Politechnika Śląska

Wydział Automatyki, Elektroniki i Informatyki

Kierunek Informatyka

Mateusz Stanik

###### **Porównanie wydajności baz danych napisanych w technologii ORM**

##### praca dyplomowa magisterska

Promotor: Aleksandra Werner

Konsultant:

Gliwice, luty 2018

Załącznik Nr 2 do Zarz. Nr 97/08/09

**Oświadczenie**

Wyrażam zgodę/nie wyrażam\* zgody na udostępnienie mojej pracy dyplomowej/rozprawy doktorskiej\*

…………….………., dnia …………………………

……………………………………..……………….……

*(podpis)*

……………………………………………………………

*(poświadczenie wiarygodności podpisu przez Dziekanat)*

*\* właściwe podkreślić*

**Oświadczenie promotora**

Oświadczam, że praca „Porównanie wydajności baz danych napisanych w technologii ORM” spełnia wymagania formalne pracy dyplomowej magisterskiej.

|  |  |
| --- | --- |
| Gliwice, dnia ……………………… | ………………..……………….……  *(podpis)* |

Spis treści

[1. Wstęp 9](#_Toc514873947)

[2. Technologia ORM 10](#_Toc514873948)

[2.1. Technologia Entity Framework 11](#_Toc514873949)

[2.1.1. Sposoby/możliwości tworzenia bazy 11](#_Toc514873950)

[2.1.2. Klasa kontekstowa Entity Framework 13](#_Toc514873951)

[3. Aplikacja 15](#_Toc514873952)

[3.1. Wymagania funkcjonalne 15](#_Toc514873953)

[3.2. Wymagania niefunkcjonalne 17](#_Toc514873954)

[3.3. Opis bazy danych aplikacji 18](#_Toc514873955)

[3.4. Testy jednostkowe aplikacji 20](#_Toc514873956)

[4. Część badawcza 22](#_Toc514873957)

[4.1. Metodyka badań/Plan eksperymentów/testów 22](#_Toc514873958)

[4.2. Konfiguracja środowiska testowego/Środowisko testowe 24](#_Toc514873959)

[4.3. Zbiory danych 26](#_Toc514873960)

[~~5.~~ ~~Testy~~ 26](#_Toc514873961)

[5.1. Uzyskane rezultaty/Wyniki 26](#_Toc514873962)

[5.2. Wnioski 26](#_Toc514873963)

[6. Podsumowanie 28](#_Toc514873964)

[Bibliografia i](#_Toc514873965)

[Spis skrótów i symboli ii](#_Toc514873966)

[Zawartość dołączonej płyty iii](#_Toc514873967)

[Spis rysunków iv](#_Toc514873968)

[Spis tabel v](#_Toc514873969)

# Wstęp

Celem pracy badawczej jest zaprezentowanie możliwości tworzenia bazy danych przy użyciu technologii ORM z wykorzystaniem narzędzia Entity Framework. W kolejnej części pracy zaprezentowano przebieg badań, których celem było porównanie wydajności zapytań bazodanowych napisanych w narzędziu Entity Framework, w odróżnieniu do powszechnych zapytań w strukturalnym języku zapytań bazodanowych SQL. W celu przeprowadzenie zaplanowanych badań została wcześniej przygotowana przykładowa aplikacja internetowa umożliwiająca wystawianie, a następnie sprzedaż sprzętu żeglarskiego. Dla aplikacji stworzono relacyjną bazę danych na której kolejno zostały przeprowadzane zapytania badające wydajność bazy danych opartej o wyżej wspomnianej technologii.

Architekci tworzący systemy informatyczne ciągle stoją przed ciężkim wyborem jakim jest technologia wykorzystywana do tworzenia bazy danych, a następnie do jej zarządzania. Niestety nie jest jasno sprecyzowane, czy środowisko oparte o technologię ORM jest wydajniejsze i łatwiejsze w zarządzaniu od tradycyjnego podejścia w SQL. W środowisku deweloperskim jest odwieczna walka między zwolennikami obydwóch podejść. Niniejsza praca ma na celu zweryfikowania w czym dany rodzaj bazy jest lepszy od drugiego.

