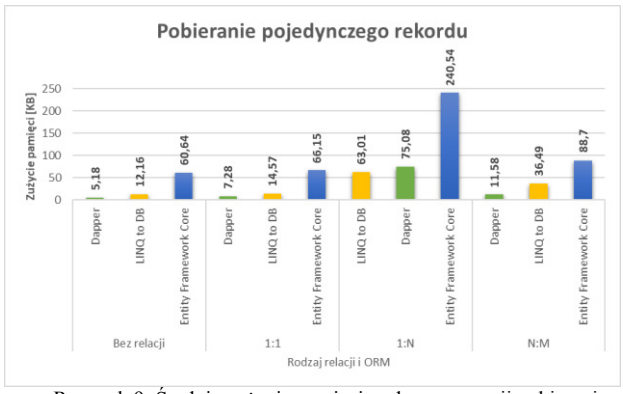
Rysunek 7: Średni czas wykonywania operacji usuwania pojedynczego rekordu.



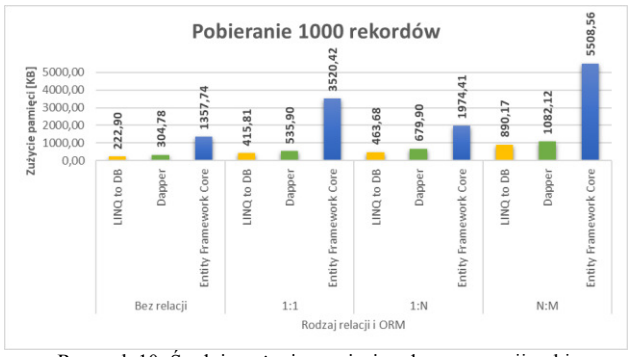
Rysunek 8: Średni czas wykonywania operacji usuwania 1000 rekordów.

6.2. Średnie zużycie pamięci

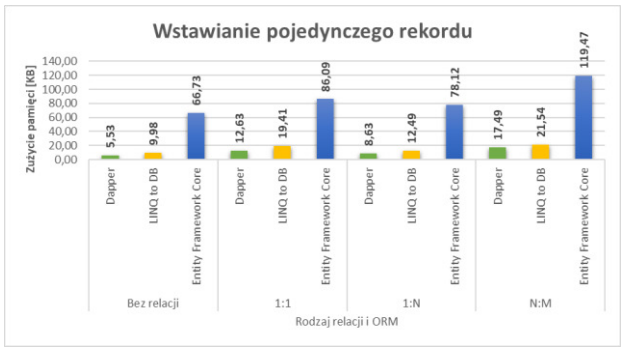
Średnie zużycie pamięci konkretnych operacji przedstawiono na Rysunkach 9-16.



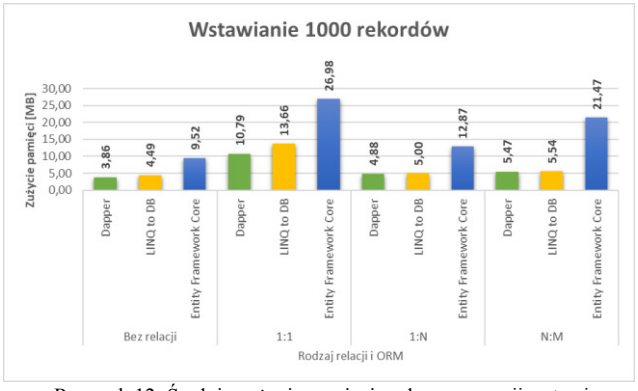
Rysunek 9: Średnie zużycie pamięci podczas operacji pobierania pojedynczego rekordu.



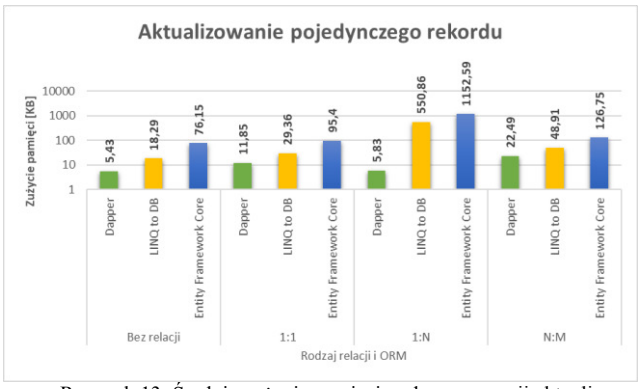
Rysunek 10: Średnie zużycie pamięci podczas operacji pobierania 1000 rekordów.



