

Contents

Entity Framework

[Porównanie programów EF Core i EF6](#)

[Programy EF6 i EF Core w tej samej aplikacji](#)

[Przenoszenie z programu EF6 do programu EF Core](#)

[Sprawdzanie poprawności wymagań](#)

[Przenoszenie modelu opartego na EDMX](#)

[Przenoszenie modelu opartego na kodzie](#)

Entity Framework Core

[Co nowego](#)

[Plany rozwoju programu EF Core](#)

[EF Core 2.2 \(najnowsza stabilna wersja\)](#)

[EF Core 2.1](#)

[EF Core 2.0](#)

[EF Core 1.1](#)

[EF Core 1.0](#)

[Uaktualnianie z wcześniejszych wersji](#)

[Z wersji 1.0 RC1 do wersji RC2](#)

[Z wersji 1.0 RC2 do wersji RTM](#)

[Z wersji 1.x do wersji 2.0](#)

[Wprowadzenie](#)

[Instalowanie programu EF Core](#)

[.NET Core](#)

[Nowa baza danych](#)

[ASP.NET Core](#)

[Nowa baza danych](#)

[Istniejąca baza danych](#)

[□ Produkty EF Core i Razor Pages](#)

[Platforma uniwersalna systemu Windows \(UWP\)](#)

[Nowa baza danych](#)

.NET Framework

Nowa baza danych

Istniejąca baza danych

Podstawy

Parametry połączeń

Rejestrowanie

Elastyczność połączenia

Testowanie

Testowanie za pomocą SQLite

Testowanie za pomocą InMemory

Konfigurowanie typu DbContext

Tworzenie modelu

Uwzględnianie i wykluczanie typów

Uwzględnianie i wykluczanie właściwości

Klucze (podstawowe)

Generowane wartości

Właściwości wymagane/opcjonalne

Maksymalna długość

Tokeny współbieżności

Właściwości w tle

Relacje

Indeksy

Klucze alternatywne

Dziedziczenie

Pola zapasowe

Konwersje wartości

Wstępne wypełnianie danych

Konstruktorzy typów jednostek

Posiadane typy jednostek

Typy zapytań

Modele naprzemienne z tym samym typem DbContext

Dane przestrzenne (GIS)

Modelowanie relacyjnej bazy danych

Mapowanie tabeli

Mapowanie kolumn

Typy danych

Klucze podstawowe

Schemat domyślny

Kolumny obliczane

Sekwencje

Wartości domyślne

Indeksy

Ograniczenia klucza obcego

Klucze alternatywne (ograniczenia unikatowe)

Dziedziczenie (relacyjna baza danych)

Zarządzanie schematami bazy danych

Migracje

Środowiska zespołowe

Operacje niestandardowe

Korzystanie z osobnego projektu

Wielu dostawców

Niestandardowa tabela historii

Tworzenie i upuszczanie interfejsów API

Odtwarzanie (tworzenie szkieletów)

Wykonanie zapytania o dane

Zapytanie podstawowe

Ładowanie powiązanych danych

Klient a wyznaczenie wartości na serwerze

Śledzenie a brak śledzenia

Pierwotne zapytania SQL

Zapytania asynchroniczne

Jak działa zapytanie

Filtr zapytań globalnych

Tagi zapytań

Zapisywanie danych

[Zapisywanie podstawowe](#)

[Powiązane dane](#)

[Usuwanie kaskadowe](#)

[Konflikty współbieżności](#)

[Transakcje](#)

[Zapisywanie asynchroniczne](#)

[Odłączone jednostki](#)

[Jawne wartości dla wygenerowanych właściwości](#)

Obsługiwane implementacje platformy .NET

Dostawcy baz danych

[Microsoft SQL Server](#)

[Tabele zoptymalizowane pod kątem pamięci](#)

[SQLite](#)

[Ograniczenia SQLite](#)

[InMemory \(do testowania\)](#)

[Tworzenie dostawcy bazy danych](#)

[Zmiany wpływające na dostawcę](#)

Narzędzia i rozszerzenia

Dokumentacja wiersza poleceń

[Konsola menedżera pakietów \(Visual Studio\)](#)

[.NET Core CLI](#)

[Tworzenie typu DbContext w czasie projektowania](#)

[Usługi w czasie projektowania](#)

Dokumentacja interfejsów API platformy EF Core

Entity Framework 6

Co nowego

[Plan rozwoju](#)

[Poprzednie wydania](#)

[Uaktualnianie do wersji EF6](#)

[Wydania programu Visual Studio](#)

Wprowadzenie

Podstawowe założenia

Pobieranie platformy Entity Framework

Praca z klasą DbContext

Interpretacja relacji

Asynchroniczne wykonywanie zapytań i zapisywanie

Konfiguracja

Oparte na kodzie

Plik konfiguracji

Parametry połączeń

Rozpoznawanie zależności

Zarządzanie połączniami

Elastyczność połączenia

Logika ponawiania próby

Niepowodzenia zatwierdzania transakcji

Powiązanie danych

WinForms

WPF

Odłączone jednostki

Własnoręczne śledzenie jednostek

Przewodnik

Rejestrowanie i przejmowanie

Wydajność

Zagadnienia dotyczące wydajności (oficjalny dokument)

Korzystanie z technologii NGEN

Korzystanie ze wcześniej wygenerowanych widoków

Dostawcy

Model dostawcy EF6

Obsługa przestrzenna w przypadku dostawców

Korzystanie z serwerów proxy

Testowanie za pomocą platformy EF6

Korzystanie z pozorowania

Pisanie własnych symulacyjnych obiektów testowych

Możliwość testowania przy użyciu platformy EF4 (artykuł)

Tworzenie modelu

Korzystanie z metody Code First

Przepływy pracy

Z nową bazą danych

Z istniejącą bazą danych

Adnotacje danych

Obiekty DbSet

Typy danych

Wyliczenia

Przestrzenne

Konwencje

Konwencje wbudowane

Konwencje niestandardowe

Konwencje modelu

Płynna konfiguracja

Relacje

Typy i właściwości

Używanie w języku Visual Basic

Mapowanie procedury składowanej

Migracje

Automatyczne migracje

Praca z istniejącymi bazami danych

Dostosowywanie historii migracji

Korzystanie z pliku Migrate.exe

Migracje w środowiskach zespołów

Korzystanie z projektanta EF

Przepływy pracy

Model-First

Database-First

Typy danych

Typy złożone

[Wyliczenia](#)

[Przestrzenne](#)

[Mapowania dzielenia](#)

[Dzielenie jednostki](#)

[Dzielenie tabeli](#)

[Mapowania dziedziczenia](#)

[Tabela na hierarchię](#)

[Tabela na typ](#)

[Mapowanie procedur składowanych](#)

[Zapytanie](#)

[Aktualizacja](#)

[Mapowanie relacji](#)

[Wiele diagramów](#)

[Wybieranie wersji środowiska uruchomieniowego](#)

[Generowanie kodu](#)

[Starsza wersja obiektu ObjectContext](#)

[Zaawansowane](#)

[Format pliku EDMX](#)

[Definiowanie zapytania](#)

[Wiele zestawów wyników](#)

[Funkcje zwracające tabelę](#)

[Skróty klawiaturowe](#)

[Wykonanie zapytania o dane](#)

[Load, metoda](#)

[Dane lokalne](#)

[Zapytania ze śledzeniem i bez śledzenia](#)

[Korzystanie z pierwotnych zapytań SQL](#)

[Wykonywanie zapytań o dane pokrewne](#)

[Zapisywanie danych](#)

[Śledzenie zmian](#)

[Automatyczne wykrywanie zmian](#)

[Stan jednostki](#)

- [Wartości właściwości](#)
- [Obsługa konfliktów współbieżności](#)
- [Korzystanie z transakcji](#)
- [Walidacja danych](#)
- [Dodatkowe zasoby](#)
 - [Blogi](#)
 - [Analizy przypadków](#)
 - [Współtworzenie](#)
 - [Uzyskiwanie pomocy](#)
 - [Słownik](#)
 - [Przykładowa bazy danych szkoły](#)
 - [Narzędzia i rozszerzenia](#)
 - [Licencje](#)
- [EF5](#)
 - [Chiński \(uproszczony\)](#)
 - [Chiński \(tradycyjny\)](#)
 - [Niemiecki](#)
 - [Angielski](#)
 - [Hiszpański](#)
 - [Francuski](#)
 - [Włoski](#)
 - [Japoński](#)
 - [Koreański](#)
 - [Rosyjski](#)
- [EF6](#)
 - [Wersja wstępna](#)
 - [Chiński \(uproszczony\)](#)
 - [Chiński \(tradycyjny\)](#)
 - [Niemiecki](#)
 - [Angielski](#)
 - [Hiszpański](#)
 - [Francuski](#)

[Włoski](#)

[Japoński](#)

[Koreański](#)

[Rosyjski](#)

[!\[\]\(71ceb62b681518c82e95d615e7265d66_img.jpg\) Dokumentacja interfejsów API platformy EF6](#)

Dokumentacja programu Entity Framework

[Entity Framework](#)

Entity Framework jest maperem obiektowo-relacyjnym (O/RM), który pozwala programistom korzystającym ze środowiska .NET pracować z bazą danych, używając obiektów platformy .NET. Dzięki temu większa część kodu dostępu do danych, który programiści muszą zwykle tworzyć, nie jest już potrzebna.



[Entity Framework Core](#)

EF Core jest lekką, rozszerzalną, międzyplatformową wersją programu Entity Framework.



[Entity Framework 6](#)

Program EF 6 opiera się na sprawdzonej i przetestowanej technologii dostępu do danych będącej wynikiem wielu lat pracy nad funkcjami i stabilizacją.



Wybór

Dowiedz się, która wersja programu EF jest dla Ciebie odpowiednia.



[Przenoszenie do programu EF Core](#)

Wskazówki dotyczące przenoszenia istniejących aplikacji programów EF 6 do programu EF Core.

[EF Core](#)

[wszystko](#)

EF Core jest lekką, rozszerzalną, międzyplatformową wersją programu Entity Framework.



Wprowadzenie

[Omówienie](#)

[Tworzenie modelu](#)

[Tworzenie zapytania o dane](#)

[Zapisywanie danych](#)



Samouczki

[.NET Framework](#)

[.NET Core](#)

[ASP.NET Core](#)

[Platforma UWP](#)

[więcej...](#)

□

Dostawcy baz danych

[SQL Server](#)

[MySQL](#)

[PostgreSQL](#)

[SQLite](#)

[więcej...](#)

□

□ Dokumentacja interfejsu API

[DbContext](#)

[DbSet< TEntity >](#)

[więcej...](#)

[EF 6](#)

Program EF 6 opiera się na sprawdzonej i przetestowanej technologii dostępu do danych będącej wynikiem wielu lat pracy nad funkcjami i stabilizacją.

□

Rozpocznij

Dowiedz się, jak uzyskać dostęp do danych za pomocą programu Entity Framework 6.

□

□ Dokumentacja interfejsu API

Przeglądaj interfejs API programu Entity Framework 6 uporządkowany według przestrzeni nazw.

Porównanie programów EF Core i EF6

16.11.2018 • 9 minutes to read • [Edit Online](#)

Entity Framework to maper obiektowo relacyjny (O/RM) dla platformy .NET. W tym artykule porównano dwie wersje: Entity Framework 6 i programem Entity Framework Core.

Entity Framework 6

Entity Framework 6 (EF6) to technologia dostępu do danych i przetestowanej. Najpierw został wydany w 2008 roku jako część .NET Framework 3.5 SP1 i Visual Studio 2008 z dodatkiem SP1. Począwszy od wersji 4.1 jest dostarczana jako [EntityFramework](#) pakietu NuGet. EF6 jest uruchamiany w środowisku .NET Framework 4.x, co oznacza, że działa tylko na Windows.

Programy EF6 w dalszym ciągu być obsługiwane produktu i będą nadal widzieć poprawki błędów i drobne ulepszenia.

Entity Framework Core

Entity Framework Core (EF Core) jest pełne ponowne zapisywania adresów EF6, która została po raz pierwszy w 2016. Wysłaniem go w pakietach Nuget, głównym jest jeden [Microsoft.EntityFrameworkCore](#). EF Core jest produktem dla wielu platform, które można uruchamiać na platformy .NET Core lub .NET Framework.

EF Core zaprojektowano tak, aby zapewnić środowisko programistyczne, podobnie jak EF6. Większość interfejsów API najwyższego poziomu pozostają takie same, EF Core współpracując z znane deweloperów, którzy korzystali z platformy EF6.

Porównanie funkcji

EF Core oferuje nowe funkcje, które nie będą realizowane w EF6 (takie jak [klucze alternatywne](#), [aktualizacji zbiorczych](#), i [mieszanego ocena bazy danych i klienta w zapytaniach LINQ](#)). Ale ponieważ jest on nowy kod podstawowy, mu również pewne funkcje, które ma EF6.

W poniższej tabeli porównano funkcje dostępne w programu EF Core i EF6. Jest to porównania ogólne i nie listę wszystkich funkcji lub wyjaśniono różnice między tej samej funkcji w różnych wersjach programu EF.

Kolumna programu EF Core wskazuje wersję produktu, w którym najpierw pojawiły się tę funkcję.

Tworzenie modelu

FUNKCJA	EF 6	EF CORE
Mapowanie klasy podstawowe	Tak	1.0
Konstruktorów z parametrami		2.1
Konwersje wartości właściwości		2.1
Mapowanych typach bez kluczy (typy zapytań)		2.1
Konwencje	Tak	1.0

FUNKCJA	EF 6	EF CORE
Konwencje niestandardowe	Tak	1.0 (częściowa Obsługa)
Adnotacje danych	Tak	1.0
Interfejs Fluent API	Tak	1.0
Dziedziczenie: Tabel na hierarchii (TPH)	Tak	1.0
Dziedziczenie: Tabela według typu (TPT)	Tak	
Dziedziczenie: Tabel na konkretnej klasie (TPC)	Tak	
Właściwości stanu w tle		1.0
Klucze alternatywne		1.0
Wiele do wielu bez łączenia jednostek	Tak	
Generowanie klucza: bazy danych	Tak	1.0
Generowanie klucza: klienta		1.0
Typy złożone/właścicielem	Tak	2.0
Dane przestrzenne	Tak	2.2
Wizualizacja graficzna modelu	Tak	
Graficzny Edytor kodu	Tak	
Wzór format: kod	Tak	1.0
Wzór format: EDMX (XML)	Tak	
Tworzenie modelu z bazy danych: wiersza polecenia	Tak	1.0
Tworzenie modelu z bazy danych: Kreator programu VS	Tak	
Aktualizowanie modelu z bazy danych	Częściowe	
Filtры запытаний глобальных		2.0
Dzielenie tabeli	Tak	2.0
Podział jednostki	Tak	
Mapowanie funkcji skalarnej bazy danych	Słabo	2.0

FUNKCJA	EF 6	EF CORE
Mapowanie pól		1.1

Wykonanie zapytania o dane

FUNKCJA	EF6	EF CORE
zapytania LINQ	Tak	1.0 (w toku dla złożonych zapytań)
Elementu Readable wygenerowanego kodu SQL	Słabo	1.0
Ocena mieszane klient serwer		1.0
GroupBy tłumaczenia	Tak	2.1
Ładowanie powiązanych danych: Eager	Tak	1.0
Ładowanie powiązanych danych: Eager ładowania dla typów pochodnych		2.1
Ładowanie powiązanych danych: powolne	Tak	2.1
Ładowanie powiązanych danych: jawne	Tak	1.1
Pierwotne zapytania SQL: typy jednostek	Tak	1.0
Pierwotne zapytania SQL: typy innego niż jednostka (typy zapytań)	Tak	2.1
Pierwotne zapytania SQL: tworzenie za pomocą LINQ		1.0
Zapytania skompilowane jawnie	Słabo	2.0
Język zapytań tekstowych (jednostki SQL)	Tak	

Zapisywanie danych

FUNKCJA	EF6	EF CORE
Śledzenie zmian: migawki	Tak	1.0
Śledzenie zmian: powiadomień	Tak	1.0
Śledzenie zmian: serwery proxy	Tak	
Uzyskiwanie dostępu do śledzonych stanu	Tak	1.0

FUNKCJA	EF6	EF CORE
Optymistyczna współbieżność	Tak	1.0
Transakcje	Tak	1.0
Przetwarzanie wsadowe instrukcji		1.0
Mapowanie procedur składowanych	Tak	
Odłączony niskiego poziomu interfejsy API grafów	Słabo	1.0
Wykres odłączonego End-to-end		1.0 (częściowa Obsługa)

Inne funkcje

FUNKCJA	EF6	EF CORE
Migracje	Tak	1.0
Baza danych tworzenia/usuwania interfejsów API	Tak	1.0
Wstępne wypełnianie danych	Tak	2.1
Elastyczność połączenia	Tak	1.1
Cykl życia przechwytuje (zdarzenia, przejmowanie)	Tak	
Rejestrowanie proste (Database.Log)	Tak	
Buforowanie typu DbContext		2.0

Dostawcy baz danych

FUNKCJA	EF6	EF CORE
SQL Server	Tak	1.0
MySQL	Tak	1.0
PostgreSQL	Tak	1.0
Oracle	Tak	1.0 ⁽¹⁾
Bazy danych SQLite	Tak	1.0
SQL Server Compact	Tak	1.0 ⁽²⁾
DB2	Tak	1.0

FUNKCJA	EF6	EF CORE
Firebird	Tak	2.0
Jet (Microsoft Access)		w wersji 2.0 ⁽²⁾
Pamięć (do testowania)		1.0

¹ obecnie jest dostępna dla rozwiązań Oracle płatnych dostawcy. Trwa praca bezpłatne oficjalnego dostawcę dla Oracle.

² dostawcy programu SQL Server Compact i Jet działają tylko w programie .NET Framework (nie na platformie .NET Core).

Implementacje platformy .NET

FUNKCJA	EF6	EF CORE
.NET framework (Konsola, WinForms, WPF, ASP.NET)	Tak	1.0
.NET core (Konsola, platformy ASP.NET Core)		1.0
Narzędzie mono i Xamarin		1.0 (w toku)
Platforma UWP		1.0 (w toku)

Wskazówki dotyczące nowych aplikacji

Należy wziąć pod uwagę przy użyciu programu EF Core dla nowej aplikacji, jeśli są spełnione oba poniższe warunki:

- Aplikacja musi możliwości platformy .NET Core. Aby uzyskać więcej informacji, zobacz [Wybieranie między programami .NET Core i .NET Framework dla aplikacji serwerowych](#).
- EF Core obsługuje wszystkie funkcje, których wymaga aplikacja. Jeśli brakuje żądanej funkcji, sprawdź [harmonogram działania dla platformy EF Core](#) Aby dowiedzieć się, jeśli istnieją plany pomocy technicznej w przyszłości.

Należy wziąć pod uwagę przy użyciu platformy EF6, jeśli są spełnione oba poniższe warunki:

- Aplikacja będzie uruchamiana na Windows i program .NET Framework 4.0 lub nowszy.
- EF6 obsługuje wszystkie funkcje, których wymaga aplikacja.

Wskazówki dotyczące istniejących aplikacji EF6

Ze względu na fundamentalne zmiany w programie EF Core zaleca się przeniesienie aplikacji EF6 do programu EF Core, chyba że istnieje istotny powód, aby wprowadzić zmianę. Jeśli chcesz przejść do programu EF Core, aby korzystać z nowych funkcji, upewnij się, że masz świadomość jego pewne ograniczenia. Aby uzyskać więcej informacji, zobacz [przenoszenie z programu EF6 do programu EF Core](#). **Przenoszenie z programu EF6 do programu EF Core jest więcej niż uaktualnienie portu.**

Następne kroki

Aby uzyskać więcej informacji zobacz dokumentację:

- [Przegląd — EF Core](#)
- [Przegląd — EF6](#)

Przy użyciu programu EF Core i EF6 w tej samej aplikacji

28.08.2018 • 2 minutes to read • [Edit Online](#)

Istnieje możliwość użycia programu EF Core i EF6 w tej samej aplikacji .NET Framework lub biblioteki, instalując oba pakiety NuGet.

Niektóre typy takich samych nazwach w programie EF Core i EF6 i różnią się jedynie przestrzeni nazw, które mogą skomplikować przy użyciu programu EF Core i EF6, w tym samym pliku kodu. Niejednoznaczności można łatwo usunąć za pomocą dyrektywy aliasu przestrzeni nazw. Na przykład:

```
using Microsoft.EntityFrameworkCore; // use DbContext for EF Core  
using EF6 = System.Data.Entity; // use EF6.DbContext for the EF6 version
```

Jeśli są przenoszenie istniejących aplikacji, która ma wiele modeli EF, można selektywnie portu niektóre z nich do programu EF Core i kontynuować korzystanie z platformy EF6 dla innych użytkowników.

Przenoszenie z programu EF6 do programu EF Core

28.08.2018 • 2 minutes to read • [Edit Online](#)

Ze względu na fundamentalne zmiany w programie EF Core nie jest zalecane Próba przeniesienia aplikacji EF6 do programu EF Core, chyba że istnieje istotny powód, aby wprowadzić zmianę. Należy wyświetlić przenoszenie z programu EF6 do programu EF Core jako portu, a nie uaktualnienie.