# Technologia ORM

Celem pracy było zweryfikowanie, czy baza danych oparta o technologię ORM jest wydajniejsza od standardowej bazy danych napisanej w języku SQL. Jest to odwieczna bolączką deweloperów piszących systemy informatyczne, które mają zostać rozwiane dzięki niniejszej pracy badawczej. Od momentu kiedy pojawiła się technologia ORM programiści zastanawiają się, czy aby na pewno zastosować ją w swoim projekcie. Istnieje przypuszczenie, że owa technologia tworzy chaos w kodzie przez co staje się „*nieczysty*”, a ponadto zapytania wykonywane do bazy danych przetwarzane są kilkukrotnie dłużej od zwykłych zapytań SQL.

Praca zawierać będzie porównania dotyczące zarówno szybkości zapytań wykonanych w obu typach bazy jak i również łatwość ich zapisania przez developera wytwarzającego kod. Dla celów badawczych utworzono przykładową aplikację korzystającą z relacyjnej bazy danych na której omawiane badania zostały przeprowadzone. **TODO…**

## Technologia Entity Framework

Jednym ze sposobów tworzenia baz danych jest język SQL dzięki któremu możliwe jest zakładania tabel z polami różnego rodzaju, dodatkowo można dodawać między nimi relację. Możliwe jest również w opisywany języku tworzyć widoki składające się z wcześniej utworzonych tabel. Tworzenie baz danych może odbywać się również w inny sposób – za pomocą technologii mapowania obiektowo relacyjnego ORM  (ang. Object-Relational Mapping) . Jest to sposób w którym tabele oraz ich relacje są opisywane przy pomocy obiektowego stylu programowania. Jednym z frameworków tworzących bazy w taki stylu jest Entity Framework, który został wykorzystany w trakcie badań. Utworzona baza danych dla aplikacji została wykonana w oparciu o jedno z trzech możliwych podejść wytwarzania baz w omawianej technologii. W podrozdziale 2.1.1 przedstawiono dokładniej każdy z nich.

### Sposoby/możliwości tworzenia bazy

Entity Framework daje możliwość tworzenia bazy dany w trzech różnych podejściach, które zostały opisane poniżej.

* **Database First:**

Jest najbardziej spopularyzowany podejściem ze względu na jego powszechność i standardowość. W tym rodzaju użytkownicy wychodzi od utworzenia bazy w typowym stylu, a mianowicie pisząc całą bazę w języku SQL. Następnie po tak utworzonym projekcie, użytkownik przystępuje do generacji kodu obiektowego. Proces tworzenia bazy od strony developera odbywa się dzięki kreatorowi dostarczonemu przez Visual Studio, w którym użytkownik musi wybrać wcześniej utworzoną bazę w SQL, po czym generator na jej podstawie tworzy obiektowo odwzorowanie bazy wraz z relacjami. Dodatkowo oprócz plików związanych z tabelami, znajdą się tu również pliki zawierający kontekst do tabel dzięki którym developer ma możliwość odwoływania się w kodzie do każdej z nich.