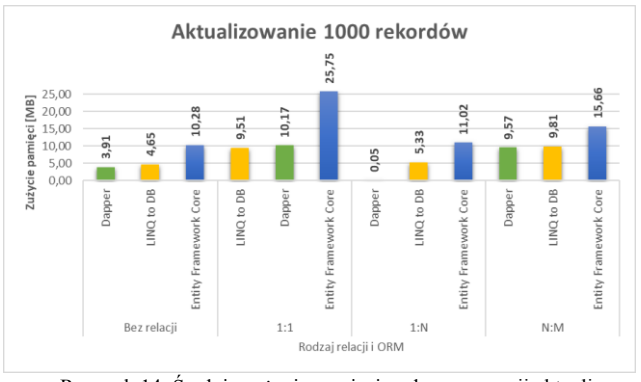
Rysunek 11: Średnie zużycie pamięci podczas operacji pobierania pojedynczego rekordu.



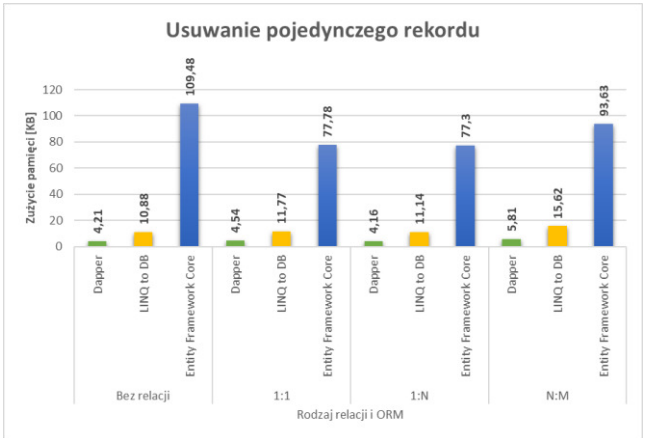
Rysunek 12: Średnie zużycie pamięci podczas operacji wstawiania 1000 rekordów.



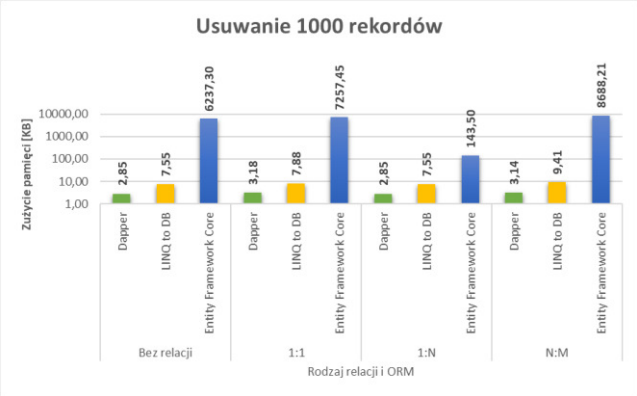
Rysunek 13: Średnie zużycie pamięci podczas operacji aktualizowania pojedynczego rekordu.



Rysunek 14: Średnie zużycie pamięci podczas operacji aktualizowania 1000 rekordów.



Rysunek 15: Średnie zużycie pamięci podczas operacji usuwania pojedynczego rekordu.



Rysunek 16: Średnie zużycie pamięci podczas operacji usuwania 1000 rekordów.

7. Podsumowanie

Przeprowadzone badania oraz analiza literatury pozwoliły na udowodnienie słuszności hipotezy „System ORM Dapper jest bardziej wydajny w operacjach DML niż Entity Framework Core i LINQ to DB". Dapper okazał się w większości przypadków testowych nie tylko wydajniejszy, ale także o wiele mniej zasobożerny niż pozostałe badane systemy.