Przed przenoszeniem z programu EF6 do programu EF Core: Sprawdzanie poprawności wymagań aplikacji

28.08.2018 • 5 minutes to read • [Edit Online](#)

Przed rozpoczęciem procesu przenoszenia należy sprawdzić, czy programu EF Core spełnia wymagania dotyczące dostępu do danych aplikacji.

Brak funkcji

Upewnij się, że programu EF Core zawiera wszystkie funkcje potrzebne do użycia w aplikacji. Zobacz [porównanie funkcji](#) szczegółowe porównanie porównanie zestaw w wersji EF Core funkcji platformy EF6. Jeśli brakuje dowolnej wymaganej funkcji, upewnij się, że możesz kompensować braku tych funkcji, przed eksportowaniem do programu EF Core.

Zmiany zachowania

To jest niepełna lista zmian w zachowaniu między EF6 i EF Core. Należy zachować te, aby pamiętać, jak port aplikacji zgodnie z nimi może zmienić sposób, w jaki aplikacja działa, ale nie będą wyświetlane jako błędy komplikacji po zamianie do programu EF Core.

Zachowanie DbSet.Add/Attach i wykres

W EF6 wywołanie `DbSet.Add()` na jednostki skutkuje wyszukiwanie cykliczne dla wszystkich jednostek, do którego odwołuje się jego właściwości nawigacyjne. Wszystkie jednostki, które znajdują się i nie są już śledzone przez kontekście, są również oznaczane w miarę dodawania. `DbSet.Attach()` zachowuje się tak samo, z wyjątkiem wszystkich jednostek są oznaczone jako niezmieniony.

EF Core wykonuje wyszukiwanie cykliczne podobne, ale niektóre nieco inne reguły.

- Jednostka główna jest zawsze żądany stan (dodane do `DbSet.Add` bez zmian dla `DbSet.Attach`).
- **W przypadku jednostek, które zostały znalezione w czasie cykliczne wyszukiwanie właściwości nawigacji:**
 - **Jeśli klucz podstawowy jednostki jest generowany magazynu**
 - Jeśli nie ustawiono klucza podstawowego z wartością, stan jest ustawiony do dodanych. Wartość klucza podstawowego jest uznawany za "nie jest ustawiona". Jeśli jest ona przypisana wartość domyślna CLR dla typu właściwości (na przykład `0` dla `int`, `null` dla `string` itp.).
 - Jeśli klucz podstawowy jest ustawiona na wartość, stan jest ustawiony bez zmian.
 - Jeśli klucz podstawowy nie jest generowany w bazie danych, jednostka jest umieszczany w tym samym stanie, jako katalog główny.

Kod pierwszy inicjowanie bazy danych

EF6 ma znacznej ilości magic, który wykonuje wokół wybranie połączenia z bazą danych i inicjowania bazy danych. Oto niektóre z tych reguł:

- Jeśli konfiguracja nie jest wykonywana, EF6 wybierze bazę danych programu SQL Express lub LocalDb.
- Jeśli parametry połączenia z taką samą nazwą jak kontekstu znajduje się w aplikacji `App/Web.config` pliku, to połączenie będzie używane.

- Jeśli baza danych nie istnieje, zostanie utworzony.
- Jeśli żadna z tabel z modelu istnieje w bazie danych, schematów dla bieżącego modelu jest dodawany do bazy danych. Jeśli migracja jest włączona, następnie służą one do utworzenia bazy danych.
- Jeśli baza danych istnieje i EF6 poprzednio utworzono schemat, schemat jest sprawdzane pod kątem zgodności z bieżącego modelu. Wyjątek jest generowany, jeśli model został zmieniony od czasu utworzenia schematu.

EF Core nie wykonuje tego magic.

- Połączenie z bazą danych muszą być jawnie skonfigurowane w kodzie.
- Inicjowanie nie jest wykonywane. Należy użyć `DbContext.Database.Migrate()` do zastosowania migracji (lub `DbContext.Database.EnsureCreated()` i `EnsureDeleted()` do tworzenia/usuwania bazy danych bez używania migracji).

Pierwsza tabela kodu konwencje nazewnictwa

EF6 uruchamia Nazwa klasy jednostki za pośrednictwem usługi pluralizacja do obliczania jednostki jest mapowany na domyślną nazwę tabeli.

EF Core używa nazwy `DbSet` właściwości jednostki jest widoczna w kontekście pochodnych. Jeśli jednostka ma `DbSet` właściwości, a następnie nazwę klasy jest używany.

Przenoszenie modelu opartego na EDMX EF6 do programu EF Core

28.08.2018 • 2 minutes to read • [Edit Online](#)

EF Core nie obsługuje formatu pliku EDMX dla modeli. Najlepszym rozwiązaniem do portu tych modeli jest do generowania nowego modelu opartego na kodzie z bazy danych dla aplikacji.

Instalowanie pakietów programu EF Core NuGet

Zainstaluj `Microsoft.EntityFrameworkCore.Tools` pakietu NuGet.

Ponowne generowanie modelu

Funkcje odtwarzania można teraz używać do tworzenia modeli, w oparciu o istniejącą bazę danych.

Uruchom następujące polecenie w konsoli Menedżera pakietów (Narzędzia -> Menedżer pakietów NuGet -> Konsola Menedżera pakietów). Zobacz [Konsola Menedżera pakietów \(Visual Studio\)](#) opcji polecenia do tworzenia szkieletu podzbioru tabel itp.

```
Scaffold-DbContext "<connection string>" <database provider name>
```

Na przykład poniżej przedstawiono polecenia do tworzenia szkieletu modelu z bazy danych do obsługi blogów w ramach wystąpienia programu SQL Server LocalDB.

```
Scaffold-DbContext "Server=(localdb)\mssqllocaldb;Database=Blogging;Trusted_Connection=True;"  
Microsoft.EntityFrameworkCore.SqlServer
```

Usuń EF6 model

EF6 model należy teraz usunąć z aplikacji.

Jest dobrym rozwiązaniem pozostawić pakiet NuGet platformy EF6 (EntityFramework), zainstalowane, zgodnie z programem EF Core i EF6 mogą być używane side-by-side w tej samej aplikacji. Jednak w przypadku korzystania z platformy EF6 w żadnych obszarów aplikacji nie są pomyślny przebieg operacji, następnie odinstalowaniu pakietu pomoże spowodować błędy kompilacji na fragmenty kodu, które wymagają uwagi.

Aktualizowanie kodu

W tym momencie jest kwestią adresowania błędów kompilacji i recenzowania kodu, aby sprawdzić, czy zmiany zachowania między EF6 i EF Core będzie miało wpływ na możesz.

Testowanie portu

Po prostu, ponieważ kompiluje aplikację, nie znaczy, że pomyślnie są przenoszone do programu EF Core. Należy przetestować wszystkie obszary w aplikacji, aby upewnić się, że żadne zmiany zachowania niekorzystnie wpłynąć na nie wpływ aplikacji.

TIP

Zobacz [rozpoczęcie pracy z programem EF Core programu ASP.NET Core z istniejącej bazy danych](#) Aby uzyskać dodatkowe informacje na temat sposobu pracy z istniejącej bazy danych

Przenoszenie modelu opartego na kodzie EF6 do programu EF Core

28.08.2018 • 4 minutes to read • [Edit Online](#)

Jeśli wszystkie ostrzeżenia zostały przeczytane, możesz przystąpić do portu, poniżej przedstawiono wskazówki, które pomogą Ci rozpoczęć pracę.

Instalowanie pakietów programu EF Core NuGet

Aby korzystać z programu EF Core, zainstaluj pakiet NuGet dla dostawcy bazy danych, którego chcesz użyć. Na przykład, gdy obiektem docelowym programu SQL Server, można zainstalować `Microsoft.EntityFrameworkCore.SqlServer`. Zobacz [dostawcy baz danych](#) Aby uzyskać szczegółowe informacje.

Jeśli planujesz użyć migracji, a następnie należy również zainstalować `Microsoft.EntityFrameworkCore.Tools` pakietu.

Jest dobrym rozwiązaniem pozostawić pakiet NuGet platformy EF6 (EntityFramework), zainstalowane, zgodnie z programem EF Core i EF6 mogą być używane side-by-side w tej samej aplikacji. Jednak w przypadku korzystania z platformy EF6 w żadnych obszarów aplikacji nie są pomyślny przebieg operacji, następnie odinstalowaniu pakietu pomoże spowodować błędy komplikacji na fragmenty kodu, które wymagają uwagi.

Przestrzenie nazw wymiany

Większość interfejsów API, których używasz w EF6 znajdują się w `System.Data.Entity` przestrzeni nazw (i pokrewnych przestrzeniach nazw sub). Pierwszy zmiana kodu jest możliwa zamienić na `Microsoft.EntityFrameworkCore` przestrzeni nazw. Będzie zazwyczaj rozpoczęć pochodnej kontekstu pliku kodu, a następnie pozwolimy na opracowanie stamtąd adresowania błędów komplikacji w miarę ich występowania.

Konfiguracja kontekstu (połączenie itp.)

Zgodnie z opisem w [upewnij się, EF Core będzie pracą dla aplikacji](#), programem EF Core ma mniej magic wokół wykrywanie bazy danych, aby nawiązać połączenie. Należy zastąpić `OnConfiguring` metodę w pochodnej kontekstu i użyj interfejsu API określonego dostawcy bazy danych, aby skonfigurować połączenie z bazą danych.

Większość aplikacji EF6 przechowywanie parametrów połączenia w aplikacjach `App/Web.config` pliku. W programie EF Core przeczytanie tego parametry połączenia za pomocą `ConfigurationManager` interfejsu API. Może być konieczne dodanie odwołania do `System.Configuration` zestawu struktury, aby można było używać tego interfejsu API.

```
public class BloggingContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }
    public DbSet<Post> Posts { get; set; }

    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    {
        optionsBuilder.UseSqlServer(ConfigurationManager.ConnectionStrings["BloggingDatabase"].ConnectionString);
    }
}
```

Aktualizowanie kodu

W tym momencie jest kwestią adresowania błędy komplikacji i przeglądania kodu, aby zobaczyć zmiany zachowania będzie miało wpływ użytkownik.

Migracja istniejących

Tak naprawdę nie jest to możliwe sposobu portu istniejących migracji platformy EF6 do programu EF Core.

Jeśli to możliwe najlepiej Załóżmy, że zostały zastosowane wszystkie poprzednich migracji z programu EF6 do bazy danych, a następnie rozpoczęcia migracji schematu od tego punktu, przy użyciu programu EF Core. Aby to zrobić, należy użyć `Add-Migration` polecenie do dodania do migracji, gdy model jest przenoszone do programu EF Core.

Następnie należy usunąć cały kod z `Up` i `Down` metody szkieletu migracji. Porównuje następnej migracji do modelu podczas tej początkowej migracji został szkielet.

Testowanie portu

Po prostu, ponieważ komplikuje aplikację, nie znaczy, że pomyślnie są przenoszone do programu EF Core. Należy przetestować wszystkie obszary w aplikacji, aby upewnić się, że żadne zmiany niekorzystnie wpłynąć na nie wpływ aplikacji.

Entity Framework Core

01.11.2018 • 2 minutes to read • [Edit Online](#)

Entity Framework (EF) Core to lekka, rozszerzalna i wieloplatformowa wersja popularnej technologii dostępu do danych — Entity Framework.

EF Core może służyć jako maper obiektowo relacyjny (O/RM), dzięki czemu deweloperzy platformy .NET mogą pracować z bazą danych, używając obiektów platformy .NET i eliminując potrzebę pisania większości kodu dostępu do danych.

EF Core obsługuje wiele aparatów baz danych, zobacz [Dostawcy baz danych](#), aby uzyskać szczegółowe informacje.

Model

Z programem EF Core dostęp do danych odbywa się przy użyciu modelu. Model składa się z klas jednostek i pochodnej kontekstu, który reprezentuje sesję z bazą danych, pozwalając na zapytania i zapisywanie danych. Zobacz [Tworzenie modelu](#), aby dowiedzieć się więcej.

Możesz wygenerować model z istniejącej bazy danych, przekazać w kodzie model pasujący do Twojej bazy danych albo użyć EF Migrations, aby utworzyć bazę danych z podanego modelu (i rozwijać ją w miarę zmian modelu z upływem czasu).

```
using Microsoft.EntityFrameworkCore;
using System.Collections.Generic;

namespace Intro
{
    public class BloggingContext : DbContext
    {
        public DbSet<Blog> Blogs { get; set; }
        public DbSet<Post> Posts { get; set; }

        protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
        {
            optionsBuilder.UseSqlServer(@"Server=
(loclaldb)\mssqllocaldb;Database=MyDatabase;Trusted_Connection=True;");
        }
    }

    public class Blog
    {
        public int BlogId { get; set; }
        public string Url { get; set; }
        public int Rating { get; set; }
        public List<Post> Posts { get; set; }
    }

    public class Post
    {
        public int PostId { get; set; }
        public string Title { get; set; }
        public string Content { get; set; }

        public int BlogId { get; set; }
        public Blog Blog { get; set; }
    }
}
```

Wykonywanie zapytań

Wystąpienia klas jednostek są pobierane z bazy danych przy użyciu języka Language Integrated Query (LINQ). Zobacz [Wykonywanie zapytania o dane](#), aby dowiedzieć się więcej.

```
using (var db = new BloggingContext())
{
    var blogs = db.Blogs
        .Where(b => b.Rating > 3)
        .OrderBy(b => b.Url)
        .ToList();
}
```

Zapisywanie danych

Dane są tworzone, usuwane i modyfikowane w bazie danych za pomocą wystąpień klas jednostek. Zobacz [zapisywanie danych](#), aby dowiedzieć się więcej.

```
using (var db = new BloggingContext())
{
    var blog = new Blog { Url = "http://sample.com" };
    db.Blogs.Add(blog);
    db.SaveChanges();
}
```

Następne kroki

Aby skorzystać z samouczków wprowadzających, zobacz [Wprowadzenie do platformy Entity Framework Core](#).

Co nowego w programie EF core

05.12.2018 • 2 minutes to read • [Edit Online](#)

Aby dowiedzieć się więcej o nowych funkcjach w każdej wersji, można użyć poniższych linków:

Przyszłe wersje

- [Plany rozwoju programu EF Core](#)

Najnowsze wersje

- [EF Core 2.2 \(najnowsza stabilna wersja\)](#)
- [EF Core 2.1](#)

Poprzednich wersjach

- [EF Core 2.0](#)
- [EF Core 1.1](#)
- [EF Core 1.0](#)

Entity Framework Core plan

07.01.2019 • 13 minutes to read • [Edit Online](#)

IMPORTANT

Należy pamiętać, że zestawy funkcji i harmonogramy przyszłych wersji zawsze mogą ulec zmianie i czasu, mimo że firma Microsoft podejmie próbę tej strony na bieżąco, jego mogą nie odzwierciedlać najnowszych planów na wszystkich.

EF Core 3.0

Przy użyciu programu EF Core 2.2 spory naszym głównym celem jest teraz EF Core 3.0, które mają zostać wyrównane przy użyciu platformy .NET Core 3.0 i zwalnia ASP.NET 3.0.

Firma Microsoft nie wykonano żadnych nowych funkcji, więc [EF Core 3.0 w wersji zapoznawczej 1 pakiety opublikowane w galerii NuGet](#) grudnia 2018 r. tylko zawierać [poprawki, drobne ulepszenia i zmiany, które wprowadziliśmy w Przygotowanie do pracy 3.0](#).

W rzeczywistości nadal trzeba dostosować naszych [wersji, planowania](#) 3.0, aby upewnić się, mamy właściwy zestaw funkcji, które można wykonać w wyznaczonym czasie. Firma Microsoft udostępni więcej informacji, uzyskujemy bardziej przejrzysty, ale poniżej przedstawiono pewne ogólne motywy i funkcje, firma Microsoft intend pracować nad:

- **Ulepszenia zapytań LINQ (#12795):** LINQ umożliwia tworzenie zapytań bazy danych bez opuszczania usługi z wybranego języka, wykorzystując zaawansowane wpisy informacje funkcji IntelliSense i sprawdzanie typów w czasie komplikacji. Ale LINQ można również napisać nieograniczonej liczby zapytań skomplikowane, a który zawsze było ogromnym wyzwaniem dla dostawców LINQ. W pierwszym kilku wersji programu EF Core możemy rozwiązać, w części za ustalenie, jakie części zapytania mogą być tłumaczone do bazy danych SQL, a następnie, umożliwiając pozostałe kwerenda do wykonania w pamięci na komputerze klienckim. Wykonanie tego po stronie klienta może być pożądana w niektórych sytuacjach, ale w innych przypadkach może to skutkować nieefektywne zapytań, które nie mogą być określone, dopóki aplikacja jest wdrażana w środowisku produkcyjnym. W programie EF Core 3.0 planujemy wprowadzić głęboki zmiany sposobu działania naszej implementacji LINQ i jak możemy przetestować. Cele są aby działał on bardziej niezawodnie (na przykład, aby uniknąć przerywanie zapytania w wersjach poprawki), aby można było tłumaczenie więcej wyrażeń poprawnie w bazie SQL, można wygenerować wydajne zapytania w większości przypadków i uniemożliwić nieefektywne zapytania przechodząc niewykryte.
- **Usługa cosmos DB obsługuje (#8443):** Pracujemy nad dostawcą usługi Cosmos DB dla platformy EF Core, aby umożliwić deweloperom zapoznać się z modelem programistycznym EF łatwo Kieruj usługi Azure Cosmos DB jako bazę danych aplikacji. Celem jest zapewnienie niektóre korzyści wynikające z usługi Cosmos DB, takie jak dystrybucji globalnej, "zawsze włączone" dostępność, elastyczną skalowalność i małego opóźnienia, jeszcze bardziej dostępne dla deweloperów platformy .NET. Dostawca umożliwi większość funkcji EF Core, takie jak automatyczna zmiana, śledzenie, LINQ i konwersji wartości, przy użyciu interfejsu SQL API w usłudze Cosmos DB. Rozpoczeliśmy ten nakład pracy przed programem EF Core 2.2, i [Wprowadziliśmy pewne wersji dostawcy dostępne w wersji zapoznawczej](#). Nowy plan jest, aby kontynuować, tworzenie dostawcy wraz z programem EF Core 3.0.
- **C#Obsługa 8.0 (#12047):** Chcemy, aby klienci mogły korzystać z niektórych [nowe funkcje zostaną dodane w C# 8.0](#) strumieni asynchronicznych, takich jak (w tym await dla każdego) i typy referencyjne dopuszczającego wartość null podczas korzystania z programu EF Core.
- **Odwroć inżynierów widoki bazy danych do typów zapytań (#1679):** W programie EF Core 2.1

Dodaliśmy obsługę typów zapytań, które mogą reprezentować dane, które mogą być odczytywane z bazy danych, ale nie można zaktualizować. Typy zapytań są widoki doskonałe rozwiązań dla mapowania bazy danych, więc w programie EF Core 3.0, prosimy o poświęcenie do zautomatyzowania tworzenia typów zapytań na widoki baz danych.

- **Właściwość zbiór jednostek (#13610 i #9914):** Ta funkcja jest zapewnieniem jednostek, które przechowują dane w właściwości indeksowanych zamiast regularnego właściwości, a także o bieżącym mogąc korzystać z wystąpienie klasy .NET (potencjalnie coś tak proste, jak `Dictionary<string, object>`) do reprezentowania typów różnych jednostek w tym samym modelu platformy EF Core. Ta funkcja jest kamień przechodzenia krok po kroku do obsługi relacji wiele do wielu, bez jednostki sprzężenia, który jest jednym z najbardziej pożądanych ulepszenia dla platformy EF Core.
- **EF 6.3 na platformie .NET Core (EF6 #271):** Rozumiemy, że wiele istniejących aplikacji używa starszych wersji programu EF i że przenoszenia ich do programu EF Core tylko po to, aby móc korzystać z platformy .NET Core może czasem wymagać znaczących nakładów pracy. Z tego powodu firma Microsoft będzie mogła dostosowanie Następna wersja programu EF 6 do uruchamiania na .NET Core 3.0. Robimy to w celu ułatwienia przenoszenia istniejących aplikacji przy minimalnych zmianach. Czy powstaną pewne ograniczenia (na przykład będzie wymagać nowego dostawcy, obsługa przestrzennych z programem SQL Server nie jest włączony) i są planowane do programów EF 6 żadne nowe funkcje.

W międzyczasie możesz użyć [tego zapytania w naszym narzędziu do śledzenia problemów](#) aby zobaczyć elementy robocze wstępnie przypisane do wersji 3.0.

Harmonogram

Harmonogram dla platformy EF Core jest zsynchronizowany z [harmonogram platformy .NET Core](#) i [harmonogram platformy ASP.NET Core](#).