* **Code First**

Podejście to zostało między innymi wykorzystane podczas tworzenia bazy danych dla aplikacji na której wykonywano badania. Polega ono na tworzeniu bazy wychodząc w pierwszej kolejności od obiektowej części. Tabele tworzone są w oddzielnych plikach *.cs*, natomiast relacje są dodawane w dwóch możliwych podejściach. Pierwszy z nich to racji przy wykorzystaniu adnotacji oraz metod abstrakcyjnych określających połączenie z kolejną tabelą. Adnotacje wykorzystuje się w celu wskazania między innymi klucza obcego („*ForeignKey*”), czy klucza głównego („*PrimaryKey*”). Dodatkowo przy użyciu adnotacji nadaję się właściwości dla pól, tj. ilość dopuszczalnych znaków, wymagalność, typ. Drugim możliwym podejściem tworzenia relacji jest wykorzystanie tzw. *Fluent Api.* Polega ono na wskazaniu relacji między klasami tabel w oddzielnym pliku. W tym podejściu nie stosuje się adnotacji (oczywiście za wyjątkiem właściwości pól). Po tak utworzonych tabelach i relacjach, developer w celu utworzenia lub zaktualizowania bazy na serwerze musi wykonać migracje w dodatku *Visual Studio* o nazwie *Package Manager Console.* Migracje dodaje się przy pomocy komendy *Add-migration name-„<nazwa migracji>”*, z kolei samą aktualizacji wykonuję się podając *Update-database.* Jeżeli chcemy wygenerować skrypt z aktualizacji bazy musimy po komendzie dodać słowo *– verbose.*

* **Model First**

W tym rodzaju, wykorzystujemy do tworzenia bazy specjalny generator dostarczony do Visual Studio. Jest to najprawdopodobniej najłatwiejsza wersja ze wszystkich trzech podejść, ze względu na prostotę generacji tabel i ich relacji. W generatorze użytkownik dodaję nową tabele klikając prawym przyciskiem myszy na diagramie bazy. Następnie we właściwościach wcześniej utworzonej encji dodawane są kolejne pola o sprecyzowanych typach. Po utworzeniu przynajmniej dwóch tabel możliwe jest dodanie relacji miedzy nimi. Wszystkie zmiany wprowadzone w tabelach (pola, czy też relacje), prezentowane są na diagramie bazy danych. W celu zapisania projektu lub jego utworzeniu na serwerze, użytkownik musi wygenerować odpowiedni skrypt SQL. Taka generacja skryptu jest również łatwa do przeprowadzenia ponieważ wykonują ja za nas generator, poczym taki skrypt jest uruchamiany na serwerze bazodanowym tworząc na nim wcześniej utworzoną bazę.

### Klasa kontekstowa Entity Framework

Dostęp do bazy danych w technologii ORM tworzony jest dzięki obiektowości. Aby stworzyć taki obiekt musimy w pierwszej kolejności utworzyć klasę tzw. Kontekstową, która dziedziczy po klasie *DbContext* dostarczonej przez bibliotekę Entity Framework.

W ciele tej klasy dodajemy jako pola, kolekcje tabel utworzonych wcześniej jako model *CodeFirst.* Kolekcja reprezentująca jest typu *DbSet*, a przykładowa składnia wygląda następująco: public DbSet<Advert> adverts { get; set; }. W przypadku jeśli wybraliśmy dodawanie relacji w bazie przy użyciu FluentApi, a nie poprzez adnotacje i metody visrtualne, wtedy do klasy dodajemy metodę - protected override void OnModelCreating(DbModelBuilder modelBuilder), gdzie rozpisujemy konkretne relacje.

Aby utworzony kontekst wiedział z jakiej bazy powinien korzystać, należy odwołać się w konstruktorze do klasy bazowej i podać we właściwości *name* nazwę bazy, która znajduje się w sekcji *ConnectionString* w pliku konfiguracyjnym. Składnia takiego konstruktora wygląda następująco:

public EFDbContext() : base("name=BoatsAdverts"), natomiast sekcję konfiguracyjną zapisuje się w taki sposób:

<connectionStrings>

<add name = "UnitOfWork.Properties.Settings.BoatsAdvertsConnectionString" connectionString="Data Source = (LocalDb)\v11.0; AttachDbFilename=&quot; C:\Users\Mateusz\Desktop\Odchudzanie Loggera\Durandal451v2\App\_Data\BoatsAdverts.dbo&quot;; Initial Catalog = BoatsAd; Integrated Security=True " providerName = "System.Data.SqlClient" />