Przeprowadzone testy wydajnościowe wykazały, że każdy z badanych systemów ORM ma swoje mocne i słabe strony w zależności od rodzaju operacji bazodanowych oraz typu relacji tabel. Entity Framework Core osiągał szczególnie dobre rezultaty w operacjach wstawiania wielu rekordów jednocześnie. Jednakże w przypadku pozostałych operacji cechował się on o wiele niższą wydajnością w porównaniu do innych badanych systemów. Dapper okazał się być najbardziej wydajny dla większości typów operacji. Szczególnie dobrze radził sobie z operacjami pobierania i usuwania rekordów. LINQ to DB wykazywał się stabilną wydajnością we wszystkich rodzajach operacji. Nie osiągał on najlepszych rezultatów w większości testów, lecz jego wydajność była zawsze na akceptowalnym poziomie.

Jeśli chodzi o zużycie pamięci podczas wykonywania operacji bazodanowych przez dany system ORM, to najmniejszym zużyciem w większości przypadków cechował się system Dapper a największym Entity Framework Core, zwłaszcza przy operacjach wymagających przetwarzania dużej ilości rekordów. LINQ to DB zużywał szczególnie mało pamięci podczas operacji pobierania wielu rekordów. W pozostałych przypadkach alokacja pamięci była większa niż przy użyciu systemu Dapper, lecz była na akceptowalnym poziomie.

Analiza pozycji literaturowych również pozwoliła zauważyć, że najwydajniejszym systemem ORM dla platformy .NET jest Dapper. Jednak w wielu pracach brakowało podziału na typy relacji oraz testy na pojedyncze i masowe. Mniejszy nacisk położono również na aspekt zużycia pamięci przez dany system ORM. Nie znaleziono również pozycji, w których badano system LINQ to DB, co spowodowane jest tym, że jest to stosunkowo nowy system.

Analiza porównawcza systemów ORM dla platformy .NET była skomplikowanym zadaniem, ale pozwoliła na lepsze zrozumienie, jak różne narzędzia radzą sobie w różnych sytuacjach. Wybór odpowiedniego systemu ORM jest kluczowy dla wydajności i skuteczności aplikacji, dlatego warto inwestować czas w dokładną analizę i testy. Przeprowadzone badania stanowią solidną podstawę do podjęcia decyzji w wyborze odpowiedniego systemu ORM i mogą być przydatne dla programistów pracujących w środowisku .NET.

Literatura

[1] Object-relational mapping, https://en.wikipedia.org/wiki/Object-relational_mapping, [18.11.2023].

[2] Entity Framework Core, https://learn.microsoft.com/en-us/ef/core, [18.11.2023].

[3] A. Ullah, M. Usman, M. Abrar, F. Ullah, N. Shah, M. F. Nadeem, Systematic performance and Security evaluation of .NET models for accessing database, VFAST Transactions on Software Engineering 9(4) (2021) 18-24.

[4] A Güvercin, B. Avenoglu, Performance Analysis of Object-Relational Mapping (ORM) Tools in .NET 6 Environment, Bilişim Teknolojileri Dergisi 15(4) (2022) 453-465.

[5] D. Zmaranda, L. Pop-Fele, C. Gyorödi, R. Gyorödi, G. Pecherle, Performance comparison of crud methods using .NET object relational mappers: A case study, International Journal of Advanced Computer Science and Applications 11(1) (2020) 55-65.

[6] W. Wiphusitphunpol, T. Lertrusdachakul, Fetch Performance Comparison of Object Relational Mapper in .NET Platform, In 2017 14th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON) IEEE Xplore (2017) 423-426.

[7] I. Basheleishvili, A. Bardavelidze, K. Bardavelidze, Study and Analysis of the .Net Platform-based Technologies for Working with the Databases, In Proceedings of the 33rd International Conference on Information Technologies InfoTech (2019) 1-8.

[8] Dapper, https://github.com/DapperLib/Dapper, [18.11.2023].

[9] LINQ to DB, https://linq2db.github.io, [18.11.2023].

[10] Bogus, https://github.com/bchavez/Bogus, [22.11.2023].

[11] BenchmarkDotNet, https://github.com/dotnet/BenchmarkDotNet, [22.11.2023].