Zaległości

[Punkt kontrolny zaległości](#) w naszym problem śledzenia zawiera problemy, oczekujemy, że działasz kiedyś lub uważamy, że ktoś od społeczności można analizować. Klienci mogą przesyłać komentarze i głosów na temat tych problemów — Zapraszamy. Współautorzy chcą pracować na dowolnym z tych problemów są zachęcani do najpierw Rozpocznij dyskusję na temat sposobu ich podejście.

Nigdy nie jest gwarancją, jaką pracujemy nad tym, na dowolnej danej funkcji w określonej wersji programu EF Core. Tak jak wszystkie projekty oprogramowania priorytetów, harmonogramy wersji i dostępnych zasobów można zmienić w dowolnym momencie. Ale jeśli Chcieliśmy rozwiązać problem w określonym przedziale czasu, firma Microsoft będzie przypisać go do punktu kontrolnego wersji, a nie punktu kontrolnego zaległości. Firma Microsoft regularnie przenoszenie problemów punkty kontrolne wydania i zaległości w ramach naszych [wersji procesu planowania](#).

Firma Microsoft będzie prawdopodobnie Zamknij problem, jeśli firma Microsoft nie jest planowane kiedykolwiek rozwiązać problem. Ale możemy ponowne rozpatrzenie problem, który możemy wcześniej zamknięte, jeśli możemy uzyskać nowe informacje o nim.

Proces planowania wydania

Uzyskujemy często zadawane pytania dotyczące sposobu Wybierzmy określonych funkcji, aby przejść do określonej wersji. Naszych planach na pewno nie przekłada się automatycznie w planach wydania. Obecność funkcją EF6 również nie automatycznie oznacza, że ta funkcja musi zostać wdrożone w programie EF Core.

Trudno szczegółowo cały proces wykonamy planowanie wydania. Część informacji jest Omawiając określone funkcje, możliwości i priorytety, a sam proces ewoluje się również z każdym wydaniem. Jednak firma Microsoft Podsumowując często zadawanych pytań, którą spróbujemy odpowiedzieć przy podejmowaniu decyzji co do pracy

po kliknięciu przycisku Dalej:

1. **Deweloperzy liczbę naszym zdaniem będzie używać tej funkcji i jak dużo lepiej wprowadzi na ich/korzystanie z aplikacji?** Odpowiedzi na to, firma Microsoft zbieranie opinii z wielu źródeł, komentarze i głosów problemów jest jednym z tych źródeł.
2. **Co to są osób rozwiązań można użyć, jeśli firma Microsoft nie jeszcze zaimplementowała tę funkcję?** Na przykład wielu deweloperów można mapować tabelę sprzężenia w celu obejścia Brak natywnej obsługi wiele do wielu. Oczywiście nie wszystkim deweloperom chcesz to zrobić, ale można wiele i który jest liczona jako czynnika podczas podjęcie decyzji.
3. **Wdrażanie tej funkcji ewolucji architektury programu EF Core tak, aby przemieszczał się nam przybliżyć do wykonania innych funkcji?** Firma Microsoft zwykle preferować funkcje, które działają jako bloków konstrukcyjnych dla innych funkcji. Na przykład właściwość zbiór jednostek, ułatwisz nam idą w kierunku obsługi wiele do wielu i konstruktory jednostki włączono obsługę ładowania z opóźnieniem.
4. **Funkcja punkt rozszerzeń?** Zwykle aby preferował punkty rozszerzeń za pośrednictwem funkcji normalne, ponieważ umożliwiają one programistom podłączyć ich zachowania, a także kompensuje wszelkie brakujące funkcje.
5. **Co to jest współdziałanie funkcji w połączeniu z innymi produktami?** Będziemy preferować funkcje, które włącza lub znacznie poprawić środowisko przy użyciu programu EF Core przy użyciu innych produktów, takich jak .NET Core najnowszą wersję programu Visual Studio, Microsoft Azure, itp.
6. **Co to są umiejętności osób, które są dostępne w funkcji oraz jak najlepiej wykorzystać te zasoby?** Każdy członek zespołu platformy EF i naszej społeczności ma różne poziomy doświadczenia w innych obszarach, więc musimy odpowiednio zaplanować. Nawet wtedy, gdy chcemy mieć "cały zespół na pokładzie" pracy w określonych funkcji, takich jak tłumaczenia GroupBy lub wiele do wielu, byłoby niepraktyczne.

Jak wspomniano wcześniej, ten proces ewoluje w każdej wersji. W przyszłości zostanie podjęta próba dodać więcej możliwości dla członków społeczności. Podaj dane wejściowe w naszych planach wydania. Na przykład chcemy ułatwić przeglądanie projektów projektu funkcji i samego planu wersji.

Nowe funkcje programu EF Core 2.2

16.11.2018 • 4 minutes to read • [Edit Online](#)

Obsługa danych przestrzennych

Dane przestrzenne może służyć do reprezentowania fizycznej lokalizacji i kształt obiektów. Wiele baz danych natywnie można przechowywać, indeksu i wykonywanie zapytań o dane przestrzenne. Typowe scenariusze obejmują tworzenie zapytań dotyczących obiektów w określonej odległości i testowania, jeśli wielokąta zawiera danej lokalizacji. Obsługuje teraz EF Core 2.2 z Praca z danymi przestrzennymi z różnymi bazami danych przy użyciu typów z [NetTopologySuite](#) biblioteki (NTS).

Obsługa danych przestrzennych jest zaimplementowana jako szereg pakietów rozszerzeń właściwe dla dostawcy. Każda z tych pakietów wspiera mapowanie dla typów nktury przerwania i metod oraz odpowiadające typy przestrzenne i funkcji w bazie danych. Takie rozszerzenia dostawcy są teraz dostępne dla [programu SQL Server](#), [SQLite](#), i [PostgreSQL](#) (z [projektu Npgsql](#)). Typy przestrzennych mogą być używane bezpośrednio z [dostawcy w pamięci programu EF Core](#) bez dodatkowych rozszerzeń.

Po zainstalowaniu rozszerzenia dostawcy można dodać właściwości obsługiwane typy do jednostek. Na przykład:

```
using NetTopologySuite.Geometries;

namespace MyApp
{
    public class Friend
    {
        [Key]
        public string Name { get; set; }

        [Required]
        public Point Location { get; set; }
    }
}
```

Można następnie zachować jednostkę o dane przestrzenne:

```
using (var context = new MyDbContext())
{
    context.Add(
        new Friend
        {
            Name = "Bill",
            Location = new Point(-122.34877, 47.6233355) {SRID = 4326 }
        });
    context.SaveChanges();
}
```

I można wykonywać zapytania bazy danych na podstawie danych przestrzennych i operacje:

```
var nearestFriends =
    (from f in context.Friends
    orderby f.Location.Distance(myLocation) descending
    select f).Take(5).ToList();
```

Aby uzyskać więcej informacji na temat tej funkcji, zobacz [dokumentacji typów przestrzennych](#).

Kolekcje jednostki należące do firmy

EF Core 2.0 dodanie do użytkowania model w jeden do jednego skojarzenia. EF Core 2.2 rozszerza możliwości express własność skojarzenia jeden do wielu. Własność pomaga ograniczyć, jak jednostki są używane.

Na przykład należących do jednostki:

- Może się pojawić tylko dla właściwości nawigacji innych typów jednostek.
- Są ładowane automatycznie i może być śledzone tylko przez DbContext wraz z ich właścicielem.

Relacyjne bazy danych kolekcji należące do firmy są mapowane do oddzielnych tabel z jego właściciela, podobnie jak regularne skojarzenia jeden do wielu. Jednak w bazach danych korzystający z dokumentów, planujemy zagnieździć należących do podmiotów (w kolekcji należącej do firmy lub odwołania) w obrębie tego samego dokumentu jako właściciel.

Można użyć tej funkcji przez wywołanie metody nowy interfejs API OwnsMany():

```
modelBuilder.Entity<Customer>().OwnsMany(c => c.Addresses);
```

Aby uzyskać więcej informacji, zobacz [zaktualizowane należących do jednostek dokumentacja](#).

Tagi kwerendy

Ta funkcja ułatwia korelacja zapytania LINQ w kodzie za pomocą wygenerowanego zapytań SQL przechwycone w dziennikach.

Aby móc korzystać z kwerendy, tagi, dodawać adnotacje zapytania LINQ za pomocą nowej metody TagWith(). Przy użyciu zapytań przestrzennych z poprzedniego przykładu:

```
var nearestFriends =
    (from f in context.Friends.TagWith(@"This is my spatial query!")
     orderby f.Location.Distance(myLocation) descending
     select f).Take(5).ToList();
```

To zapytanie LINQ generuje następujące dane wyjściowe SQL:

```
-- This is my spatial query!

SELECT TOP(@__p_1) [f].[Name], [f].[Location]
FROM [Friends] AS [f]
ORDER BY [f].[Location].STDistance(@__myLocation_0) DESC
```

Aby uzyskać więcej informacji, zobacz [zapytania tagów dokumentacji](#).

Nowe funkcje programu EF Core 2.1

13.09.2018 • 11 minutes to read • [Edit Online](#)

Oprócz licznych poprawkach błędów i małych ulepszenia wydajności i funkcjonalności programu EF Core 2.1 zawiera niektóre istotne nowe funkcje:

Ładowanie z opóźnieniem

EF Core zawiera teraz konieczne bloki konstrukcyjne dla każdego, kto Tworzenie klas jednostek, które mogą ładować swoje właściwości nawigacji na żądanie. Utworzyliśmy również nowy pakiet

Microsoft.EntityFrameworkCore.Proxies, który wykorzystuje te bloki konstrukcyjne do produkcji proxy powolne ładowanie klas, w oparciu o co najmniej modyfikacji klas jednostek (na przykład klasy z właściwością nawigacji wirtualnego).

[Odczyt sekcji na ładowanie z opóźnieniem](#) Aby uzyskać więcej informacji na ten temat.

Parametry w konstruktorach jednostki

Tworzenie jednostek, które przyjmują parametry w ich konstruktory mogą umożliwić jako jeden z bloków konstrukcyjnych wymagane do załadowania z opóźnieniem. Parametry można użyć do dodania wartości właściwości, delegatów powolne ładowanie i usług.

[Odczyt sekcji jednostki konstruktora z parametrami](#) Aby uzyskać więcej informacji na ten temat.

Konwersje wartości

Do tej pory programu EF Core może mapować tylko właściwości typów natywnie obsługiwane przez dostawcę podstawowej bazy danych. Wartości zostały skopiowane i z powrotem między kolumnami i właściwości bez przetwarzania. Począwszy od programu EF Core 2.1 konwersji wartości można zastosować do przekształcania wartości z kolumn przed są stosowane do właściwości i na odwrót. Mamy wiele konwersji, które mogą być stosowane zgodnie z Konwencją, zgodnie z potrzebami, a także interfejsu API jawnego konfiguracji, który umożliwia rejestrowanie niestandardowe konwersje między kolumnami i właściwości. Zastosowanie tej funkcji, należą:

- Przechowywanie typów wyliczeniowych w postaci ciągów
- Mapowanie niepodpisane liczby całkowite z programem SQL Server
- Automatyczne szyfrowanie i odszyfrowywanie wartości właściwości

[Odczyt sekcji konwersji wartości](#) Aby uzyskać więcej informacji na ten temat.

Tłumaczenie LINQ GroupBy

Przed wersją 2.1 w wersji EF Core operatorów GroupBy LINQ zawsze oceniono w pamięci. Obsługujemy teraz go tłumaczenia klauzuli SQL GROUP BY w najbardziej typowe przypadki.

W tym przykładzie przedstawiono zapytanie z GroupBy używany do obliczania różne funkcje agregujące:

```
var query = context.Orders
    .GroupBy(o => new { o.CustomerId, o.EmployeeId })
    .Select(g => new
    {
        g.Key.CustomerId,
        g.Key.EmployeeId,
        Sum = g.Sum(o => o.Amount),
        Min = g.Min(o => o.Amount),
        Max = g.Max(o => o.Amount),
        Avg = g.Average(o => o.Amount)
    });
});
```

Odpowiednie tłumaczenia SQL wygląda następująco:

```
SELECT [o].[CustomerId], [o].[EmployeeId],
       SUM([o].[Amount]), MIN([o].[Amount]), MAX([o].[Amount]), AVG([o].[Amount])
  FROM [Orders] AS [o]
 GROUP BY [o].[CustomerId], [o].[EmployeeId];
```

Wstępne wypełnianie danych

Z pomocą nowej wersji będzie możliwe zapewnienie początkowe dane, aby wypełnić bazę danych. W odróżnieniu od w EF6, wstępne wypełnianie danych jest skojarzony typ jednostki jako część konfiguracji modelu. Następnie migracje EF Core można automatycznie obliczyć co Wstawianie, aktualizowanie lub usuwanie potrzebę operacji mają być stosowane podczas aktualniania bazy danych do nowej wersji modelu.

Na przykład można użyć go do skonfigurowania danych inicjatora dla wpisu w `OnModelCreating`:

```
modelBuilder.Entity<Post>().HasData(new Post{ Id = 1, Text = "Hello World!" });
```

Odczyt sekcji na wstępne wypełnianie danych Aby uzyskać więcej informacji na ten temat.

Typy zapytań

Modelu platformy EF Core mogą teraz zawierać typy zapytania. Inaczej niż w przypadku typów jednostek, typy zapytań nie kluczy zdefiniowane i nie można wstawić, usunąć lub zaktualizować (oznacza to, że są one tylko do odczytu), ale mogą być zwrócone bezpośrednio przez zapytania. Scenariusze użycia dla typów zapytań, należą:

- Mapowanie do widoków bez kluczy podstawowych
- Mapowania tabel bez kluczy podstawowych
- Mapowanie do zapytań zdefiniowanych w modelu
- Służy jako typ zwracany dla `FromSql()` zapytań

Odczyt sekcji na typy zapytań Aby uzyskać więcej informacji na ten temat.

Zawierają typy pochodne

Zostanie on teraz jest to możliwe, aby określić właściwości nawigacji tylko wobec podczas pisania wyrażeń dla typów pochodnych `Include` metody. Silnie typizowaną wersję `Include`, obsługujemy za pomocą jawnego rzutowania lub `as` operatora. Firma Microsoft obsługuje teraz również odwołuje się do nazwy właściwości nawigacji zdefiniowany dla typów pochodnych w wersję ciągę `Include`:

```
var option1 = context.People.Include(p => ((Student)p).School);
var option2 = context.People.Include(p => (p as Student).School);
var option3 = context.People.Include("School");
```

Odczyt sekcji [Dołącz z typami pochodnymi](#) Aby uzyskać więcej informacji na ten temat.

Obsługa System.Transactions

Dodaliśmy możliwość współpracy z System.Transactions funkcje, takie jak TransactionScope. Będzie to działać dla platformy .NET Core i .NET Framework podczas korzystania z dostawcy baz danych, które go obsługują.

Odczyt sekcji [System.Transactions](#) Aby uzyskać więcej informacji na ten temat.

Lepsze kolejność kolumn w początkowej migracji

Na podstawie opinii klientów Zaktualizowaliśmy migracji można wstępnie wygenerować kolumn dla tabel w tej samej kolejności, ponieważ właściwości są deklarowane w klasach. Należy pamiętać, programem EF Core nie można zmienić kolejności po dodaniu nowych elementów członkowskich po utworzeniu początkowego tabeli.

Optymalizacja skorelowany podzapytań

Ulepszyliśmy nasze translacji zapytania, aby uniknąć wykonywania "N + 1" zapytania SQL w wielu typowych scenariuszy, w których użycie właściwości nawigacji w projekcji prowadzi do łączenie danych z zapytania katalogu głównego przy użyciu danych z skorelowane podzapytanie. Optymalizacja, wymagane jest buforowanie wyników z podzapytanie, a firma Microsoft wymaga, aby zmodyfikować zapytanie, aby zdecydować się na nowe zachowanie.

Na przykład następujące zapytanie zwykle pobiera przetłumaczone na jednej kwerendzie dla klientów, a także N (gdzie "N" oznacza liczbę klientów zwrócił) oddzielnych zapytań dla zleceń:

```
var query = context.Customers.Select(
    c => c.Orders.Where(o => o.Amount > 100).Select(o => o.Amount));
```

Jeśli dołączysz `ToList()` w odpowiednim miejscu, wskazujesz, że buforowanie jest odpowiednia dla zamówienia, które umożliwiają optymalizację:

```
var query = context.Customers.Select(
    c => c.Orders.Where(o => o.Amount > 100).Select(o => o.Amount).ToList());
```

Należy pamiętać, że to zapytanie będzie tłumaczona tylko dwa kwerendy SQL: jeden dla klientów i kolejny zamówień.

Atrybut [należące do firmy]

Teraz jest możliwa do skonfigurowania [posiadane typy jednostek](#), po prostu Dodawanie adnotacji do typu z `[Owned]` a następnie sprawdzając, czy jednostka właściciel jest dodawany do modelu:

```
[Owned]
public class StreetAddress
{
    public string Street { get; set; }
    public string City { get; set; }
}

public class Order
{
    public int Id { get; set; }
    public StreetAddress ShippingAddress { get; set; }
}
```

Narzędzie wiersza polecenia dotnet-ef zawarte w zestawie SDK programu .NET Core

Dotnet ef polecenia są teraz częścią programu .NET Core SDK, w związku z tym nie będzie konieczne użycie DotNetCliToolReference w projekcie, aby można było użyć migracji lub tworzenia szkieletu DbContext z istniejącej bazy danych.

Zobacz sekcję dotyczącą [instalowania narzędzi](#) Aby uzyskać więcej informacji o sposobie włączania narzędzia wiersza polecenia dla różnych wersji zestawu SDK programu .NET Core i programem EF Core.

Pakiet Microsoft.EntityFrameworkCore.Abstractions

Nowy pakiet zawiera atrybuty i interfejsy, które można użyć w swoich projektach, aby wzbogacić funkcji EF Core bez zależna od programu EF Core jako całości. Na przykład atrybut [posiadane] i interfejs ILazyLoader znajdują się w tym miejscu.

Zdarzenia zmiany stanu

Nowe `Tracked` i `StateChanged` zdarzenia `ChangeTracker` może być użyty do zapisu logikę, która reaguje na jednostki, wprowadzając kontekstu DbContext lub zmianę ich stanu.

Nieprzetworzone analizatora parametru SQL

Nowy analizator kodu jest dołączana do programu EF Core, który wykrywa potencjalnie niebezpieczną użycia interfejsów API raw SQL, takie jak `FromSql` lub `ExecuteSqlCommand`. Na przykład dla następującego zapytania, zobaczy ostrzeżenie ponieważ `minAge` nie jest sparametryzowane:

```
var sql = $"SELECT * FROM People WHERE Age > {minAge}";
var query = context.People.FromSql(sql);
```

Zgodności dostawcy bazy danych

Zaleca się używać programu EF Core 2.1 z dostawcami, które zostały zaktualizowane lub co najmniej przetestowany w celu pracy z programem EF Core 2.1.

TIP

Jeśli okaże się wszelkie nieoczekiwane niezgodności dowolne wysłać nowych funkcji lub jeśli chcesz przesłać opinię na nich, zgłoś go za pomocą [nasze narzędzia do śledzenia błędów](#).

Nowe funkcje programu EF Core 2.0

28.08.2018 • 16 minutes to read • [Edit Online](#)

.NET standard 2.0

EF Core teraz jest przeznaczony dla .NET Standard 2.0, co oznacza, że można pracować przy użyciu platformy .NET Core 2.0, .NET Framework 4.6.1 i innych bibliotek, które implementują .NET Standard 2.0. Zobacz [obsługiwane implementacje platformy .NET](#) więcej informacji o tym, co jest obsługiwane.

Modelowanie

Dzielenie tabeli

Teraz jest możliwa do mapowania dwóch lub więcej typów jednostek do tej samej tabeli, gdzie zostaną udostępnione kolumny klucza podstawowego, a każdy wiersz odpowiada dwóch lub więcej podmiotów.

Można użyć tabeli podział identyfikującą relację (gdzie właściwości klucza obcego tworzą klucz podstawowy) musi być skonfigurowany między wszystkie typy jednostek, udostępnianie w tabeli:

```
modelBuilder.Entity<Product>()
    .HasOne(e => e.Details).WithOne(e => e.Product)
    .HasForeignKey<ProductDetails>(e => e.Id);
modelBuilder.Entity<Product>().ToTable("Products");
modelBuilder.Entity<ProductDetails>().ToTable("Products");
```

Typy należące do firmy

Typ jednostki należące do firmy można udostępnić tego samego typu CLR z innym typem jednostki należące do firmy, ale ponieważ nie można zidentyfikować przez typ CLR musi istnieć nawigację do niego z innego typu jednostki. Obiekt zawierający definiujące nawigacji jest właścicielem. Podczas wykonywania zapytań dotyczących właściciela należących do typów będą uwzględniane domyślnie.

