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 3](#_Toc511647813)

[2. Analiza tematu 4](#_Toc511647814)

[2.1. Baza danych oparta o technologię ORM – Entity Framework 4](#_Toc511647815)

[2.1.1. Rodzaje podejść tworzenia bazy 5](#_Toc511647816)

[2.1.2. Klasa kontekstowa Entity Framework 7](#_Toc511647817)

[2.2. Wymagania funkcjonalne 9](#_Toc511647818)

[2.3. Wymagania niefunkcjonalne 10](#_Toc511647819)

[2.4. Wykorzystane oprogramowanie w trakcie badań 11](#_Toc511647820)

[3. Przedmiot pracy 15](#_Toc511647821)

[3.1. Relacyjna baza danych 15](#_Toc511647822)

[3.1.1. Diagram bazy 15](#_Toc511647823)

[3.2. Testy jednostkowe aplikacji 15](#_Toc511647824)

[4. Badania 17](#_Toc511647825)

[4.1. Metodyka badań 17](#_Toc511647826)

[4.2. Zbiory danych 17](#_Toc511647827)

[4.3. Wyniki 17](#_Toc511647828)

[5. Podsumowanie 19](#_Toc511647829)

[Bibliografia i](#_Toc511647830)

[Spis skrótów i symboli ii](#_Toc511647831)

[Zawartość dołączonej płyty iii](#_Toc511647832)

[Spis rysunków iv](#_Toc511647833)

[Spis tabel v](#_Toc511647834)

# Wstęp

W niniejszej pracy chciałbym zaprezentować możliwości tworzenia bazy danych przy użyciu technologii ORM z wykorzystaniem narzędzia Entity Framework. W kolejnej części pracy zaprezentuję 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.

# Analiza tematu

Celem pracy było zweryfikowanie, czy baza danych oparta o technologię ORM jest wydajniejsza od standardowej bazy danych napisanej w języku SQL. Porównania dotyczą zarówno szybkości zapytań wykonanych w obu typach bazy jak i również łatwością ich zapisania przez developera wytwarzającego kod. Dla celów badawczych utworzono przykładową aplikację korzystającą z relacyjnej bazy danych na której omawiane badania zostały przeprowadzone.

## Baza danych oparta o technologię ORM – 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.

### Rodzaje podejść 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.

## 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.

## Wykorzystane oprogramowanie w trakcie badań

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

# Przedmiot pracy

## Relacyjna baza danych

### Diagram bazy

## Testy jednostkowe aplikacji

# Badania

## Metodyka badań

## Zbiory danych

## Wyniki

|  |
| --- |
|  |

# 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