</connectionStrings>

Konstruktor jest również wykorzystywany do ustawiania inicjalizatora. Dzięki niemu wskazuje mu co ma się zadziać po stronie bazy przy każdej kolejnej kompilacji kodu. Entity Framework udostępnia cztery takie inicjalizatory. Pierwszy z nich *–* CreateDatabaseIfNotExists, umożliwia utworzenie nowej bazy danych jeżeli wieszcze taka o konkretnej nazwie nie istniej. Druga – DropCreateDatabaseIfModelChanges, za każdym razem jeżeli model bazy jest zmieniony, EF usuwa całkowicie dane i bazę, a następnie od nowa ją zakłada. Trzeci inicjalizator – DropCreateDatabaseAlways, za każdym razem bez względu na zmianę lub nie modelu, uzuwa baze i tworzy ją na nowo. Ostatni inicjalizator - Custom DB Initializer: daje użytkownikowi całkowitą dowolność, ponieważ możemy w tym przypadku dodać swój własny inicjalizator jeśli żaden z powyższych nie spełnia wymaganych oczekiwań.

W miejscu aplikacji gdzie chcemy odwołać się do danej tabeli musimy wcześniej utworzyć obiekt takiej klasy kontekstowej, a następnie dla niego odwoływać się do poszczególnych kolekcji (tabel), przy użyciu technologii Linq będącej składnikiem języka C#. Warto jednak wykorzystać w odwoływaniu się do kontekstu przy pomocy wzorca projektowego *Dependency Injection*, dzięki któremu nie będzie konieczne za każdym razem tworzenia nowego obiektu.

# Aplikacja

Eksperymenty wykonywane w trakcie badań zostały przeprowadzone na relacyjnej bazie danych wykorzystywanych przez aplikacje internetową. Aplikacja ta ma pełnić rolę żeglarskiej aukcji internetowych. Jej użytkownicy mają możliwość wystawiania jachtów żaglowych, motorowych, silników zaburtowych oraz stacjonarnych, przyczep podłodziowych i wszelkiego rodzaju osprzętu żeglarskiego. W celu wystawienia takiego ogłoszenia, osoba musi być zalogowanych użytkownikiem serwisu. Po wypełnieniu formularza oraz dodaniu zdjęć przedmiotu użytkownik wybiera okres ważności ogłoszenia oraz dodatkowe funkcjonalności takie jak wyróżnienie ogłoszenia, które jest dodatkowo płatne. Dzięki podłączeniu się aplikacji do zewnętrznego systemu płatności możliwe jest automatyczne przekserowanie użytkownika do wykonania przelewu w swoim banku. **TODO…**

## Wymagania funkcjonalne

* Aplikacja umożliwia wystawianie do sprzedaży przedmiotów żeglarskich.
* System powinien posiadać graficzny interfejs użytkownika składający się dwóch widoków: kliencki, oraz administratorski.
* Widok administratorski ma możliwość bezpośredniego dodawania użytkowników indywidualnych jak i firmowych bez weryfikacji mailowej.
* Każdy z użytkowników ma możliwość przeglądania ofert wystawionych na stronie.
* Zarejestrowany użytkownik ma możliwość wystawienia ogłoszenia związanego z przedmiotem lub ogłoszenia związanego z usługą.
* Aplikacja powinna prezentować dane kontaktowe do administracji serwisu w razie ewentualnych niejasności używania systemu.
* Każdy użytkownik ma mieć wgląd do instrukcji dodawania przedmiotu na stronie.
* Logowanie może odbywać się poprzez wcześniej utworzone konto w aplikacji bądź zalogowanie się przy użyciu zewnętrznego konta (np. konto Google).
* Kreator ogłoszenia powinien umożliwiać dodawanie zdjęć wystawianego przedmiotu.
* Użytkownik w ramach jednej rejestracji zgłoszenia może wystawić tylko jeden przedmiot.