Zgodnie z Konwencją klucza podstawowego w tle zostanie utworzony dla typu należące do firmy i będzie można zamapować na tej samej tabeli jako właściciel przy użyciu dzielenie tabeli. Dzięki temu Użyj należące do typów podobnie jak złożonych typów są używane w EF6:

```

modelBuilder.Entity<Order>().OwnsOne(p => p.OrderDetails, cb =>
{
    cb.OwnsOne(c => c.BillingAddress);
    cb.OwnsOne(c => c.ShippingAddress);
});

public class Order
{
    public int Id { get; set; }
    public OrderDetails OrderDetails { get; set; }
}

public class OrderDetails
{
    public StreetAddress BillingAddress { get; set; }
    public StreetAddress ShippingAddress { get; set; }
}

public class StreetAddress
{
    public string Street { get; set; }
    public string City { get; set; }
}

```

Odczyt sekcji posiadane typy jednostek Aby uzyskać więcej informacji na temat tej funkcji.

Filtr na poziomie modelu kwerendy

EF Core 2.0 zawiera nową funkcję nazywaną filtrami kwerendy na poziomie modelu. Ta funkcja umożliwia predykatów zapytań LINQ (wyrażenia logicznego zwykle są przekazywane do operatora zapytania LINQ w przypadku gdy) należy zdefiniować bezpośrednio na typy jednostek w modelu metadanych (zwykle w OnModelCreating). Takie filtry są automatycznie stosowane do żadnych zapytań LINQ, obejmujące tych typów jednostek, w tym odwołania do właściwości nawigacji odwołanie pośrednio, takie jak przy użyciu Include lub bezpośredniego typów jednostek. Niektóre typowe aplikacje tej funkcji są następujące:

- Usuwanie nietrwałe — typy jednostek definiuje właściwość IsDeleted.
- Wielodostępność — typ jednostki definiuje właściwość identyfikatora dzierżawcy.

Poniżej przedstawiono prosty przykład Demonstrowanie funkcji w przypadku dwóch scenariuszy wymienionych powyżej:

```

public class BloggingContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }
    public DbSet<Post> Posts { get; set; }

    public int TenantId { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Post>().HasQueryFilter(
            p => !p.IsDeleted
            && p.TenantId == this.TenantId );
    }
}

```

Definiujemy filtr na poziomie modelu, który implementuje wielodostępu i usuwania nietrwałego dla wystąpienia elementu Post typu jednostki. Zwróć uwagę na użycie właściwości poziomu wystąpienia typu DbContext: TenantId. Filtry na poziomie modelu będzie używać wartości z wystąpienia poprawny kontekst (czyli wystąpienia kontekstu wykonywanego zapytania).

Filtry mogą być wyłączone dla poszczególnych zapytań LINQ, za pomocą operatora `IgnoreQueryFilters()`.

Ograniczenia

- Nawigacja odwołania nie są dozwolone. Ta funkcja mogą być dodawane w oparciu o opinie.
- Filtry można zdefiniować tylko w katalogu głównym typ jednostki dla hierarchii.

Mapowanie funkcji skalarnej bazy danych

EF Core 2.0 obejmuje ważnym wkładem z [Paul Middleton](#) umożliwiającą mapowania funkcji skalarnej bazy danych do metody zastępczych, dzięki czemu mogą być używane w kwerendach LINQ i przetłumaczone do bazy danych SQL.

Poniżej przedstawiono krótki opis sposobu użycia funkcji:

Zadeklaruj metodę statyczną na Twoje `DbContext` i Adnotuj ją za pomocą `DbFunctionAttribute`:

```
public class BloggingContext : DbContext
{
    [DbFunction]
    public static int PostReadCount(int blogId)
    {
        throw new Exception();
    }
}
```

Metod, takich jak to są automatycznie rejestrowane. Po zarejestrowaniu wywołania metody w zapytaniu programu LINQ mogą być tłumaczone do wywołania funkcji w języku SQL:

```
var query =
    from p in context.Posts
    where BloggingContext.PostReadCount(p.Id) > 5
    select p;
```

Kilka kwestii, które należy zwrócić uwagę:

- Zgodnie z Konwencją Nazwa metody jest używana jako nazwa funkcji (w tym przypadku funkcję zdefiniowaną przez użytkownika) podczas generowania SQL, ale można zastąpić nazwę i schematu podczas rejestracji — metoda
- Obecnie obsługiwane są tylko funkcje skalarne
- Należy utworzyć zamapowanego funkcji w bazie danych. EF Core migracje nie zajmie się jego tworzenia

Konfiguracja typu niezależna kodu pierwszy

W EF6 było możliwe do hermetyzacji kodu konfiguracji pierwszego określonego typu, wynikające z `EntityTypeConfiguration`. W programie EF Core 2.0 możemy ponownie także tego wzorca:

```
class CustomerConfiguration : IEntityTypeConfiguration<Customer>
{
    public void Configure(EntityTypeBuilder<Customer> builder)
    {
        builder.HasKey(c => c.AlternateKey);
        builder.Property(c => c.Name).HasMaxLength(200);
    }
}

...
// OnModelCreating
builder.ApplyConfiguration(new CustomerConfiguration());
```

Wysoka wydajność

Buforowanie typu DbContext

Podstawowy wzorzec przy użyciu programu EF Core w aplikacji ASP.NET Core zwykle obejmuje rejestrowanie niestandardowego typu DbContext w systemie iniekcji zależności i później uzyskanie wystąpienia tego typu za pomocą konstruktora parametrów w kontrolerów. Oznacza to, że nowe wystąpienie klasy kontekstu DbContext jest tworzone dla każdego żądania.

W wersji 2.0 wprowadzamy nowy sposób rejestrowania niestandardowego typu DbContext w wstrzykiwanie zależności, które wprowadza w sposób niewidoczny dla użytkownika pulę wystąpien wielokrotnego użytku DbContext. Aby użyć typu DbContext buforowanie, użyj `AddDbContextPool` zamiast `AddDbContext` podczas rejestracji usługi:

```
services.AddDbContextPool<BlogginContext>(  
    options => options.UseSqlServer(connectionString));
```

Jeśli ta metoda jest używana, w momencie wystąpienia typu DbContext jest wymagany przez kontrolera, że firma Microsoft będzie najpierw sprawdzić, czy wystąpienie dostępne w puli. Po Kończenie znajdujących się przetwarzanie żądań, każdy stan w wystąpieniu jest resetowany, a wystąpienie jest zwracane do puli.

Jest to zachowuje się podobnie jak jak buforowanie połączeń działa w dostawcy ADO.NET i ma tę zaletę zapisywania koszt inicjowania wystąpienia typu DbContext.

Ograniczenia

Nowa metoda wprowadza pewne ograniczenia na co można zrobić `OnConfiguring()` metoda kontekstu DbContext.

WARNING

Należy unikać używania buforowania DbContext, jeśli to Ty masz własne stanie (na przykład pola prywatne) w swojej otrzymanej klasie DbContext, który nie powinien być współużytkowane przez wiele żądań. EF Core tylko spowoduje zresetowanie stanu który zna przed dodaniem wystąpienia typu DbContext do puli.

Zapytania skompilowane jawnie

Jest to drugi zdecydować się na wydajności funkcji oferują korzyści w scenariuszach o dużej skali.

Ręczne lub jawnie skompilowanych zapytanie interfejsy API zostały dostępne w poprzednich wersjach programu EF, a także w składniku LINQ to SQL, aby umożliwić aplikacjom tak, aby może zostać obliczony tylko raz w pamięci podręcznej tłumaczenie zapytań i wykonywane wiele razy.

Chociaż ogólnie rzecz biorąc programu EF Core można automatycznie komplilują i kwerendy oparte na mieszanych reprezentacji wyrażenia zapytania w pamięci podręcznej, mechanizm ten może służyć do uzyskania małych są bardziej wydajne, pomijając obliczania skrótu i przeszukiwania pamięci podręcznej, dzięki czemu aplikacji do korzystania z zapytania już skompilowane, za pośrednictwem wywołania delegata.

```
// Create an explicitly compiled query
private static Func<CustomerContext, int, Customer> _customerById =
    EF.CompileQuery((CustomerContext db, int id) =>
        db.Customers
            .Include(c => c.Address)
            .Single(c => c.Id == id));

// Use the compiled query by invoking it
using (var db = new CustomerContext())
{
    var customer = _customerById(db, 147);
}
```

Śledzenie zmian

Dołącz można śledzić wykres nowych i istniejących jednostek

EF Core obsługuje automatyczne generowanie wartości klucza przy użyciu różnych mechanizmów. Podczas używania tej funkcji, wartość jest generowany, gdy właściwość klucza jest ustawieniem domyślnym CLR — zwykle zero lub wartość null. Oznacza to, że wykres jednostki mogą być przekazywane do `DbContext.Attach` lub `DbSet.Attach` i programem EF Core spowoduje oznaczenie tych obiektów, które mają klucz, który został już ustawiony jako `Unchanged` podczas tych jednostek, które nie mają zestawu kluczy zostaną oznaczone jako `Added`. Ułatwia można dołączyć wykres mieszane nowych i istniejących jednostek, korzystając z wygenerowanych kluczy. `DbContext.Update` i `DbSet.Update` działają w taki sam sposób, z tą różnicą, że jednostki z zestawu kluczy, są oznaczone jako `Modified` zamiast `Unchanged`.

Zapytanie

Ulepszone LINQ tłumaczenia

Umożliwia więcej zapytań pomyślnie wykonać logiką więcej ocenianego w bazie danych (a nie w pamięci), a dane niepotrzebnie pobrane z bazy danych.

Groupjoin — ulepszenia

Ta Praca zwiększa SQL Server, który jest generowany dla sprzężenia grupowane. Sprzężenia grupowane w większości przypadków są wynikiem podzapytaniami właściwości nawigacji opcjonalne.

Interpolacja ciągów w FromSql i ExecuteSqlCommand

C# 6 wprowadzono Interpolacja ciągów funkcja, która umożliwia wyrażeń języka C# do osadzenia bezpośrednio Literaly ciągu, dzięki czemu ładny building ciągów w czasie wykonywania. W programie EF Core 2.0 dodaliśmy specjalne obsługę ciągów interpolowanych do naszych dwa podstawowe interfejsów API, które akceptują pierwotne ciągów SQL: `FromSql` i `ExecuteSqlCommand`. Ta obsługa nowych umożliwia interpolacji ciągu C# ma być używany w sposób "bezpieczne". Oznacza to w sposób zapewniający ochronę przed typowymi pomyłek iniekcji SQL, które mogą wystąpić podczas dynamicznego tworzenia SQL w czasie wykonywania.

Oto przykład:

```
var city = "London";
var contactTitle = "Sales Representative";

using (var context = CreateContext())
{
    context.Set<Customer>()
        .FromSql($@"
            SELECT *
            FROM ""Customers"""
            WHERE ""City"" = {city} AND
                  ""ContactTitle"" = {contactTitle}")
        .ToArray();
}
```

W tym przykładzie istnieją dwie zmienne osadzone w ciągu formatu SQL. EF Core generuje następujące instrukcje SQL:

```
@p0='London' (Size = 4000)
@p1='Sales Representative' (Size = 4000)

SELECT *
FROM ""Customers"""
WHERE ""City"" = @p0
    AND ""ContactTitle"" = @p1
```

EF.Functions.Like()

Dodaliśmy EF. Właściwości funkcji, która może być używany przez programu EF Core lub dostawców do definiowania metod mapowane na funkcje bazy danych lub operatorów, tak, aby te mogą być wywoływane w zapytaniach LINQ. Pierwszy przykład taka metoda jest Like():

```
var aCustomers =
    from c in context.Customers
    where EF.Functions.Like(c.Name, "a%")
    select c;
```

Należy pamiętać, że Like() pochodzi z implementacją w pamięci, które mogą być przydatne podczas pracy w bazie danych w pamięci lub oceny predykatu musi nastąpić po stronie klienta.

Zarządzanie bazą danych

Pluralizacja podłączania do tworzenia szkieletów DbContext

EF Core 2.0 wprowadzono nowy *IPluralizer* usługę, która jest używana do końcówek jednostki wpisz nazwy użytkowników i DbSet nazwy w liczbie mnogiej. Domyślna implementacja jest pusta, więc jest to po prostu podłączania, gdzie osoby zajmujące się łatwo podłączyć w ich własnych pluralizer.

Poniżej przedstawiono, jak to wygląda dla dewelopera, można dołączyć w ich własnych pluralizer:

```

public class MyDesignTimeServices : IDesignTimeServices
{
    public void ConfigureDesignTimeServices(IServiceCollection services)
    {
        services.AddSingleton<IPluralizer, MyPluralizer>();
    }
}

public class MyPluralizer : IPluralizer
{
    public string Pluralize(string name)
    {
        return Inflector.Inflector.Pluralize(name) ?? name;
    }

    public string Singularize(string name)
    {
        return Inflector.Inflector.Singularize(name) ?? name;
    }
}

```

Inne osoby

Przenieś dostawcy ADO.NET SQLite SQLitePCL.raw

To daje nam bardziej niezawodne rozwiązanie w Microsoft.Data.Sqlite dystrybucji natywnych SQLite plików binarnych na różnych platformach.

Tylko jeden dostawca na modelu

Znacznie rozszerzają sposobu interakcji z modelem dostawców i upraszcza sposób działania Konwencji, adnotacji i płynnych interfejsów API za pomocą różnych dostawców.

EF Core 2.0 będą teraz tworzyć inną [IModel](#) dla każdego innego dostawcy używane. Jest to zazwyczaj niewidoczna dla aplikacji. Ma to ułatwione uproszczenia metadanych niskiego poziomu interfejsy API, taki sposób, że dowolny dostęp do *typowe pojęcia relacyjnych metadanych* zawsze jest wykonywane za pomocą wywołania [.Relational](#) zamiast [.SqlServer](#), [.Sqlite](#) itp.

Skonsolidowane rejestrowania i diagnostyki

Rejestrowanie (oparte na [ILogger](#)) i mechanizmy diagnostyki (oparte na [DiagnosticSource](#)) teraz udostępnić więcej kodu.

Identyfikatory zdarzeń dla wiadomości wysyłanych do [ILogger](#) zostały zmienione w wersji 2.0. Identyfikatory zdarzeń obecnie są unikatowe w obrębie kod programem EF Core. Te komunikaty teraz również wykonać standardowego wzorca dla rejestraniem strukturalnym używane przez, na przykład MVC.

Kategorie rejestratora również zostały zmienione. Ma teraz dobrze znanego zestawu kategorii dostępne za pośrednictwem [DbLoggerCategory](#).

Zdarzenia [DiagnosticSource](#) teraz użyć takich samych nazwach identyfikator zdarzenia odpowiadającego [ILogger](#) wiadomości.

Nowe funkcje programu EF Core 1.1

28.08.2018 • 2 minutes to read • [Edit Online](#)

Modelowanie

Mapowanie pól

Umożliwia skonfigurowanie z polem zapasowym dla właściwości. Może to być przydatne w przypadku właściwości tylko do odczytu lub danymi, które mają metod Get/Set, a nie właściwości.

Mapowanie do tabel zoptymalizowanych pod kątem pamięci w programie SQL Server

Można określić, czy jednostka jest zamapowana do tabel jest zoptymalizowane pod kątem pamięci. Podczas tworzenia i bazę danych przy użyciu programu EF Core na podstawie modelu (przy użyciu migracji lub `Database.EnsureCreated()`), zoptymalizowana pod kątem pamięci tabeli zostanie utworzony dla tych jednostek.

Śledzenie zmian

Kolejna zmiana, śledzenie interfejsów API z programu EF6

Takie jak `Reload`, `GetModifiedProperties`, `GetDatabaseValues` itp.

Zapytanie

Jawne ładowanie

Służy do wyzwalania populacji właściwości nawigacji jednostki, która została wcześniej załadowana z bazy danych.

DbSet.Find

Zapewnia łatwy sposób pobrać jednostki, w oparciu o jego wartość klucza podstawowego.

Inne

Elastyczność połączenia

Automatycznie ponownych prób nie powiodło się polecenia bazy danych. Jest to szczególnie przydatne podczas połączenia z platformą Azure SQL, w których typowych błędów przejściowych.

Uproszczona wymienna

Ułatwia Zastąp wewnętrznych usług, których używa EF.

Funkcje zawarte w programie EF Core 1.0

28.08.2018 • 7 minutes to read • [Edit Online](#)

Platformy

.NET Framework 4.5.1

Obejmuje konsolę, WPF, WinForms, platformy ASP.NET 4, itd.

.NET standard 1.3

W tym platformy ASP.NET Core przeznaczone dla .NET Framework i .NET Core w Windows, OS x i Linux.

Modelowanie

Podstawowego modelowania

Oparte na jednostkach POCO za pomocą właściwości get/set popularnych typów skalarnych (`int`, `string` itp.).

Relacje i właściwości nawigacji

W modelu, w oparciu o klucz obcy można określić jeden do wielu i relacje jeden do zero lub jeden. Właściwości nawigacji typów prostych kolekcji lub odwołania może być skojarzony z tych relacji.

Konwencje wbudowane

Te tworzenie początkowej model oparty na kształcie klas jednostek.

Interfejs Fluent API

Zezwala na zastąpienie `OnModelCreating` metody na kontekst dodatkowo skonfigurować modelu, która została wykryta przez Konwencję.

Adnotacje danych

To atrybutów, można dodać do swojej klasy/właściwości jednostki, które wpływają na modelu platformy EF. Na przykład dodanie `[Required]` umożliwi EF wiedzieć, że właściwość jest wymagana.

Relacyjne Mapowanie tabeli

Umożliwia jednostek, które mają być mapowane do tabele/kolumny.

Generowanie wartości klucza

W tym generowania po stronie klienta i generowanie bazy danych.

Baza danych generowane wartości

Zezwala na wartości, które mają być generowane przez bazę danych na wstawiania (wartości domyślne) lub aktualizacji (kolumn obliczanych).

Sekwencje w programie SQL Server

Umożliwia sekwencji obiektów zdefiniowanych w modelu.

Unikatowych ograniczeń

Umożliwia określenie klucze alternatywne i możliwość definiowania relacji przeznaczonych tego klucza.

Indeksy

Definiowanie indeksów w modelu automatycznie wprowadza indeksy w bazie danych. Unikatowe indeksy są również obsługiwane.

Właściwości stanu w tle

Umożliwia właściwości zdefiniowanych w modelu, które nie są deklarowane i nie są przechowywane w klasie .NET, ale mogą być śledzone i aktualizowane przez platformę EF Core. Często używane do właściwości klucza obcego, gdy udostępnianie w obiekcie nie jest wymagane.

Tabela wg hierarchii dziedziczenia wzorzec

Umożliwia jednostek w hierarchii dziedziczenia do zapisania pojedynczej tabeli, aby zidentyfikować ich typ jednostki dla danego rekordu w bazie danych przy użyciu kolumny dyskryminatora.

Walidacja modelu

Wykrywa nieprawidłowe wzorców w modelu i zawiera komunikaty o błędach pomocne.

Śledzenie zmian

Śledzenie zmian migawki

Umożliwia zmiany w jednostkach, aby zostało wykryte automatycznie przez porównanie bieżącego stanu pierwotnego stanu kopii (Migawka).

Śledzenie zmian powiadomień

Umożliwia jednostek do powiadamiania śledzenie zmian wartości właściwości są modyfikowane.

Uzyskiwanie dostępu do śledzonych stanu

Za pomocą `DbContext.Entry` i `DbContext.ChangeTracker`.

Dołączanie odłączonych jednostek/wykresów

Nowy `DbContext.AttachGraph` interfejs API pomaga ponownie podłączyć jednostkę do kontekstu w celu zapisania nowych/zmodyfikowanych jednostek.

Zapisywanie danych

Podstawowa funkcja zapisywania

Umożliwia zmiany wystąpień jednostek w celu jego utrwalenia w bazie danych.

Optymistyczna współprzeźność

Chroni przed zastępowaniem zmian wprowadzonych przez innego użytkownika, ponieważ dane została pobrana z bazy danych.

Async SaveChanges

Zwolnić bieżącego wątku przetwarzanie innych żądań, gdy baza danych przetwarza poleceń wydawanych z `SaveChanges`.

Transakcje bazy danych

Oznacza, że `SaveChanges` jest zawsze niepodzielne (tzn. jej albo całkowicie zakończy się pomyślnie, lub są wprowadzane żadne zmiany w bazie danych). Istnieją również transakcji powiązanych interfejsów API, aby zezwolić na udostępnianie transakcji między wystąpieniami kontekstu itp.

Relacyjne: Przetwarzanie wsadowe instrukcji

Zapewnia lepszą wydajność, przetwarzanie wsadowe się wielu poleceń WSTAWIANIA/AKTUALIZOWANIA/usuwania do pojedynczego obie strony do bazy danych.

Zapytanie

Podstawowa pomoc techniczna LINQ

Zapewnia możliwość korzystania z LINQ do pobierania danych z bazy danych.

Ocena mieszane klient serwer

Umożliwia zapytaniom zawiera logikę, która nie może być ocenione w bazie danych, a w związku z tym muszą być oceniane, po pobraniu danych do pamięci.

NoTracking

Zapytania umożliwia szybsze wykonywanie zapytania, gdy kontekst nie jest konieczne monitorowanie zmian w wystąpieniu jednostek (jest to przydatne, gdy wyniki są tylko do odczytu).

Wczesne ładowanie

Udostępnia `Include` i `ThenInclude` metodami do identyfikowania powiązanych danych, które również mają być pobierane, podczas wykonywania zapytania.