## Wymagania niefunkcjonalne

* Użytkownik może dodać ogłoszenie po wcześniejszym zarejestrowaniu, lub zalogowaniu do systemu.
* Rejestracja użytkownika jest możliwa po wprowadzeniu wymagalnych pół i uwierzytelnieniu mailowym.
* Pola wymagalne przy rejestracji to login będący adresem mailowym, oraz hasło powtórzone dwukrotnie zawierające małe i duże litery, znaki oraz cyfry.
* Zalogowany użytkownik ma możliwość dodania przedmiotu dopiero po poprawnym wypełnieniu formularza, którego pola są odpowiednio walidowane.
* Wgląd do produktów serwisu internetowego ma każdy użytkownik niezależnie czy jest on zalogowany, czy też nie.
* Zdjęcia dodawane w kreatorze ogłoszenia, muszą być walidowane pod względem ich ilości, rozmiaru plików oraz rozszerzenia plików. Walidacja musi odbywać się po obu częściach aplikacji tj. klienckiej oraz serwerowej.
* Modele dostarczane do kontrolerów na serwer muszą być odpowiednio walidowane w taki sam sposób ja po stronie klienckiej.

## Opis bazy danych aplikacji

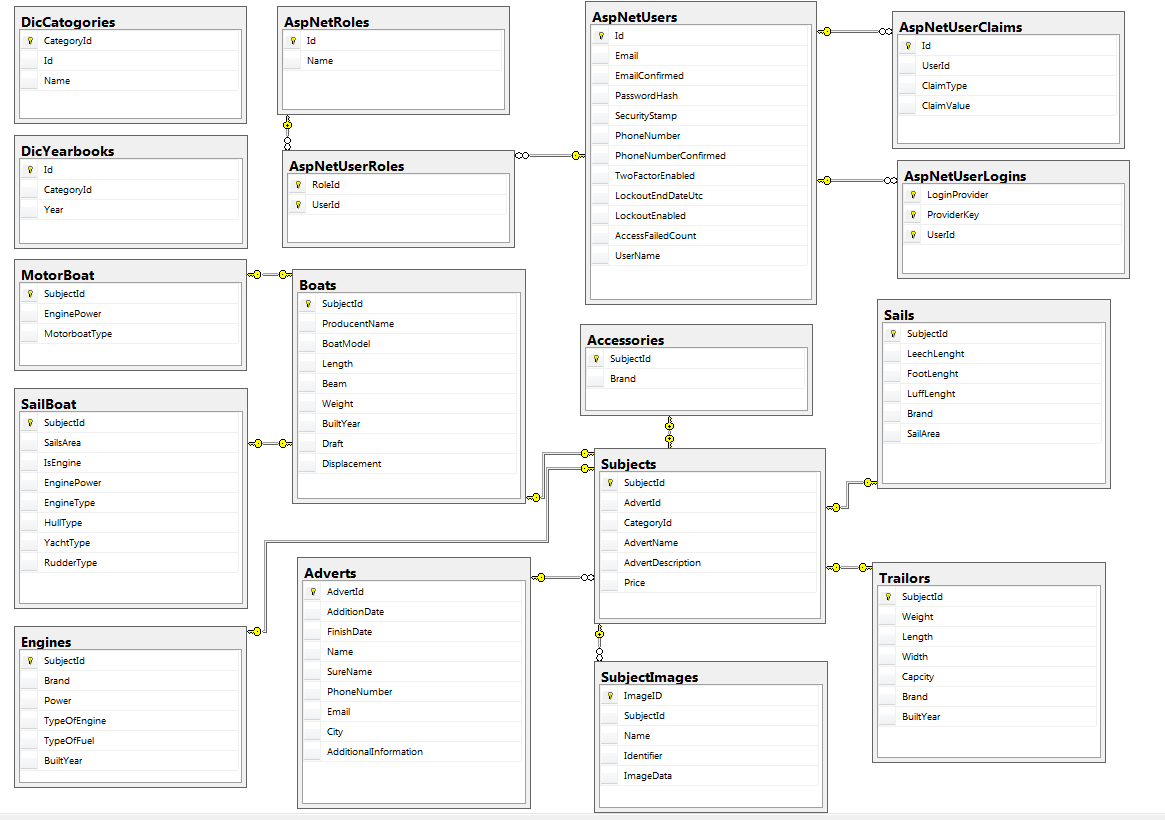
Baza danych na której przeprowadzano badania została zaprojektowana w technologii ORM przy użyciu biblioteki Entity Framework. Przy projektowaniu wykorzystano podejście Code-First, którego działanie dokładniej zostało opisane w rozdziale *2.2.1 Rodzaje podejść tworzenia bazy.*

W skład bazy wchodzą dwie kluczowe podsekcje wykorzystywane w systemie. Pierwsza z nich odpowiada za logowanie, natomiast kolejna jest wykorzystywana do kategoryzowania produktów wraz z ich słownikami. W celu umożliwienia logowania użytkowników do systemu, baza danych zaopatrzona jest w szereg tabel odpowiedzialnych za tą czynność. Zawierają one podstawowe informacje na temat użytkownika, którym może być klient jak i administrator aplikacji. W skład danych takiej osoby wchodzą między innymi dane kontaktowe (mail, numer telefonu), informacje potrzebne do logowania (nazwa użytkownika, hasło), informacje statystyczne dotyczące prób logowania (licznik ilości prób logowania, data wylogowania). Ze względu na to, że w aplikacji konieczne jest rozróżnienie użytkowników w celu poszerzenia dla nich funkcjonalności wprowadzono dodatkowo słownik ról. Na obecny stan aplikacja posiada zaimplementowane dwie role - Administrators, RegisteredUsers, które nadawane są dla następujących przypadków.

Prawa administratorskie nadawane są dla użytkowników zarządzających całym systemem oraz mogącym dodawać i sterować wszystkimi przedmiotami znajdującymi się w systemie. Takie uprawnienia może wprowadzać tylko i wyłącznie programista bądź developer wspierający aplikacje poprzez odpowiedni wpis do bazy danych.

Kolejną rolę – RegisteredUsers, nadaję automatycznie system dla nowo zarejestrowanych użytkowników. Mają oni dostęp do podstawowych danych informacyjnych znajdujących się na stronie, oraz umożliwia się im dodawanie i zażądanie tylko swoimi przedmiotami.

Druga sekcja, która wchodzi w skład relacyjnej bazy danych przeznaczona jest do zarządzania wstawianymi przedmiotami w aplikacji. Główną tabelą opisującą przedmiot jest tabela *Advert* zawierająca podstawowe pola na temat ogłoszenia takie jak data jego dodania, przewidziana data jego zakończenia, imię i nazwisko właściciela, telefon kontaktowy, mail, miasto oraz informacje dodatkowe. Tak stworzone ogłoszenie może posiadać wiele przedmiotów, które opisuje tabela *Subjects*. Pola tu zawarte opisują dodatkowe informacje na temat ogłoszenia oraz przedmiotu tj. nazwa ogłoszenia, opis przedmiotu, cena. Dla przedmiotu użytkownik ma możliwość wstawienia wielu zdjęć prezentujących ofertę. Reprezentuję to tabela *SubjectImages*. Dla nazwy zdjęcia przeznaczone jest pole *Name*, do przechowywania meta danych pliku – *ImageData*, natomiast identyfikator zdjęcia – *Identifier.*



## Testy jednostkowe aplikacji

# Część badawcza

Praca badawcza ma za zadanie wykazać jak wydajne są środowiska zarządzające bazą danych, oraz sprecyzować dlaczego jedno jest lepsze od drugiego. Jako środowiska badawcza w pracy wykorzystano – SQL Management Studio, używane do zapytań w języku SQL, oraz biblioteka ORM Entity Framework w którym zapytania wykonujemy przy użyciu technologii obiektowej.