Zapytania asynchroniczne

Można zwolnić miejsce na bieżącego wątku (i jego skojarzone zasoby) do przetworzenia inne żądania, gdy baza danych przetwarza zapytania.

Pierwotne zapytania SQL

Udostępnia `DbSet.FromSql` metodę SQL pierwotne zapytania można pobrać danych. Te zapytania może składać się także na temat korzystania z LINQ.

Zarządzanie schematami bazy danych

Baza danych tworzenia/usuwania interfejsów API

Przede wszystkim są przeznaczone do testowania, które chcesz szybko tworzyć/usuwać bazy danych bez używania migracji.

Migracje relacyjnej bazy danych

Zezwala na schemat relacyjnej bazy danych w rozwój nadgodzinach jako zmiany modelu.

Odtwarzanie z bazy danych

Szkielety mechanizmów modelu platformy EF oparte na istniejący schemat relacyjnej bazy danych.

Dostawcy baz danych

SQL Server

Łączy się z programu Microsoft SQL Server 2008 lub nowszym.

Bazy danych SQLite

Nawiązuje połączenie z bazą danych SQLite 3.

W pamięci

Jest umożliwiających łatwe testowanie bez połączenia z prawdziwą bazą danych.

3 dostawców

Kilku dostawców są dostępne dla innych baz danych. Zobacz [dostawcy baz danych](#) pełną listę.

Uaktualnianie z programu EF Core 1.0 RC1 do wersji 1.0 RC2

28.08.2018 • 8 minutes to read • [Edit Online](#)

Ten artykuł zawiera wskazówki dotyczące przenoszenia aplikacji skompilowanej za pomocą pakietów RC1 do RC2.

Nazwy pakietów i wersje

Między RC1 i RC2 firma Microsoft zmieniony z "Entity Framework 7" na "Entity Framework Core". Możesz dowiedzieć się więcej o ze względu na zmianę w [ten wpis, Scott hanselman](#). Ze względu na tę zmianę nazw naszych pakietów zmieniła się z `EntityFramework.*` do `Microsoft.EntityFrameworkCore.*` i nasze w wersjach od `7.0.0-rc1-final` do `1.0.0-rc2-final` (lub `1.0.0-preview1-final` narzędzi).

Konieczne będzie całkowicie usunąć pakiety RC1, a następnie zainstaluj RC2 te. Poniżej przedstawiono mapowanie niektórych popularnych pakietów innych.

PAKIEĆ RC1	ODPOWIEDNIK RC2
<code>EntityFramework.MicrosoftSqlServer 7.0.0-rc1-final</code>	<code>Microsoft.EntityFrameworkCore.SqlServer 1.0.0-rc2-final</code>
<code>EntityFramework.SQLite 7.0.0-rc1-final</code>	<code>Microsoft.EntityFrameworkCore.Sqlite 1.0.0-rc2-final</code>
<code>EntityFramework7.Npgsql 3.1.0-rc1-3</code>	<code>Npgsql.EntityFrameworkCore.Postgres</code>
<code>EntityFramework.SqlServerCompact35 7.0.0-rc1-final</code>	<code>EntityFrameworkCore.SqlServerCompact35 1.0.0-rc2-final</code>
<code>EntityFramework.SqlServerCompact40 7.0.0-rc1-final</code>	<code>EntityFrameworkCore.SqlServerCompact40 1.0.0-rc2-final</code>
<code>EntityFramework.InMemory 7.0.0-rc1-final</code>	<code>Microsoft.EntityFrameworkCore.InMemory 1.0.0-rc2-final</code>
<code>EntityFramework.IBMDriver 7.0.0-beta1</code>	Nie jest jeszcze dostępna RC2
<code>EntityFramework.Commands 7.0.0-rc1-final</code>	<code>Microsoft.EntityFrameworkCore.Tools 1.0.0-preview1-final</code>
<code>EntityFramework.MicrosoftSqlServer.Design 7.0.0-rc1-final</code>	<code>Microsoft.EntityFrameworkCore.SqlServer.Design 1.0.0-rc2-final</code>

Namespaces

Wraz z nazwy pakietu, przestrzenie nazw zmieniła się z `Microsoft.Data.Entity.*` do `Microsoft.EntityFrameworkCore.*`. Może obsługiwać tej zmiany z Znajdź/Zamień z `using Microsoft.Data.Entity` z `using Microsoft.EntityFrameworkCore`.

Tabela zmian konwencji nazewnictwa

Znaczące zmiany funkcjonalne w wersji RC2, skorzystaliśmy było użyć nazwy `DbSet< TEntity >` właściwości dla danej jednostki, jako nazwę tabeli mapowania, a nie tylko nazwę klasy. Możesz dowiedzieć się więcej o tej zmianie w [problem powiązane ogłoszenie](#).

W przypadku istniejących aplikacji RC1, firma Microsoft zaleca, dodając następujący kod na początku swoje `OnModelCreating` metodę, aby zachować strategię nazewnictwa RC1:

```
foreach (var entity in modelBuilder.Model.GetEntityTypes())
{
    entity.Relational().TableName = entity.DisplayName();
}
```

Jeśli chcesz, który wdrożył nowy strategię nazewnictwa, czy zalecane jest pomyślnie Kończenie pozostałe kroki aktualnienia, a następnie usunięcie kodu i tworzenia migracji do zastosowania w tabeli zmienia nazwę.

AddDbContext / Startup.cs zmian (tylko w przypadku projektów ASP.NET Core)

W wersji RC1, trzeba było dodać usługi Entity Framework do dostawcy usług w aplikacji — w

```
Startup ConfigureServices(...):
```

```
services.AddEntityFramework()
    .AddSqlServer()
    .AddDbContext<ApplicationContext>(options =>
        options.UseSqlServer(Configuration["ConnectionStrings:DefaultConnection"]));

```

W wersji RC2, możesz usunąć wywołania `AddEntityFramework()`, `AddSqlServer()` itp.:

```
services.AddDbContext<ApplicationContext>(options =>
    options.UseSqlServer(Configuration["ConnectionStrings:DefaultConnection"]));

```

Musisz również dodać Konstruktor pochodnej kontekst, który przyjmuje opcje kontekstu i przekazuje je do konstruktora podstawowego. Jest to niezbędne, ponieważ firma Microsoft usunęła niektóre scary magic, który snuck je w tle:

```
public ApplicationContext(DbContextOptions<ApplicationContext> options)
    : base(options)
{}
```

Przekazywanie w IServiceProvider.

Jeśli masz kod RC1, który przekazuje `IServiceProvider` kontekst, to teraz przeniesiono do `DbContextOptions`, a nie parametrem oddzielnego konstruktora. Użyj `DbContextOptionsBuilder.UseInternalServiceProvider(...)` można ustawić dostawcę usług.

Testowanie

Najbardziej typowym scenariuszem tego zrobić był umożliwiał kontrolowanie zakresu InMemory bazy danych, podczas testowania. Zobacz zaktualizowanego [testowania](#) artykułu, na przykład w ten sposób za pomocą RC2.

Rozpoznawanie wewnętrznych usług od dostawcy usług w aplikacji (tylko w przypadku projektów ASP.NET Core)

Aplikacja ASP.NET Core, i chcesz EF, aby rozwiązać wewnętrznych usług od dostawcy usług w aplikacji, czy przeciążenia `AddDbContext` pozwala skonfigurować to:

```
services.AddEntityFrameworkSqlServer()
    .AddDbContext<ApplicationContext>((serviceProvider, options) =>
    options.UseSqlServer(Configuration["ConnectionStrings:DefaultConnection"])
        .UseInternalServiceProvider(serviceProvider)); );
```

WARNING

Firma Microsoft zaleca, dzięki czemu EF wewnętrznie Zarządzanie własnych usług, chyba że masz powód, aby połączyć wewnętrznych usług EF w aplikacji dostawcy usługi. Głównym powodem, czy chcesz to zrobić jest Zastąp usług, które jest używane wewnętrznie EF za pomocą dostawcy usługi aplikacji

Poleceń środowiska DNX = > .NET interfejsu wiersza polecenia (tylko w przypadku projektów ASP.NET Core)

Jeśli wcześniej używano `dnx ef` poleceń dla projektów ASP.NET 5, te zostały przeniesione do `dotnet ef` poleceń. Tej samej składni polecenia nadal obowiązuje ograniczenie. Możesz użyć `dotnet ef --help` Aby uzyskać informacje o składni.

Sposób, w jaki są rejestrowane polecenia został zmieniony w wersji RC2 z powodu środowiska DNX, jest zastępowany przez interfejs wiersza polecenia platformy .NET. Polecenia są teraz zarejestrowane w usłudze `tools` sekcji `project.json`:

```
"tools": {
  "Microsoft.EntityFrameworkCore.Tools": {
    "version": "1.0.0-preview1-final",
    "imports": [
      "portable-net45+win8+dnxcore50",
      "portable-net45+win8"
    ]
  }
}
```

TIP

Jeśli używasz programu Visual Studio, możesz teraz używać Konsola Menedżera pakietów do uruchamiania polecień EF dla projektów ASP.NET Core (to zostało nie obsługiwane w wersji RC1). Nadal trzeba zarejestrować polecień w `tools` części `project.json` w tym celu.

Menedżer pakietów polecenia wymagają programu PowerShell 5

Jeśli używasz polecień programu Entity Framework w konsoli Menedżera pakietów w programie Visual Studio, następnie należy upewnić się, że masz zainstalowanego 5 programu PowerShell. Jest to wymagane tymczasowe, która zostanie usunięta w następnej wersji (zobacz [wystawiać #5327](#) Aby uzyskać więcej informacji).

Za pomocą "import" w pliku project.json

Niektóre z programem EF Core zależności nie obsługują .NET Standard jeszcze. EF Core w projektach .NET Standard i .NET Core może wymagać, dodając "imports" do pliku project.json jako rozwiązanie tymczasowe.

Podczas dodawania EF, przywracanie pakietów NuGet spowoduje wyświetlenie tego komunikatu o błędzie:

```
Package Ix-Async 1.2.5 is not compatible with netcoreapp1.0 (.NETCoreApp,Version=v1.0). Package Ix-Async 1.2.5
supports:
- net40 (.NETFramework,Version=v4.0)
- net45 (.NETFramework,Version=v4.5)
- portable-net45+win8+wp8 (.NETPortable,Version=v0.0,Profile=Profile78)
Package Remotion.Linq 2.0.2 is not compatible with netcoreapp1.0 (.NETCoreApp,Version=v1.0). Package
Remotion.Linq 2.0.2 supports:
- net35 (.NETFramework,Version=v3.5)
- net40 (.NETFramework,Version=v4.0)
- net45 (.NETFramework,Version=v4.5)
- portable-net45+win8+wp8+wpa81 (.NETPortable,Version=v0.0,Profile=Profile259)
```

Obejście polega na ręcznie zainportować przenośnej profilu "portable net451 + win8". Tego pakietu NuGet, aby traktować to pliki binarne, które odpowiadają tym wymusza działają jako środowisko zgodne z technologią .NET Standard, nawet jeśli nie są one. Mimo że "portable net451 + win8" nie jest zgodny z .NET Standard w 100%, są one zgodne, przejścia od PCL .NET Standard. Importy można usunąć zależności EF firmy po pewnym czasie uaktualnić do wersji .NET Standard.

Do "import" można dodać wielu platform przy użyciu składni tablicy. Pozostałe importy może być konieczne, jeśli dodatkowe biblioteki zostaną dodane do projektu.

```
{
  "frameworks": {
    "netcoreapp1.0": {
      "imports": [ "dnxcore50", "portable-net451+win8" ]
    }
  }
}
```

Zobacz [wystawiać #5176](#).

Uaktualnianie z programu EF Core 1.0 RC2 do RTM

28.08.2018 • 4 minutes to read • [Edit Online](#)

Ten artykuł zawiera wskazówki dotyczące przenoszenia aplikacji skompilowanej za pomocą pakietów 1.0.0 RC2 RTM.

Wersje pakietów

Nazwy pakietów najwyższego poziomu, które zazwyczaj można zainstalować na aplikację nie zmieniły się od wersji RC2 i RTM.

Należy uaktualnić zainstalowane pakiety do wersji RTM:

- Pakiety środowiska uruchomieniowego (na przykład `Microsoft.EntityFrameworkCore.SqlServer`) zmieniła się z `1.0.0-rc2-final` do `1.0.0`.
- `Microsoft.EntityFrameworkCore.Tools` Pakietu zmieniła się z `1.0.0-preview1-final` do `1.0.0-preview2-final`. Należy pamiętać, że narzędzia jest nadal w wersji wstępnej.

Migracja istniejących może być konieczne maxLength dodane

W wersji RC2, definicji kolumny, w przypadku migracji wyglądał jak `table.Column<string>(nullable: true)` i długość kolumny została wyszukiwana w niektóre metadane są przechowywane w kodzie migracji. W wersji RTM, długość jest teraz zawarta w utworzony szkielet kodu `table.Column<string>(maxLength: 450, nullable: true)`.

Istniejące migracji, które zostały szkieletu przed użyciem RTM nie będzie miał `maxLength` określony argument. Oznacza to, maksymalna długość obsługiwane przez bazę danych, który będzie używany (`nvarchar(max)` w programie SQL Server). Może to być dobrym rozwiążaniem dla niektórych kolumn, ale także kolumny, które są częścią klucza, klucz obcy lub indeksu, należy zaktualizować maksymalną długość. Zgodnie z Konwencją 450 jest maksymalna długość używane do kluczy, kluczy obcych i indeksowanych kolumn. Jeśli długość skonfigurowano jawnie w modelu, następnie należy użyć tej długości zamiast tego.

ASP.NET Identity

Ta zmiana ma wpływ na projekty użycia produktu ASP.NET Identity, które zostały utworzone na podstawie pre-RTM szablonu projektu. Szablon projektu obejmuje migrację użyty do utworzenia bazy danych. Ta migracja musi być edytowany, aby określić maksymalną długość `256` dla następujących kolumn.

• AspNetRoles

- Nazwa
- NormalizedName

• AspNetUsers

- Adres e-mail
- NormalizedEmail
- NormalizedUserName
- UserName

Nie można dokonać tej zmiany spowoduje następujący wyjątek podczas początkowej migracji jest stosowana do

bazy danych.

```
System.Data.SqlClient.SqlException (0x80131904): Column 'Normalized Name' in table 'AspNetRoles' is of a type  
that is invalid for use as a key column in an index.
```

.NET core: Usuń "import" w pliku project.json

Jeśli zostały przeznaczone dla platformy .NET Core za pomocą RC2, trzeba było dodać `imports` do pliku `project.json` jako rozwiązanie tymczasowe dla niektórych zależności programu EF Core nie obsługuje .NET Standard. Te można teraz usunąć.

```
{  
  "frameworks": {  
    "netcoreapp1.0": {  
      "imports": [ "dnxcore50", "portable-net451+win8" ]  
    }  
  }  
}
```

NOTE

Począwszy od wersji 1.0 RTM, zestawu [.NET Core SDK](#) nie obsługuje już `project.json` lub tworzenia aplikacji platformy .NET Core przy użyciu programu Visual Studio 2015. Firma Microsoft zaleca [migracji z plików project.json do csproj](#). Jeśli używasz programu Visual Studio, zaleca się uaktualnienie do [programu Visual Studio 2017](#).

Platformy uniwersalnej systemu Windows: Dodać przekierowania powiązań

Przy próbie uruchomienia programu EF poleceń na wynikach projektów platformy uniwersalnej Windows (UWP) w następujący błąd:

```
System.IO.FileLoadException: Could not load file or assembly 'System.IO.FileSystem.Primitives, Version=4.0.0.0,  
Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a' or one of its dependencies. The located assembly's manifest  
definition does not match the assembly reference.
```

Należy ręcznie dodać przekierowania powiązań do projektu platformy uniwersalnej systemu Windows. Utwórz plik o nazwie `App.config` w projekcie folder główny i dodać przekierowania do wersji poprawny zestaw.

```
<configuration>
  <runtime>
    <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
      <dependentAssembly>
        <assemblyIdentity name="System.IO.FileSystem.Primitives"
                          publicKeyToken="b03f5f7f11d50a3a"
                          culture="neutral" />
        <bindingRedirect oldVersion="4.0.0.0"
                         newVersion="4.0.1.0"/>
      </dependentAssembly>
      <dependentAssembly>
        <assemblyIdentity name="System.Threading.Overlapped"
                          publicKeyToken="b03f5f7f11d50a3a"
                          culture="neutral" />
        <bindingRedirect oldVersion="4.0.0.0"
                         newVersion="4.0.1.0"/>
      </dependentAssembly>
    </assemblyBinding>
  </runtime>
</configuration>
```

Uaktualnianie aplikacji z poprzedniej wersji do programu EF Core 2.0

27.09.2018 • 14 minutes to read • [Edit Online](#)

Podejmujemy szansy sprzedaży, aby znacznie udoskonalić naszych istniejących interfejsów API i zachować w wersji 2.0. Istnieje kilka ulepszeń, które mogą wymagać od modyfikowania istniejącego kodu aplikacji, ale uważamy, że dla większości aplikacji wpływ będzie niska, w większości przypadków wymaga podania tylko ponownej komplikacji i minimalnych zmianach z przewodnikiem, aby zastąpić przestarzałe interfejsy API.

Aktualizowanie istniejącej aplikacji do programu EF Core 2.0 mogą wymagać:

1. Uaktualnianie wdrożenia .NET docelowej aplikacji obsługującej .NET Standard 2.0. Zobacz [obsługiwane implementacje platformy .NET](#) Aby uzyskać więcej informacji.
2. Określ dostawcę dla docelowej bazy danych, które są zgodne z programem EF Core 2.0. Zobacz [EF Core 2.0 wymaga dostawcy bazy danych 2.0](#) poniżej.
3. Uaktualnienie wszystkich pakietów programu EF Core (środowiska uruchomieniowego i narzędzi) do wersji 2.0. Zapoznaj się [Instalowanie programu EF Core](#) Aby uzyskać więcej informacji.
4. Wprowadź wszelkie zmiany niezbędny kod w celu kompensacji przełomowe zmiany opisane w dalszej części tego dokumentu.

Platforma ASP.NET Core zawiera teraz EF Core

Aplikacje przeznaczone na platformy ASP.NET Core 2.0 mogą używać programu EF Core 2.0 bez dodatkowe zależności oprócz dostawcy bazy danych. Jednak aplikacje przeznaczone na poprzednie wersje platformy ASP.NET Core konieczne uaktualnienie do programu ASP.NET Core 2.0, aby można było używać programu EF Core 2.0.

Aby uzyskać więcej informacji na temat uaktualniania aplikacji platformy ASP.NET Core 2.0, zobacz [dokumentacji platformy ASP.NET Core na ten temat](#).

Nowy sposób, w jakim usługi aplikacji w programie ASP.NET Core

Zaktualizowano zalecanego wzorca dla aplikacji sieci web platformy ASP.NET Core 2.0 w taki sposób, że Przerwano logiki czasu projektowania, jakie programu EF Core w 1.x. Wcześniej w czasie projektowania programu EF Core będzie spróbować ponownie wywołać `Startup.ConfigureServices` bezpośrednio po to, aby uzyskać dostęp do aplikacji dostawcy usług. W programie ASP.NET Core 2.0 konfiguracji jest inicjowany poza `Startup` klasy.

Aplikacje zazwyczaj przy użyciu programu EF Core uzyskiwały dostęp do swoich parametrów połączenia z konfiguracji, więc `Startup` przez siebie nie jest już wystarczająca. Jeśli zaktualizujesz aplikację ASP.NET Core 1.x, otrzymasz następujący błąd podczas korzystania z narzędzi programu EF Core.

Żaden konstruktor bez parametrów nie został odnaleziony w "ApplicationContext". Dodaj konstruktor bez parametrów do "ApplicationContext" lub Dodaj implementację "IDesignTimeDbContextFactory<ApplicationContext>" z tego samego zestawu jako "ApplicationContext"

Nowe hook czasu projektowania został dodany w szablonie domyślnego programu ASP.NET Core 2.0. Statyczne `Program.BuildWebHost` metoda umożliwia programu EF Core dostępu do aplikacji usługi dostawcy w czasie projektowania. Jeśli aktualniasz aplikacji ASP.NET Core 1.x należy zaktualizować `Program` klasy na podobny do następującego.

```

using Microsoft.AspNetCore;
using Microsoft.AspNetCore.Hosting;

namespace AspNetCoreDotNetCore2._0App
{
    public class Program
    {
        public static void Main(string[] args)
        {
            BuildWebHost(args).Run();
        }

        public static IWebHost BuildWebHost(string[] args) =>
            WebHost.CreateDefaultBuilder(args)
                .UseStartup<Startup>()
                .Build();
    }
}

```

Przyjęcie tego nowego wzorca podczas aktualizowania aplikacji w wersji 2.0 zdecydowanie zaleca się i jest wymagane dla funkcji produktów, takich jak migracje Entity Framework Core pracować. Powszechną alternatywą jest [zaimplementować `IDesignTimeDbContextFactory<TContext>`](#).

Zmieniono nazwę `IDbContextFactory`

W celu obsługi wzorców dla różnych aplikacji i udostępniać użytkownikom większą kontrolę nad jak ich `DbContext` jest używany w czasie projektowania, mamy, w przeszłości, pod warunkiem `IDbContextFactory<TContext>` interfejsu. W czasie projektowania, narzędzia programu EF Core odnajdzie implementacje tego interfejsu w projekcie i użyć go do utworzenia `DbContext` obiektów.

Ten interfejs ma bardzo ogólne nazwy, która wprowadzać w błąd niektórych użytkowników, aby spróbować ponownie go dla innych `DbContext` — tworzenie scenariuszy. Były one przechwytywane wyłączona ochrona, gdy narzędzia EF następnie dotarła do ich wykonania w czasie projektowania i spowodowane polecień, takich jak `Update-Database` lub `dotnet ef database update` nie powiedzie się.

Aby komunikować się z silną semantyką czasu projektowania tego interfejsu, zmieniono jego `IDesignTimeDbContextFactory<TContext>`.

Dla wersji 2.0 wersji `IDbContextFactory<TContext>` nadal istnieje, ale jest oznaczony jako przestarzały.

`DbContextFactoryOptions` usunięte

Ze względu na zmiany ASP.NET Core 2.0, opisane powyżej, Odkryliśmy, że `DbContextFactoryOptions` został nie będą już potrzebne w nowym `IDesignTimeDbContextFactory<TContext>` interfejsu. Poniżej przedstawiono rozwiązań alternatywnych, z których powinien być używany zamiast tego.

DBCONTEXTFACTORYOPTIONS	ALTERNATYWNA
ApplicationBasePath	<code>ApplicationContext.BaseDirectory</code>
ContentRootPath	<code>Directory.GetCurrentDirectory()</code>
EnvironmentName	<code>Environment.GetEnvironmentVariable("ASPNETCORE_ENVIRONMENT")</code>

Katalog roboczy czasu projektowania został zmieniony

Katalog roboczy używany przez również wymaganych zmian platformy ASP.NET Core 2.0 `dotnet ef` wyrównać katalog roboczy używany przez program Visual Studio, podczas uruchamiania aplikacji. Jeden dostrzegalnych efekt uboczny tej jest tej bazy danych SQLite nazwy plików są teraz względem katalogu projektu, a nie katalog wyjściowy jak kiedyś.

EF Core 2.0 wymaga dostawcy 2.0 bazy danych

EF Core 2.0 wprowadzono wiele definiowania i ulepszeń w dostawcy baz danych w sposób pracy. Oznacza to, że dostawcy 1.0.x i 1.1.x, nie będą działać przy użyciu programu EF Core 2.0.

Dostawcy programu SQL Server i bazy danych SQLite są dostarczane przez zespół programu EF i wersji 2.0 będzie dostępna w wersji 2.0 w ramach wersji. Dostawcy typu open-source innych firm dla [SQL Compact](#), [PostgreSQL](#), i [MySQL](#) są aktualizowane dla wersji 2.0. W przypadku innych dostawców, skontaktuj się z modułu zapisującego dostawcy.

Zmieniono zdarzenia rejestrowania i diagnostyki

Uwaga: te zmiany nie powinien wpływać na większość kodu aplikacji.

Komunikaty wysyłane do identyfikatorów zdarzeń [ILogger](#) zostały zmienione w wersji 2.0. Identyfikatory zdarzeń obecnie są unikatowe w obrębie kod programem EF Core. Te komunikaty teraz również wykonać standardowego wzorca dla rejestraniem strukturalnym używane przez, na przykład MVC.

Kategorie rejestratora również zostały zmienione. Ma teraz dobrze znanego zestawu kategorii dostępne za pośrednictwem [DbLoggerCategory](#).

[DiagnosticSource](#) zdarzenia teraz użyć takich samych nazwach identyfikator zdarzenia odpowiadającego [ILogger](#) wiadomości. Ładunki zdarzeń są wszystkie typy nominalna pochodną [EventData](#).

Identyfikatory zdarzeń, typy ładunku i kategorie są udokumentowane w [CoreEventId](#) i [RelationalEventId](#) klasy.

Identyfikatory również zostały przeniesione z `Microsoft.EntityFrameworkCore.Infrastructure` do nowej przestrzeni nazw `Microsoft.EntityFrameworkCore.Diagnostics`.

EF Core relacyjnych metadanych interfejsu API zmiany

EF Core 2.0 będą teraz tworzyć inną [IModel](#) dla każdego innego dostawcy używane. Jest to zazwyczaj niewidoczna dla aplikacji. Ma to ułatwione uproszczenia metadanych niskiego poziomu interfejsy API, taki sposób, że dowolny dostęp do *typowe pojęcia relacyjnych metadanych* zawsze jest wykonywane za pomocą wywołania `.Relational` zamiast `.SqlServer`, `.Sqlite` itp. Na przykład 1.1.x kod następująco:

```
var tableName = context.Model.FindEntityType(typeof(User)).SqlServer().TableName;
```

Powinny teraz być zapisany jako:

```
var tableName = context.Model.FindEntityType(typeof(User)).Relational().TableName;
```

Zamiast używać metod, takich jak `ForSqlServerToTable`, metody rozszerzające są teraz dostępne do zapisu kod warunkowy, w oparciu o bieżący Dostawca używany. Na przykład:

```
modelBuilder.Entity<User>().ToTable(
    Database.IsSqlServer() ? "SqlServerName" : "OtherName");
```

Należy zauważać, że ta zmiana ma zastosowanie tylko do interfejsów API/metadanych, który jest zdefiniowany dla

wszystkich relacyjnych dostawców. Interfejs API i metadanych pozostaje taki sam, gdy jest ona specyficzne dla tylko jednego dostawcę. Na przykład klastrowanych indeksów są specyficzne dla SQL Server, dzięki czemu

`ForSqlServerIsClustered` i `.SqlServer().IsClustered()` nadal muszą być używane.

Nie przejąć kontrolę nad EF dostawcy usług

EF Core używa wewnętrznego `IServiceProvider` (kontener iniekcji zależności) do swojej wewnętrznej implementacji. Aplikacje powinny zezwalać programu EF Core utworzyć i zarządzać tego dostawcy, z wyjątkiem w szczególnych przypadkach. Zdecydowanie rozważyć usunięcie wszelkie wywołania `UseInternalServiceProvider`. Jeśli aplikacja musi wywołać `UseInternalServiceProvider`, należy rozważyć [rejestrując problem](#), dzięki czemu firma Microsoft można zbadać inne sposoby obsługi danego scenariusza.

Wywoływanie `AddEntityFramework`, `AddEntityFrameworkSqlServer`, itp. nie jest wymagane przez kod aplikacji, chyba że `UseInternalServiceProvider` skrót. Usuń wszystkie istniejące wywołania `AddEntityFramework` lub `AddEntityFrameworkSqlServer` itd `AddDbContext` należy nadal używać w taki sam sposób jak wcześniej.

Musi mieć nazwę bazy danych w pamięci

Globalne bez nazwy bazy danych w pamięci została usunięta, a zamiast tego muszą nosić wszystkich baz danych w pamięci. Na przykład:

```
optionsBuilder.UseInMemoryDatabase("MyDatabase");
```

To tworzy/używa bazy danych o nazwie "Moja_baza_danych". Jeśli `UseInMemoryDatabase` wywoływana jest ponownie o takiej samej nazwie tej samej bazy danych w pamięci zostanie użyta, dzięki któremu być współużytkowane przez wiele wystąpień kontekstu.

Zmiany interfejsu API tylko do odczytu

`IsReadOnlyBeforeSave`, `IsReadOnlyAfterSave`, i `IsStoreGeneratedAlways` zdezaktualizowane i zastąpione `BeforeSaveBehavior` i `AfterSaveBehavior`. Te zachowania mają zastosowanie do wszystkich właściwości (nie tylko generowane przez Magazyn właściwości) i określić, jak wartość właściwości powinna być używana w przypadku wstawiania do wiersza bazy danych (`BeforeSaveBehavior`) lub podczas aktualizowania istniejącej bazy danych wiersza (`AfterSaveBehavior`).

Właściwości są oznaczone jako `ValueGenerated.OnAddOrUpdate` (na przykład w przypadku kolumn obliczanych) domyślnie zignoruje dowolną wartość aktualnie ustawiona we właściwości. Oznacza to, że wygenerowana przez Magazyn wartość, zawsze można uzyskać niezależnie od tego, czy dowolną wartość ma ustawić lub zmodyfikować śledzonych jednostki. Można to zmienić, ustawiając inną `Before\AfterSaveBehavior`.

Nowe zachowanie dotyczące usuwania ClientSetNull

W poprzednich wersjach `DeleteBehavior.Restrict` miał zachowanie w przypadku jednostek śledzonych przez kontekst, jeden zamknięty dopasowane `SetNull` semantyki. W programie EF Core 2.0 nową `ClientSetNull` zachowanie zostało wprowadzona jako domyślne dla relacji opcjonalne. To zachowanie ma `SetNull` semantyki śledzonych jednostek i `Restrict` zachowanie dla baz danych utworzonych przy użyciu programu EF Core. W naszych doświadczeń wynika to najbardziej oczekiwano/przydatne zachowania dla jednostek śledzonych i bazy danych. `DeleteBehavior.Restrict` teraz zostanie uznane dla obiektów śledzonych dla relacji opcjonalne.

Dostawca czasu projektowania pakietów usunięte

`Microsoft.EntityFrameworkCore.Relational.Design` Pakiet został usunięty. Jego zawartości zostały skonsolidowane w `Microsoft.EntityFrameworkCore.Relational` i `Microsoft.EntityFrameworkCore.Design`.

Propaguje to do pakietów projektowania dostawcy. Te pakiety (`Microsoft.EntityFrameworkCore.Sqlite.Design`,
`Microsoft.EntityFrameworkCore.SqlServer.Design`, itp.) zostały usunięte i ich zawartość skonsolidowane w pakiety główne dostawcy.

Aby włączyć `Scaffold-DbContext` lub `dotnet ef dbcontext scaffold` w programie EF Core 2.0 wystarczy odwoływać się do jednego dostawcy pakietu:

```
<PackageReference Include="Microsoft.EntityFrameworkCore.SqlServer"
    Version="2.0.0" />
<PackageReference Include="Microsoft.EntityFrameworkCore.Tools"
    Version="2.0.0"
    PrivateAssets="All" />
<DotNetCliToolReference Include="Microsoft.EntityFrameworkCore.Tools.DotNet"
    Version="2.0.0" />
```

Wprowadzenie do platformy Entity Framework Core

29.11.2018 • 2 minutes to read • [Edit Online](#)

Instalowanie programu EF Core

Podsumowanie kroków niezbędnych do dodawania programu EF Core do aplikacji w różnych platformach oraz popularnych środowisk IDE.

Samouczki krok po kroku

Te wprowadzających samouczków wymagają nie wcześniejsza wiedza Entity Framework Core lub dowolnym określonym środowisku IDE. One spowoduje przejście krok po kroku proces tworzenia prostej aplikacji, który wykonuje kwerendę i zapisuje dane z bazy danych. Udostępniliśmy te samouczki, aby ułatwić rozpoczęcie pracy w różnych systemach operacyjnych i typach aplikacji.

Entity Framework Core może utworzyć model w oparciu o istniejącą bazę danych lub utworzyć bazę danych w oparciu o istniejący model. Dostępne są samouczki obejmujące oba te zastosowania.

- Aplikacje konsoli programu .NET core
 - [Nowa baza danych](#)
- Aplikacje platformy ASP.NET Core
 - [Nowa baza danych](#)
 - [Istniejąca baza danych](#)
 - [Produkty EF Core i Razor Pages](#)
- Universal Windows Platform (systemu Windows UWP) aplikacji
 - [Nowa baza danych](#)
- Aplikacji programu .NET framework
 - [Nowa baza danych](#)
 - [Istniejąca baza danych](#)

NOTE

Samouczki i towarzyszące im przykłady zostały zaktualizowane, aby korzystać z programu EF Core w wersji 2.1. Jednak w większości przypadków powinno być możliwe stworzenie aplikacji kompatybilnej z poprzednimi wersjami, wprowadzając zaledwie minimalne modyfikacje.

Instalowanie platformy Entity Framework Core

07.01.2019 • 10 minutes to read • [Edit Online](#)

Wymagania wstępne

- EF Core jest [.NET Standard 2.0](#) biblioteki. EF Core wymaga implementacji .NET, która obsługuje .NET Standard 2.0 do uruchomienia. EF Core również mogą być przywoływane przez inne biblioteki .NET Standard 2.0.
- Na przykład można użyć programu EF Core programować aplikacje, których platformą docelową .NET Core. Tworzenie aplikacji .NET Core wymaga [zestawu .NET Core SDK](#). Opcjonalnie umożliwia także środowiska programowania, takich jak Visual Studio, Visual Studio for Mac lub Visual Studio Code. Aby uzyskać więcej informacji, sprawdź [wprowadzenie do platformy .NET Core](#).
- EF Core służy do tworzenia aplikacji, których platformą docelową .NET Framework 4.6.1 lub później na Windows, za pomocą programu Visual Studio. Najnowszą wersję programu Visual Studio jest zalecane. Jeśli chcesz używać starszej wersji, takich jak Visual Studio 2015, upewnij się, że [uaktualnić klienta programu NuGet w wersji 3.6.0](#) do pracy z biblioteki .NET Standard 2.0.
- EF Core można uruchamiać na inne implementacje platformy .NET, takich jak Xamarin i .NET Native. Ale w praktyce te implementacje ograniczenia środowiska uruchomieniowego, które mogą dotyczyć, jak dobrze działa programu EF Core w aplikacji. Aby uzyskać więcej informacji, zobacz [implementacji platformy .NET obsługiwanych przez platformę EF Core](#).
- Na koniec dostawców innej bazy danych może wymagać konkretnej bazy danych aparatu wersje, implementacje platformy .NET lub systemów operacyjnych. Upewnij się, że [dostawcy bazy danych programu EF Core](#) jest dostępna, która obsługuje odpowiednim środowisku dla danej aplikacji.

Pobierz środowisko uruchomieniowe programu Entity Framework Core

Do programu EF Core można dodać do aplikacji, należy zainstalować pakiet NuGet dla dostawcy bazy danych, którego chcesz użyć.

Jeśli tworzysz aplikację ASP.NET Core, nie trzeba zainstalować w pamięci i dostawców programu SQL Server. Te dostawcy są uwzględnieni w bieżących wersjach platformy ASP.NET Core, wraz z środowiska uruchomieniowego EF Core.

Aby zainstalować lub zaktualizować pakiety NuGet, należy użyć [platformy .NET Core interfejsu wiersza polecenia (CLI)], okno Menedżera pakietów Visual Studio lub konsoli Menedżera pakietów Visual Studio.

.NET core interfejsu wiersza polecenia

- Użyj następującego polecenia interfejsu wiersza polecenia platformy .NET Core z wiersza polecenia systemu operacyjnego, aby zainstalować lub zaktualizować dostawcy EF Core programu SQL Server:

```
dotnet add package Microsoft.EntityFrameworkCore.SqlServer
```

- Można wskazać określonej wersji w `dotnet add package` polecenia przy użyciu `-v` modyfikator. Na przykład, aby zainstalować pakiety programu EF Core 2.2.0, należy dołączyć `-v 2.2.0` do polecenia.

Aby uzyskać więcej informacji, zobacz [narzędzia interfejsu wiersza polecenia \(CLI\) platformy .NET](#).

Okno Menedżera pakietów NuGet programu Visual Studio

- Wybierz z menu programu Visual Studio **Projekt > Zarządzaj pakietami NuGet**
- Kliknij pozycję **Przeglądaj** lub **aktualizacje** kartę
- Aby zainstalować lub zaktualizować dostawcy programu SQL Server, wybierz `Microsoft.EntityFrameworkCore.SqlServer` pakietu i potwierdź.

Aby uzyskać więcej informacji, zobacz [okno Menedżera pakietów NuGet](#).

Konsola Menedżera pakietów NuGet dla programu Visual Studio

- Wybierz z menu programu Visual Studio **Narzędzia > Menedżer pakietów NuGet > Konsola Menedżera pakietów**
- Aby zainstalować dostawcę programu SQL Server, uruchom następujące polecenie w konsoli Menedżera pakietów:

```
Install-Package Microsoft.EntityFrameworkCore.SqlServer
```

- Aby zaktualizować dostawcę, użyj `Update-Package` polecenia.
- Aby określić określonej wersji, użyj `-Version` modyfikator. Na przykład, aby zainstalować pakiety programu EF Core 2.2.0, należy dodać `-Version 2.2.0` polecenie

Aby uzyskać więcej informacji, zobacz [Konsola Menedżera pakietów](#).

Pobierz narzędzia platformy Entity Framework Core

Możesz zainstalować narzędzia do wykonywania zadań związanych z programem EF Core w projekcie, takich jak tworzenie i stosowanie migracje baz danych lub tworzenie modelu platformy EF Core oparte na istniejącej bazy danych.

Dostępne są dwa zestawy narzędzi:

- [Narzędzi interfejsu wiersza polecenia \(CLI\) platformy .NET Core](#) może służyć w Windows, Linux lub macOS. Te polecenia zaczynają się od `dotnet ef`.
- [Narzędzia konsoli Menedżera pakietów \(PMC\)](#) systemem Windows w programie Visual Studio. Te polecenia rozpoczynają się od czasownika, na przykład `Add-Migration`, `Update-Database`.

Mimo że można również użyć `dotnet ef` polecień z poziomu konsoli Menedżera pakietów, zaleca się korzystania z narzędzi Konsola Menedżera pakietów, podczas korzystania z programu Visual Studio:

- Działają automatycznie z bieżącego projektu wybrany w konsoli zarządzania Pakietami w programie Visual Studio, bez konieczności ręcznie zmienić katalog.
- One automatycznie otwarte pliki wygenerowane za pomocą polecień w programie Visual Studio po zakończeniu polecenia.

Pobierz narzędzia wiersza polecenia platformy .NET Core

Narzędzia interfejsu wiersza polecenia platformy .NET core wymagają .NET Core SDK, wymienione wcześniej w [wymagania wstępne](#).

`dotnet ef` Polecenia są zawarte w aktualnych wersjach programu .NET Core SDK, ale aby włączyć polecenia dla danego projektu, musisz zainstalować `Microsoft.EntityFrameworkCore.Design` pakietu:

```
dotnet add package Microsoft.EntityFrameworkCore.Design
```

W przypadku aplikacji platformy ASP.NET Core, ten pakiet jest uwzględniane automatycznie.

IMPORTANT

Zawsze używaj wersji zgodnej wersji głównej pakiety środowiska uruchomieniowego pakietu Narzędzia.

Pobierz narzędzia w konsoli Menedżera pakietów

Aby uzyskać Konsola Menedżera pakietów narzędzi dla platformy EF Core, należy zainstalować

`Microsoft.EntityFrameworkCore.Tools`

pakietu. Na przykład z programu Visual Studio:

```
Install-Package Microsoft.EntityFrameworkCore.Tools
```

W przypadku aplikacji platformy ASP.NET Core, ten pakiet jest uwzględniane automatycznie.

Uaktualnianie do najnowszej EF Core

- Każdym wydaniu nowej wersji programu EF Core, wydaniu nowej wersji dostawców, które są częścią projektu programu EF Core, takich jak `Microsoft.EntityFrameworkCore.SqlServer`, `Microsoft.EntityFrameworkCore.Sqlite`, i `Microsoft.EntityFrameworkCore.InMemory`. Możesz po prostu uaktualnić do nowej wersji dostawcy, aby uzyskać wszystkie ulepszenia.
- EF Core wraz z programu SQL Server i dostawców w pamięci są uwzględnione w aktualnych wersjach programu ASP.NET Core. Aby przeprowadzić uaktualnienie do nowszej wersji programu EF Core istniejącej aplikacji platformy ASP.NET Core, zawsze należy uaktualnić wersję platformy ASP.NET Core.
- Jeśli musisz zaktualizować aplikację, która używa dostawcy bazy danych innej firmy, Zawsze sprawdzaj dostępność aktualizacji dostawcy, który jest zgodny z wersją programu EF Core, którego chcesz użyć. Na przykład dostawcy baz danych dla wcześniejszych wersji nie są zgodne z wersji 2.0 środowiska uruchomieniowego EF Core.
- Innych dostawców dla platformy EF Core zwykle nie zwalniaj wersji poprawki wraz z środowiska uruchomieniowego EF Core. Aby uaktualnić aplikację, która używa dostawcy innej firmy, do wersji poprawki programu EF Core, może być konieczne dodanie bezpośrednie odwołanie do poszczególnych składników środowiska uruchomieniowego EF Core, takie jak `Microsoft.EntityFrameworkCore` i `Microsoft.EntityFrameworkCore.Relational`.
- Jeśli aktualizujesz istniejącą aplikację do najnowszej wersji programu EF Core niektórych odwołań do starszych pakietów programu EF Core może być konieczne ręczne usunięcie:
 - Bazy danych dostawcy projektowania pakietów, takich jak `Microsoft.EntityFrameworkCore.SqlServer.Design` nie są już wymagane lub obsługiwane z programem EF Core 2.0 lub nowszy, ale nie są automatycznie usuwane po uaktualnieniu innych pakietów.
 - Narzędzia wiersza poleceń platformy .NET obejmuje zestawu .NET SDK od wersji 2.1, dzięki czemu można usunąć odwołania do tego pakietu z pliku projektu:

```
<DotNetCliToolReference Include="Microsoft.EntityFrameworkCore.Tools.DotNet" Version="2.0.0" />
```

- Aplikacji przeznaczonych dla środowiska .NET Framework może być konieczne zmiany, aby pracować z biblioteki .NET Standard 2.0:
 - Edytuj plik projektu i upewnij się, że w grupie właściwości początkowe pojawia się następujący wpis:

```
<AutoGenerateBindingRedirects>true</AutoGenerateBindingRedirects>
```

- Dla projektów testowych również upewnij się, że występuje następujący wpis:

```
<GenerateBindingRedirectsOutputType>true</GenerateBindingRedirectsOutputType>
```

Wprowadzenie do programu EF Core na platformie .NET Core

28.08.2018 • 2 minutes to read • [Edit Online](#)

Te samouczki 101 wymagają nie wcześniejsza wiedza programu Entity Framework Core lub Visual Studio. One spowoduje przejście krok po kroku przez proces tworzenia prostego .NET Core aplikację Konsolową która wykonuje kwerendę i zapisuje dane z bazy danych. Samouczki, można wykonać na żadnej platformy obsługiwanej przez platformy .NET Core (Windows, OS x, Linux, itd.).