## Metodyka badań/Plan eksperymentów/testów

W celu wykazania, które z dwóch badanych środowisk jest wydajniejsze, należało przeprowadzić szereg zapytań bazodanowych. Zapytania te polegały na przetworzeniu konkretnej ilości rekordów, oraz dostępu do nich na określonym zagłębieniu rekordu w relacyjnej bazie danych. Tak przygotowane zapytania uruchamiano w trzech możliwych podejściach (w SQL management studio, ORM – przy pomocy Entity Framework oraz czystym języku SQL uruchomionym w aplikacji .NET), poczym weryfikowano które z nich jest wydajniejsze.

Wyliczanie czasu potrzebnego na otrzymanie odpowiedzi z serwera w przypadku SQL Management Studio otrzymywano za pomocą rozszerzenia dla języka zapytań nazywającego się T-SQL (Transact - SQL). Dodatek ten umożliwia tworzenie zmiennych w zapytaniach, dodawania pętli, czy też instrukcji warunkowych. Dla każdego z wykonywanych zapytań przygotowano szablon w opisywanym dodatku, który zaprezentowano poniżej.

Declare @t1 DATETIME;

Declare @t2 DATETIME;

Set @t1 = GETDATE();

ZapytanieSQL();

Set @t2 = GETDATE();

Select DATEDIFF(MILLISECOND, @t1, @t2) AS elapsed\_ms

Do wyliczenia czasu zapytania ZapytanieSQL(); potrzebne jest zadeklarowanie dwóch zmiennych czasowych *@t1*, *@t2.* Następnie przypisujemy pierwszej zmiennej aktualnej godziny używając do tego funkcji *GETDATE()*, poczym możliwe jest wywołanie badanego zapytania. Po jego wywołaniu, dla drugiej zmiennej przypisujemy kolejny raz aktualną godzinę. Na koniec wyliczamy różnicę między obydwiema zamiennymi przy użyciu funkcji *DATEDIFF()*. Wynik prezentujemy przy pomocy słowa kluczowego *Select* pod nazwą *elapsed\_ms* w milisekundach.

W przypadku zapytań ORM oraz SQL w aplikacji .NET czas potrzebny na odpowiedź wyliczano przy pomocy języka C# i jego wbudowanych bibliotek. W tym celu, korzystno z klasy wyliczeniowej Stopwatch, dla której nowo utworzony obiekt zbierał dane na temat czasu wykonania danej operacji. Klasa ta pochodzi z biblioteki System.Diagnostics. Aby obiekt mógł rozpocząć wyliczanie czasowe należy wywołać metodę StartNew(). Po takim użyciu można rozpocząć wykonywanie badanych operacji bazodanowych. Następnie po zakończeniu tej sekcji, na wcześniej utworzonym obiekcie klasy Stopwatch należy zakończyć obliczanie poprzez uruchomienie metody Stop(). W celu pobrania czasu potrzebnego na wykonanie powyższej operacji, należy odwołać się do pola obiektu Elapsed.TotalMilliseconds. Przykładowy kod obliczający czas wykonania testowej metody zamieszczono poniżej.

Stopwatch stopWatch = Stopwatch.StartNew();

TestMethod();

stopWatch.Stop();

Technologia ORM wykonująca zapytanie utworzone w języku obiektowym musi przejść przez własny translator. Zadaniem takiego mechanizmu jest, aby zamienić owy język na czysto SQL-owe zapytanie bazodanowe.

Poniżej zaprezentowano składnie wypisującą w konsoli *Debug* treść przetworzonego zapytania.

this.Database.Log = s => System.Diagnostics.Debug.WriteLine(s);

## Konfiguracja środowiska testowego/Środowisko testowe

W celu wykonania badań, należało wcześniej zaprojektować obiekt na którym zostałyby później przeprowadzone. Obiektem badań tej pracy jest aplikacja internetowa korzystająca z relacyjnej bazy danych. Projekt został w głównie mierze wykonany w oparciu o technologie wytwarzane i wspierane przez firmę Microsoft oraz źródła ogólnie dostępne (ang. open source).

Ze względów na łatwość i szybkość pobierania dodatkowych bibliotek do aplikacji, zdecydowano o wykorzystywaniu   Visual Studio w wersji Enterprise 2015, jako [zintegrowanego środowiska programistyczne](https://pl.wikipedia.org/wiki/Zintegrowane_%C5%9Brodowisko_programistyczne)go. Z powodu, iż badania przeprowadzano na aplikacji internetowej, wykorzystano w tym celu platformę ASP.NET, która umożliwia w Visual Studio tworzenie zaawansowanych serwisów internetowych. Projekty w tej technologii są tworzone w oparciu o obiektowy styl programowania dodatkowo daje możliwość odseparowania części serwerowej od części prezentacyjnej (klienckiej), przez co zwiększona jest przejrzystość kodu, oraz jego wydajność. Ponadto możliwe jest dzięki Visual Studio w krótkim czasie wykonać publikację aktualnej wersji projektu na serwer gdzie znajduję się jego hosting. Jako bazę danych wykorzystano MS SQL Server, z powodu dobrej komunikacji z Visual Studio oraz dobrego wsparcia technicznego ze strony wydawcy. Do zarządzania wszystkimi komponentami dostarczonymi z serwerem wykorzystano zintegrowane środowisko SQL Management Studio. Dzięki niemu możliwe jest podejrzenie tabel, przejrzenie danych oraz generowanie diagramu tabel wraz z relacjami zawartymi miedzy nimi. Baza danych została wygenerowana wraz z relacjami dzięki technologii opartej o Mapowanie obiektowo-relacyjne (ORM). Do jej generacji wykorzystano narzędzie Entity Framework. Technologia ta zostanie dokładniej opisana w jednym z kolejnych rozdziałów(**Rozdział z opisem EF**). Dla oprogramowania części klienckiej użyto Framework javascript – knockout.js wraz z dodatkiem Durandal.js który wykorzystuje wzorzec projektowy MVC, dzięki czemu w łatwy sposób możliwe jest połączenie go z wcześniej opisywaną platformą ASP.NET. Kolejnym argumentem determinującym wybór jest stabilność platformy Duranda.Js oraz frameworka Knockout.js. Dla usprawnienia interakcji użytkownika z aplikacją oraz rozszerzenia funkcjonalności powyższego frameworka wykorzystano bibliotekę jQuery dla języka JavaScript. Dzięki niemu możliwe są wszelkiego rodzaju akcje między innymi na każdym z elementów DOM (tj. Obiektowy model dokumentu, ang. Document Object Model). Komunikacja między częścią kliencką, a serwerową odbywa się dzięki żądaniom AJAX do WebApi będącego częścią platformy ASP.NET. Do samej części prezentacyjnej zastosowano gotową bibliotekę styli CSS – Bootstrap. W delikatnej mierze zostały zmodyfikowane i przystosowane do wymagań wyglądu strony.

Spis technologii wykorzystanych w projekcie oraz badań:

* Visual Studio Professional 2015
* SQL Server wraz z SQL Server Management Studio
* Entity Framework
* Knockout.js
* Durandal
* Jquery
* Ajax
* Bootstrap

## Zbiory danych

# ~~Testy~~

## Uzyskane rezultaty/Wyniki

## Wnioski

|  |
| --- |
|  |

# Podsumowanie

# Bibliografia

1. [Freeman](http://helion.pl/autorzy/adam-freeman,adamfreeman.htm) A., *ASP.NET MVC 5. Zaawansowane programowanie*, Helion, Gliwice, 2015.
2. Gamma W., Helm R., Johnson R., Vlissides J., *Wzorce Projektowe Elementy oprogramowania obiektowego wielokrotnego użytku,* Helion, Gliwice, 2010.
3. Rockoff L., *Język SQL. Przyjazny podręcznik*, Helion, Gliwice, 2014.
4. Wiegers K., Beatty J., *Specyfikacja oprogramowania. Inżynieria Wymagań*, Helion, Gliwice, 2014.

# Spis skrótów i symboli

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |

# Zawartość dołączonej płyty

# Spis rysunków

# Spis tabel