Można znaleźć w dokumentacji platformy .NET Core w docs.microsoft.com/dotnet/articles/core.

Wprowadzenie do nowej bazy danych z programem EF Core w aplikacji Konsolowej .NET Core

25.10.2018 • 6 minutes to read • [Edit Online](#)

W tym samouczku utworzysz aplikację konsoli .NET Core, która wykonuje dostęp do danych w bazie danych SQLite przy użyciu platformy Entity Framework Core. Migracje umożliwia tworzenie bazy danych z modelem. Zobacz [ASP.NET Core — Nowa baza danych](#) dla wersji programu Visual Studio przy użyciu platformy ASP.NET Core MVC.

[Wyświetlanie przykładowych w tym artykule w witrynie GitHub.](#)

Wymagania wstępne

- [.NET Core 2.1 SDK](#)

Tworzenie nowego projektu

- Utwórz nowy projekt konsoli:

```
dotnet new console -o ConsoleApp.SQLite
```

Zmień bieżący katalog

W kolejnych krokach samouczka, należy wydać `dotnet` polecień dla aplikacji.

- Możemy zmienić bieżący katalog do katalogu aplikacji w następujący sposób:

```
cd ConsoleApp.SQLite/
```

Instalowanie platformy Entity Framework Core

Aby korzystać z programu EF Core, należy zainstalować pakiet dla dostawców bazy danych, który ma pod kątem. W tym instruktażu wykorzystano bazy danych SQLite. Aby uzyskać listę dostępnych dostawców zobacz [dostawcy baz danych](#).

- Zainstaluj Microsoft.EntityFrameworkCore.Sqlite i Microsoft.EntityFrameworkCore.Design

```
dotnet add package Microsoft.EntityFrameworkCore.Sqlite  
dotnet add package Microsoft.EntityFrameworkCore.Design
```

- Uruchom `dotnet restore` zainstalować nowe pakiety.

Tworzenie modelu

Definiowanie klas jednostek i kontekstu, które tworzą model.

- Utwórz nową `Model.cs` pliku o następującej zawartości.

```
using Microsoft.EntityFrameworkCore;
using System.Collections.Generic;

namespace ConsoleApp.SQLite
{
    public class BloggingContext : DbContext
    {
        public DbSet<Blog> Blogs { get; set; }
        public DbSet<Post> Posts { get; set; }

        protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
        {
            optionsBuilder.UseSqlite("Data Source=blogging.db");
        }
    }

    public class Blog
    {
        public int BlogId { get; set; }
        public string Url { get; set; }

        public ICollection<Post> Posts { get; set; }
    }

    public class Post
    {
        public int PostId { get; set; }
        public string Title { get; set; }
        public string Content { get; set; }

        public int BlogId { get; set; }
        public Blog Blog { get; set; }
    }
}
```

Porada: W rzeczywistej aplikacji, możesz umieścić każda klasa w oddzielnym pliku, a parametry połączenia w zmiennej środowisku lub pliku konfiguracji. W celu uproszczenia tego samouczka wszystko, co znajduje się w jednym pliku.

Tworzenie bazy danych

Po utworzeniu modelu, użyj [migracje](#) utworzyć bazę danych.

- Uruchom `dotnet ef migrations add InitialCreate` tworzenia szkieletu migracji i utworzyć początkowy zestaw tabel dla modelu.
- Uruchom `dotnet ef database update` zastosować nową migrację do bazy danych. To polecenie tworzy bazę danych przed zastosowaniem migracji.

*Blogging.db** bazy danych SQLite znajduje się w katalogu projektu.

Użyj modelu

- Otwórz *Program.cs* i zastąp jego zawartość następującym kodem:

```

using System;

namespace ConsoleApp.SQLite
{
    public class Program
    {
        public static void Main()
        {
            using (var db = new BloggingContext())
            {
                db.Blogs.Add(new Blog { Url = "http://blogs.msdn.com/adonet" });
                var count = db.SaveChanges();
                Console.WriteLine("{0} records saved to database", count);

                Console.WriteLine();
                Console.WriteLine("All blogs in database:");
                foreach (var blog in db.Blogs)
                {
                    Console.WriteLine("- {0}", blog.Url);
                }
            }
        }
    }
}

```

- Testowanie aplikacji z konsoli. Zobacz [Uwaga programu Visual Studio](#) do uruchomienia aplikacji w programie Visual Studio.

`dotnet run`

Blog jeden jest zapisywany w bazie danych i szczegółowe informacje o wszystkich blogów są wyświetlane w konsoli.

```

ConsoleApp.SQLite>dotnet run
1 records saved to database

All blogs in database:
- http://blogs.msdn.com/adonet

```

Zmiana modelu:

- Jeśli wprowadzisz zmiany w modelu, można użyć `dotnet ef migrations add` polecenie, aby utworzyć szkielet nowego [migracji](#). Po zaznaczeniu tej opcji utworzony szkielet kodu (i wprowadzone wymagane zmiany), można użyć `dotnet ef database update` polecenie, aby zastosować schemat zmienia się z bazą danych.
- Używa programu EF Core `__EFMigrationsHistory` tabeli w bazie danych, aby śledzić migracje, które zostały już zastosowane do bazy danych.
- Aparat bazy danych SQLite nie obsługuje niektórych zmiany schematu, które są obsługiwane w większości innych relacyjnych baz danych. Na przykład `DropColumn` operacja nie jest obsługiwana. EF Core migracji wygeneruje kod dla tych operacji. Ale Jeśli spróbujesz zastosować je do bazy danych lub wygenerować skrypt programu EF Core zgłasza wyjątek wyjątków. Zobacz [ograniczenia SQLite](#). W nowych wdrożeniach należy wziąć pod uwagę porzucenie bazy danych i tworzenia nowej, a nie przy użyciu migracji po zmianie modelu.

Uruchamianie z programu Visual Studio

Aby uruchomić ten przykład z programu Visual Studio, należy ustawić katalogu roboczego ręcznie, aby być w katalogu głównym projektu. Jeśli nie ustawisz katalogu roboczego następujące

`Microsoft.Data.Sqlite.SqliteException` zgłoszany: `SQLite Error 1: 'no such table: Blogs'`.

Aby ustawić katalog roboczy:

- W **Eksploratora rozwiązań**, kliknij prawym przyciskiem myszy projekt, a następnie wybierz **właściwości**.
- Wybierz **debugowania** karty w okienku po lewej stronie.
- Ustaw **katalog roboczy** do katalogu projektu.
- Zapisz zmiany.

Dodatkowe zasoby

- Samouczek: Rozpoczynanie pracy z programem EF Core programu ASP.NET Core przy użyciu nowej bazy danych przy użyciu systemu SQLite
- Samouczek: Rozpoczynanie pracy ze stronami Razor w programie ASP.NET Core
- Samouczek: Strony Razor za pomocą platformy Entity Framework Core w programie ASP.NET Core

Wprowadzenie do programu EF Core programu ASP.NET Core

06.11.2018 • 2 minutes to read • [Edit Online](#)

Przedstawione samouczki nie wymagają wcześniejszej wiedzy dotyczącej korzystania z programów Entity Framework Core oraz Visual Studio. Pozwolą na przejście krok po kroku przez proces tworzenia prostej aplikacji konsolowej platformy ASP.NET Core, która wykonuje kwerendę i zapisuje dane z bazy danych. Możesz skorzystać z samouczka dotyczącego tworzenia modelu w oparciu o istniejącą bazę danych lub z samouczka dotyczącego tworzenia bazy danych w oparciu o model.

Dokumentację platformy ASP.NET Core można znaleźć w artykule [wprowadzeniu do platformy ASP.NET Core](#).

NOTE

Samouczki i towarzyszące im przykłady zostały zaktualizowane, aby korzystać z programu EF Core w wersji 2.0 (z wyjątkiem samouczka dotyczącego platformy uniwersalnej systemu Windows, który nadal używa programu EF Core 1.1). Jednak w większości przypadków powinno być możliwe stworzenie aplikacji zgodnej z poprzednimi wersjami, wprowadzając zaledwie minimalne modyfikacje.

Wprowadzenie do programu EF Core programu ASP.NET Core przy użyciu nowej bazy danych

11.01.2019 • 10 minutes to read • [Edit Online](#)

W tym samouczku utworzysz aplikację ASP.NET Core MVC, który wykonuje podstawowe dane dostępu przy użyciu platformy Entity Framework Core. W samouczku migracje utworzyć bazę danych z modelu danych.

Z pomocą programu Visual Studio 2017 na Windows lub za pomocą interfejsu wiersza poleceń platformy .NET Core w systemie Windows, macOS lub Linux, można wykonać kroki samouczka.

Wyświetl przykład w tym artykule w witrynie GitHub:

- [Program Visual Studio 2017 z programem SQL Server](#)
- [.NET core interfejsu wiersza poleceń za pomocą SQLite](#).

Wymagania wstępne

Zainstaluj następujące oprogramowanie:

- [Visual Studio](#)
- [.NET Core CLI](#)
- [Visual Studio 2017 w wersji 15.7 lub nowszej](#) za pomocą tych obciążeniach:
 - **ASP.NET i tworzenie aplikacji internetowych** (w obszarze **sieć Web i chmura**)
 - **Programowanie dla wielu platform .NET core** (w obszarze **inne zestawy narzędzi**)
- [.NET core 2.1 SDK](#).

Tworzenie nowego projektu

- [Visual Studio](#)
- [.NET Core CLI](#)
- Otwórz program Visual Studio 2017
- **Plik > Nowy > Projekt**
- Z menu po lewej stronie wybierz **zainstalowane > Visual C# > .NET Core**.
- Wybierz **aplikacji sieci Web platformy ASP.NET Core**.
- Wprowadź **EFGetStarted.AspNetCore.NewDb** nazwę i kliknij przycisk **OK**.
- W **Nowa aplikacja internetowa ASP.NET Core** okno dialogowe:
 - Upewnij się, że **platformy .NET Core i platformy ASP.NET Core 2.1** wybranych z listy rozwijanej
 - Wybierz **aplikacji sieci Web (Model-View-Controller)** szablonu projektu
 - Upewnij się, że **uwierzytelniania** ustawiono **bez uwierzytelniania**
 - Kliknij przycisk **OK**

Ostrzeżenie: Jeśli używasz **indywidualne konta użytkowników** zamiast **Brak dla uwierzytelniania**, a następnie na model Entity Framework Core zostanie dodany do projektu w `Models\IdentityModel.cs`. Przy użyciu technik, z którego dowiesz się, w tym samouczku, można dodać drugi model lub rozszerzyć ten istniejący model zawiera Twoje klas jednostek.

Instalowanie platformy Entity Framework Core

Aby zainstalować programu EF Core, należy zainstalować pakiet dla dostawców bazy danych programu EF Core, który ma pod kątem. Aby uzyskać listę dostępnych dostawców, zobacz [dostawcy baz danych](#).

- [Visual Studio](#)
- [.NET Core CLI](#)

Na potrzeby tego samouczka nie trzeba zainstalować pakiet dostawcy, ponieważ w tym samouczku użyto programu SQL Server. Pakiet dostawcy programu SQL Server znajduje się w [meta Microsoft.aspnetcore.all Microsoft.AspNetCore.App](#).

Tworzenie modelu

Zdefiniuj klasę kontekstu i klas jednostek, które tworzą model.

- [Visual Studio](#)
- [.NET Core CLI](#)
- Kliknij prawym przyciskiem myszy **modeli** i wybierz polecenie **Dodaj > klasa**.
- Wprowadź **Model.cs** jako nazwę i kliknij przycisk **OK**.
- Zastąp zawartość pliku następującym kodem:

```
using Microsoft.EntityFrameworkCore;
using System.Collections.Generic;

namespace EFGetStarted.AspNetCore.NewDb.Models
{
    public class BloggingContext : DbContext
    {
        public BloggingContext(DbContextOptions<BloggingContext> options)
            : base(options)
        { }

        public DbSet<Blog> Blogs { get; set; }
        public DbSet<Post> Posts { get; set; }
    }

    public class Blog
    {
        public int BlogId { get; set; }
        public string Url { get; set; }

        public ICollection<Post> Posts { get; set; }
    }

    public class Post
    {
        public int PostId { get; set; }
        public string Title { get; set; }
        public string Content { get; set; }

        public int BlogId { get; set; }
        public Blog Blog { get; set; }
    }
}
```

Zazwyczaj jest aplikacji produkcyjnej przełączyć każda klasa w oddzielnym pliku. Dla uproszczenia ten samouczek umieszcza w ramach tych zajęć w jednym pliku.

Zarejestrowanie kontekście wstrzykiwanie zależności

Usługi (takie jak `BloggingContext`) zostały zarejestrowane przy użyciu [wstrzykiwanie zależności](#) podczas uruchamiania aplikacji. Składniki, które wymagają tych usług, (na przykład kontrolerów MVC) znajdują się w tych usług za pomocą właściwości lub parametry konstruktora.

Zapewnienie `BloggingContext` dostępne dla kontrolerów MVC, zarejestruj go jako usługę.

- [Visual Studio](#)
- [.NET Core CLI](#)
- W `Startup.cs` Dodaj następujący kod `using` instrukcji:

```
using EFGetStarted.AspNetCore.NewDb.Models;
using Microsoft.EntityFrameworkCore;
```

- Dodaj następujący wyróżniony kod do `ConfigureServices` metody:

```
public void ConfigureServices(IServiceCollection services)
{
    services.Configure<CookiePolicyOptions>(options =>
    {
        // This lambda determines whether user consent for non-essential cookies is needed for a given
        request.
        options.CheckConsentNeeded = context => true;
        options.MinimumSameSitePolicy = SameSiteMode.None;
    });

    services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_1);

    var connection = @"Server=
(localdb)\mssqllocaldb;Database=EFGetStarted.AspNetCore.NewDb;Trusted_Connection=True;ConnectRetryCount
=0";
    services.AddDbContext<BloggingContext>
        (options => options.UseSqlServer(connection));
    // BloggingContext requires
    // using EFGetStarted.AspNetCore.NewDb.Models;
    // UseSqlServer requires
    // using Microsoft.EntityFrameworkCore;
}
```

Aplikacji produkcyjnej zazwyczaj umieścić ciąg połączenia w zmiennej środowisku lub pliku konfiguracji. Dla uproszczenia w tym samouczku zdefiniowano go w kodzie. Zobacz [parametry połączenia](#) Aby uzyskać więcej informacji.

Tworzenie bazy danych

Następujące kroki użycia [migracje](#) utworzyć bazę danych.

- [Visual Studio](#)
- [.NET Core CLI](#)
- **Narzędzia > Menedżer pakietów NuGet > Konsola Menedżera pakietów**
- Uruchom następujące polecenia:

```
Add-Migration InitialCreate
Update-Database
```

Jeśli zostanie wyświetlony błąd wskazujący

The term 'add-migration' is not recognized as the name of a cmdlet, zamknij i otwórz ponownie program Visual Studio.

Add-Migration Polecenia scaffolds migracji, aby utworzyć początkowy zestaw tabel dla modelu.

Update-Database Polecenie tworzy bazę danych i dotyczy nowej migracji.

Tworzenie kontrolera

Tworzenia szkieletu kontrolera i widoki dla **Blog** jednostki.

- [Visual Studio](#)
- [.NET Core CLI](#)

- Kliknij prawym przyciskiem myszy **kontrolerów** folderu w **Eksploratora rozwiązań** i wybierz **Dodaj > kontrolera**.
- Wybierz **kontroler MVC z widokami używający narzędzia Entity Framework** i kliknij przycisk **Dodaj**.
- Ustaw **klaś modelu** do **Blog** i **klaś kontekstu danych** do **BloggingContext**.
- Kliknij przycisk **Dodaj**.

Aparat tworzenia szkieletów tworzy następujące pliki:

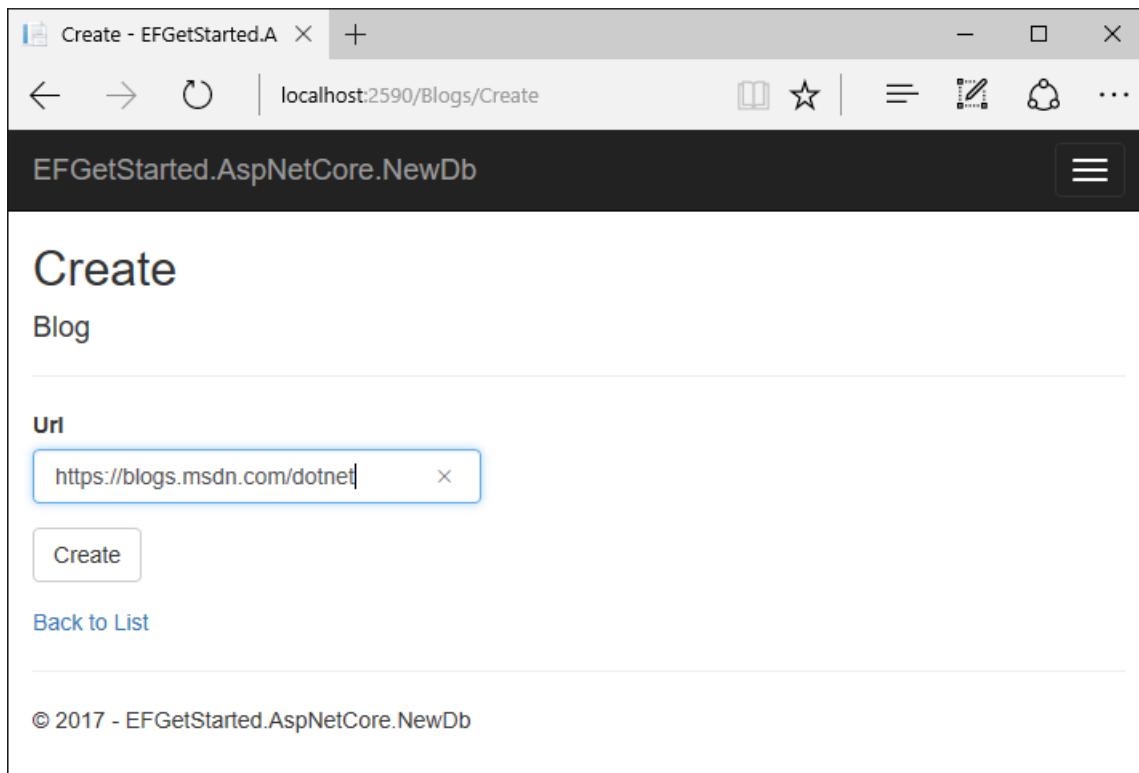
- Kontroler (*Controllers/BlogsController.cs*)
- Widokami razor dla stron Create, Delete, szczegółowe informacje, edycji i indeksu (*Views/Blogs/*.cshtml*)

Uruchamianie aplikacji

- [Visual Studio](#)
- [.NET Core CLI](#)

- **Debugowanie > Uruchom bez debugowania**

- Przejdz do **/Blogs**
- Użyj **Utwórz nowy** link, aby utworzyć wpisy na blogu.



- Test **szczegóły**, **Edytuj**, i **Usuń** łącza.

Url	
https://blogs.msdn.com/dotnet	Edit Details Delete

Dodatkowe zasoby

- Samouczek: Rozpoczynanie pracy z programem EF Core na platformie .NET Core za pomocą nowej bazy danych przy użyciu systemu SQLite
- Samouczek: Rozpoczynanie pracy ze stronami Razor w programie ASP.NET Core
- Samouczek: Strony razor za pomocą platformy Entity Framework Core w programie ASP.NET Core

Wprowadzenie do programu EF Core programu ASP.NET Core z istniejącej bazy danych

07.01.2019 • 9 minutes to read • [Edit Online](#)

W tym samouczku utworzysz aplikację ASP.NET Core MVC, który wykonuje podstawowe dane dostępu przy użyciu platformy Entity Framework Core. Możesz odtwarzać istniejącą bazę danych do tworzenia modelu Entity Framework.

[Wyświetlanie przykładowych w tym artykule w witrynie GitHub.](#)

Wymagania wstępne

Zainstaluj następujące oprogramowanie:

- [Visual Studio 2017 w wersji 15.7](#) za pomocą tych obciążeniach:
 - **ASP.NET i tworzenie aplikacji internetowych** (w obszarze **sieć Web i chmura**)
 - **Programowanie dla wielu platform .NET core** (w obszarze **inne zestawy narzędzi**)
- [.NET core 2.1 SDK.](#)

Utwórz bazę danych do obsługi blogów

W tym samouczku **do obsługi blogów** bazy danych w ramach wystąpienia LocalDb jako istniejącej bazy danych. Jeśli masz już utworzoną **do obsługi blogów** bazy danych jako część innego samouczek, należy pominąć tę procedurę.

- Otwórz program Visual Studio
- **Narzędzia -> nawiązać połączenie z bazą danych...**
- Wybierz **programu Microsoft SQL Server** i kliknij przycisk **Kontynuuj**
- Wprowadź **(localdb) \mssqllocaldb** jako **nazwy serwera**
- Wprowadź **wzorca** jako **Nazwa bazy danych** i kliknij przycisk **OK**
- Baza danych master jest teraz wyświetlany w obszarze **połączeń danych** w **Eksploratora serwera**
- Kliknij prawym przyciskiem myszy w bazie danych w **Eksploratora serwera** i wybierz **nowe zapytanie**
- Skopiuj skrypt w edytorze zapytań
- Kliknij prawym przyciskiem myszy w edytorze zapytań, a następnie wybierz pozycję **wykonania**

```

CREATE DATABASE [Blogging];
GO

USE [Blogging];
GO

CREATE TABLE [Blog] (
    [BlogId] int NOT NULL IDENTITY,
    [Url] nvarchar(max) NOT NULL,
    CONSTRAINT [PK_Blog] PRIMARY KEY ([BlogId])
);
GO

CREATE TABLE [Post] (
    [PostId] int NOT NULL IDENTITY,
    [BlogId] int NOT NULL,
    [Content] nvarchar(max),
    [Title] nvarchar(max),
    CONSTRAINT [PK_Post] PRIMARY KEY ([PostId]),
    CONSTRAINT [FK_Post_Blog_BlogId] FOREIGN KEY ([BlogId]) REFERENCES [Blog] ([BlogId]) ON DELETE CASCADE
);
GO

INSERT INTO [Blog] (Url) VALUES
('http://blogs.msdn.com/dotnet'),
('http://blogs.msdn.com/webdev'),
('http://blogs.msdn.com/visualstudio')
GO

```

Tworzenie nowego projektu

- Otwórz program Visual Studio 2017
- **Plik > Nowy > Projekt...**
- Z menu po lewej stronie wybierz **zainstalowane > Visual C# > sieci Web**
- Wybierz **aplikacji sieci Web programu ASP.NET Core** szablonu projektu
- Wprowadź **EFGetStarted.AspNetCore.ExistingDb** jako nazwa (jego musi być zgodna dokładnie obszaru nazw, które są później używane w kodzie) i kliknij przycisk **OK**
- Poczekaj, aż **Nowa aplikacja internetowa ASP.NET Core** wyświetlać okno dialogowe
- Upewnij się, że menu rozwijanego platform docelowych jest ustawiona na **platformy .NET Core**, a lista rozwijana wersji jest ustawiona na **platformy ASP.NET Core 2.1**
- Wybierz **aplikacji sieci Web (Model-View-Controller)** szablonu
- Upewnij się, że **uwierzytelniania** ustawiono **bez uwierzytelniania**
- Kliknij przycisk **OK**

Instalowanie platformy Entity Framework Core

Aby zainstalować programu EF Core, należy zainstalować pakiet dla dostawców bazy danych programu EF Core, który ma pod kątem. Aby uzyskać listę dostępnych dostawców zobacz [dostawcy baz danych](#).

Na potrzeby tego samouczka nie trzeba zainstalować pakiet dostawcy, ponieważ w tym samouczku użyto programu SQL Server. Pakiet dostawcy programu SQL Server znajduje się w [meta Microsoft.aspnetcore.all](#) [Microsoft.AspNetCore.App](#).

Model odtworzyć.

Teraz nadszedł czas na tworzenie modelu platformy EF, w oparciu o istniejącą bazę danych.

- **Narzędzia -> pakietu NuGet Manager -> Konsola Menedżera pakietów**

- Uruchom następujące polecenie, aby utworzyć model z istniejącej bazy danych:

```
Scaffold-DbContext "Server=(localdb)\mssqllocaldb;Database=Blogging;Trusted_Connection=True;"  
Microsoft.EntityFrameworkCore.SqlServer -OutputDir Models
```

Jeśli zostanie wyświetlony błąd wskazujący

The term 'Scaffold-DbContext' is not recognized as the name of a cmdlet , a następnie zamknij i otwórz ponownie program Visual Studio.

TIP

Można określić tabel, które mają zostać wygenerowane jednostki dla, dodając `-Tables` argument polecenia powyżej. Na przykład `-Tables Blog,Post` .

Proces odtwarzania utworzony klas jednostek (`Blog.cs` & `Post.cs`) i pochodnej kontekstu (`BloggingContext.cs`) na podstawie schematu istniejącej bazy danych.

Klasy jednostki są proste obiekty języka C#, które reprezentują będziesz zapytań i zapisywaniem danych. Poniżej przedstawiono `Blog` i `Post` klas jednostek:

```
using System;  
using System.Collections.Generic;  
  
namespace EFGetStarted.AspNetCore.ExistingDb.Models  
{  
    public partial class Blog  
    {  
        public Blog()  
        {  
            Post = new HashSet<Post>();  
        }  
  
        public int BlogId { get; set; }  
        public string Url { get; set; }  
  
        public ICollection<Post> Post { get; set; }  
    }  
}
```

```
using System;  
using System.Collections.Generic;  
  
namespace EFGetStarted.AspNetCore.ExistingDb.Models  
{  
    public partial class Post  
    {  
        public int PostId { get; set; }  
        public int BlogId { get; set; }  
        public string Content { get; set; }  
        public string Title { get; set; }  
  
        public Blog Blog { get; set; }  
    }  
}
```

TIP

Aby włączyć ładowanie z opóźnieniem, należy wybrać właściwości nawigacji `virtual` (Blog.Post i Post.Blog).

Kontekst reprezentuje sesję z bazą danych i umożliwia zapytania i Zapisz wystąpienie klas jednostek.

```
public partial class BloggingContext : DbContext
{
    public BloggingContext()
    {
    }

    public BloggingContext(DbContextOptions<BloggingContext> options)
        : base(options)
    {
    }

    public virtual DbSet<Blog> Blog { get; set; }
    public virtual DbSet<Post> Post { get; set; }

    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    {
        if (!optionsBuilder.IsConfigured)
        {
            #warning To protect potentially sensitive information in your connection string, you should move it
            out of source code. See http://go.microsoft.com/fwlink/?LinkId=723263 for guidance on storing connection
            strings.

            optionsBuilder.UseSqlServer(@"Server=
(localdb)\mssqllocaldb;Database=Blogging;Trusted_Connection=True;");
        }
    }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Blog>(entity =>
        {
            entity.Property(e => e.Url).IsRequired();
        });

        modelBuilder.Entity<Post>(entity =>
        {
            entity.HasOne(d => d.Blog)
                .WithMany(p => p.Post)
                .HasForeignKey(d => d.BlogId);
        });
    }
}
```

Zarejestrowanie kontekst wstrzykiwanie zależności

Pojęcie wstrzykiwanie zależności stanowi podstawę do platformy ASP.NET Core. Usługi (takie jak `BloggingContext`) zostały zarejestrowane przy użyciu iniekcji zależności podczas uruchamiania aplikacji. Składniki, które wymagają tych usług, (na przykład kontrolerach MVC) znajdują się następnie tych usług za pomocą właściwości lub parametry konstruktora. Aby uzyskać więcej informacji na temat wstrzykiwanie zależności zobacz [wstrzykiwanie zależności](#) artykułu w witrynie programu ASP.NET.

Rejestrowanie i konfigurowanie kontekstu w pliku Startup.cs

Zapewnienie `BloggingContext` dostępne dla kontrolerów MVC, zarejestruj go jako usługę.

- Otwórz `Startup.cs`
- Dodaj następujący kod `using` instrukcji na początku pliku

```
using EFGetStarted.AspNetCore.ExistingDb.Models;
using Microsoft.EntityFrameworkCore;
```

Teraz możesz używać `AddDbContext(...)` metodę, aby zarejestrować go jako usługę.

- Znajdź `ConfigureServices(...)` — metoda
- Dodaj następujący wyróżniony kod, aby zarejestrować kontekst jako usługa

```
// This method gets called by the runtime. Use this method to add services to the container.
public void ConfigureServices(IServiceCollection services)
{
    services.Configure<CookiePolicyOptions>(options =>
    {
        // This lambda determines whether user consent for non-essential cookies is needed for a given
        request.
        options.CheckConsentNeeded = context => true;
        options.MinimumSameSitePolicy = SameSiteMode.None;
    });

    services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_1);

    var connection = @"Server=
(localdb)\mssqllocaldb;Database=Blogging;Trusted_Connection=True;ConnectRetryCount=0";
    services.AddDbContext<BloggingContext>(options => options.UseSqlServer(connection));
}
```

TIP

W rzeczywistej aplikacji zwykle przełączyć parametry połączenia w zmiennej środowisku lub pliku konfiguracji. W tym samouczku dla uproszczenia, ma możesz zdefiniować go w kodzie. Aby uzyskać więcej informacji, zobacz [parametry połączenia](#).

Tworzenie kontrolera i widoki

- Kliknij prawym przyciskiem myszy **kontrolerów** folderu w **Eksploratora rozwiązań** i wybierz **Dodaj -> kontrolera...**
- Wybierz **kontroler MVC z widokami używający narzędzia Entity Framework** i kliknij przycisk **Ok**
- Ustaw **klaś modelu** do **Blog** i **klaś kontekstu danych** do **BloggingContext**
- Kliknij przycisk **Dodaj**

Uruchamianie aplikacji

Teraz można uruchomić aplikacji, aby zobaczyć go w działaniu.

- **Debuguj -> Start bez debugowania**
- Aplikacja zostanie skompilowana i zostanie otwarta w przeglądarce sieci web
- Przejdz do `/Blogs`
- Kliknij przycisk **Utwórz nową**
- Wprowadź **adresu Url** nowego bloga, a następnie kliknij przycisk **Create**

The screenshot shows a browser window with the title bar "Create - EFGetStarted.A" and the address bar "localhost:2590/Blogs/Create". The main content area is titled "Create" and has a sub-section "Blog". Below this, there is a form field labeled "Url" containing the value "https://blogs.msdn.com/dotnet". A "Create" button is visible below the input field. At the bottom of the page, there is a link "Back to List" and a copyright notice "© 2017 - EFGetStarted.AspNetCore.NewDb".

The screenshot shows a browser window with the title bar "Index - EFGetStarted.A" and the address bar "localhost:2590/Blogs". The main content area is titled "Index" and has a "Create New" link. Below this, there is a table-like structure listing multiple blog entries. Each entry includes a URL and three actions: "Edit | Details | Delete". The URLs listed are: http://blogs.msdn.com/dotnet, http://blogs.msdn.com/webdev, http://blogs.msdn.com/visualstudio, http://blogs.msdn.com/dotnet, http://blogs.msdn.com/webdev, http://blogs.msdn.com/visualstudio, http://blogs.msdn.com/dotnet, http://blogs.msdn.com/webdev, and http://blogs.msdn.com/visualstudio. At the bottom of the page, there is a copyright notice "© 2017 - EFGetStarted.AspNetCore.ExistingDb".

Następne kroki

Aby uzyskać więcej informacji na temat sposobu tworzenia szkieletu klasy kontekstu i jednostek, zobacz

następujące artykuły:

- [Odtwarzanie](#)
- [Entity Framework Core odnoszą się narzędzia — interfejs wiersza polecenia platformy .NET](#)
- [Entity Framework Core odnoszą się narzędzia — Konsola Menedżera pakietów](#)

Wprowadzenie do programu EF Core na platformie Universal Windows (UWP)

29.09.2018 • 2 minutes to read • [Edit Online](#)

Te samouczki 101-level wymaga nie wcześniejsza wiedza programu Entity Framework Core lub Visual Studio. One spowoduje przejście instrukcje dotyczące tworzenia prostej aplikacji okna platformy Uniwersalnej, który wykonuje kwerendę i zapisuje dane z bazy danych przy użyciu programu EF Core.

Więcej szczegółów na temat tworzenia aplikacji platformy uniwersalnej systemu Windows można znaleźć w [dokumentacji platformy uniwersalnej systemu Windows](#).

Wprowadzenie do programu EF Core na platformie Universal Windows (UWP) przy użyciu nowej bazy danych

25.10.2018 • 10 minutes to read • [Edit Online](#)

W tym samouczku utworzysz aplikację platformy uniwersalnej Windows (UWP), która wykonuje dostęp do podstawowych danych lokalnej bazy danych SQLite przy użyciu platformy Entity Framework Core.

[Wyświetlanie przykładowych w tym artykule w witrynie GitHub.](#)

Wymagania wstępne

- [Windows 10 Fall Creators Update \(10.0; Kompilacja 16299\) lub nowszym.](#)
- [Visual Studio 2017 w wersji 15.7 lub nowszej z Universal Windows Platform Development obciążenia.](#)
- [Zestaw SDK programu .NET core 2.1 lub nowszej](#) lub nowszej.

IMPORTANT

Ten samouczek używa platformy Entity Framework Core [migracje](#) polecień do tworzenia i aktualizowania schematu bazy danych. Te polecenia nie działają bezpośrednio z projektów platformy UWP. Z tego powodu modelu danych aplikacji znajduje się w projekcie biblioteki udostępnionej, a oddzielną aplikację konsoli .NET Core jest używany do uruchamiania polecenia.

Utwórz projekt biblioteki do przechowywania modelu danych

- Otwórz program Visual Studio
- **Plik > Nowy > Projekt**
- Z menu po lewej stronie wybierz **zainstalowane > Visual C# > .NET Standard**.
- Wybierz **biblioteki klas (.NET Standard)** szablonu.
- Nadaj projektowi nazwę *Blogging.Model*.
- Nazwij rozwiązanie *do obsługi blogów*.
- Kliknij przycisk **OK**.

Zainstaluj środowisko uruchomieniowe programu Entity Framework Core w projekcie modelu danych

Aby korzystać z programu EF Core, należy zainstalować pakiet dla dostawców bazy danych, który ma pod kątem. Ten samouczek używa bazy danych SQLite. Aby uzyskać listę dostępnych dostawców zobacz [dostawcy baz danych](#).

- **Narzędzia > Menedżer pakietów NuGet > Konsola Menedżera pakietów.**
- Upewnij się, że projekt biblioteki *Blogging.Model* jest wybrany jako **domyślny projekt** w konsoli Menedżera pakietów.
- Uruchom `Install-Package Microsoft.EntityFrameworkCore.Sqlite`

Tworzenie modelu danych

Teraz nadszedł czas, aby zdefiniować *DbContext* i klas jednostek, które tworzą model.

- Usuń *Class1.cs*.
- Tworzenie *Model.cs* następującym kodem:

```
using Microsoft.EntityFrameworkCore;
using System.Collections.Generic;

namespace Blogging.Model
{
    public class BloggingContext : DbContext
    {
        public DbSet<Blog> Blogs { get; set; }
        public DbSet<Post> Posts { get; set; }

        protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
        {
            optionsBuilder.UseSqlite("Data Source=blogging.db");
        }
    }

    public class Blog
    {
        public int BlogId { get; set; }
        public string Url { get; set; }

        public List<Post> Posts { get; set; }
    }

    public class Post
    {
        public int PostId { get; set; }
        public string Title { get; set; }
        public string Content { get; set; }

        public int BlogId { get; set; }
        public Blog Blog { get; set; }
    }
}
```

Utwórz nowy projekt konsoli, aby uruchamiać polecenia migracji

- W **Eksploratora rozwiązań**, kliknij prawym przyciskiem myszy rozwiązanie, a następnie wybierz **Dodaj > Nowy projekt**.
- Z menu po lewej stronie wybierz **zainstalowane > Visual C# > .NET Core**.
- Wybierz **Aplikacja konsoli (.NET Core)** szablonu projektu.
- Nadaj projektowi nazwę *Blogging.Migrations.Startup* i kliknij przycisk **OK**.
- Dodaj odwołanie do projektu z *Blogging.Migrations.Startup* projekt *Blogging.Model* projektu.

Zainstaluj narzędzia Entity Framework Core w projekcie uruchamiania migracji

Aby włączyć polecenia migracji EF Core w konsoli Menedżera pakietów, należy zainstalować pakiet narzędzi programu EF Core w aplikacji konsoli.

- **Narzędzia > Menedżer pakietów NuGet > Konsola Menedżera pakietów**

- Uruchom `Install-Package Microsoft.EntityFrameworkCore.Tools -ProjectName Blogging.Migrations.Startup`

Tworzenie początkowej migracji

Tworzenie początkowej migracji, określenie aplikacji konsoli jako projekt startowy.

- Uruchom `Add-Migration InitialCreate -StartupProject Blogging.Migrations.Startup`

To polecenie scaffolds migracji, który tworzy początkowy zestaw tabel bazy danych dla modelu danych.

Tworzenie projektu platformy uniwersalnej systemu Windows

- W **Eksploratorze rozwiązań**, kliknij prawym przyciskiem myszy rozwiązanie, a następnie wybierz **Dodaj > Nowy projekt**.
- Z menu po lewej stronie wybierz **zainstalowane > Visual C# > Windows Universal**.
- Wybierz **pusta aplikacja (Windows Universal)** szablonu projektu.
- Nadaj projektowi nazwę *Blogging.UWP* i kliknij przycisk **OK**

IMPORTANT

Co najmniej równa wersje docelowa i minimalna **Windows 10 Fall Creators Update (10.0; kompilacja 16299.0)**.

Poprzednie wersje systemu Windows 10 nie obsługują .NET Standard 2.0, która jest wymagana przez Entity Framework Core.

Dodaj kod, aby utworzyć bazę danych podczas uruchamiania aplikacji

Ponieważ baza danych ma zostać utworzony na urządzeniu, która jest uruchamiana aplikacja, należy dodać kod. Zastosuj wszelkie oczekujące migracji w lokalnej bazie danych podczas uruchamiania aplikacji. Przy pierwszym uruchomieniu aplikacji, to zajmie się tworzenie lokalnej bazy danych.

- Dodaj odwołanie do projektu z *Blogging.UWP* projekt *Blogging.Model* projektu.
- Otwórz *App.xaml.cs*.
- Dodaj wyróżniony kod, aby zastosować wszelkie oczekujące migracji.

```
using Blogging.Model;
using Microsoft.EntityFrameworkCore;
using System;
using Windows.ApplicationModel;
using Windows.ApplicationModel.Activation;
using Windows.UI.Xaml;
using Windows.UI.Xaml.Controls;
using Windows.UI.Xaml.Navigation;

namespace Blogging.UWP
{
    /// <summary>
    /// Provides application-specific behavior to supplement the default Application class.
    /// </summary>
    sealed partial class App : Application
    {
        /// <summary>
        /// Initializes the singleton application object. This is the first line of authored code
        /// executed, and as such is the logical equivalent of main() or WinMain().
        /// </summary>
        public App()
    }
}
```

```

{
    this.InitializeComponent();
    this.Suspending += OnSuspending;

    using (var db = new BloggingContext())
    {
        db.Database.Migrate();
    }
}

/// <summary>
/// Invoked when the application is launched normally by the end user. Other entry points
/// will be used such as when the application is launched to open a specific file.
/// </summary>
/// <param name="e">Details about the launch request and process.</param>
protected override void OnLaunched(LaunchActivatedEventArgs e)
{
    Frame rootFrame = Window.Current.Content as Frame;

    // Do not repeat app initialization when the Window already has content,
    // just ensure that the window is active
    if (rootFrame == null)
    {
        // Create a Frame to act as the navigation context and navigate to the first page
        rootFrame = new Frame();

        rootFrame.NavigationFailed += OnNavigationFailed;

        if (e.PreviousExecutionState == ApplicationExecutionState.Terminated)
        {
            //TODO: Load state from previously suspended application
        }

        // Place the frame in the current Window
        Window.Current.Content = rootFrame;
    }

    if (e.PrelaunchActivated == false)
    {
        if (rootFrame.Content == null)
        {
            // When the navigation stack isn't restored navigate to the first page,
            // configuring the new page by passing required information as a navigation
            // parameter
            rootFrame.Navigate(typeof(MainPage), e.Arguments);
        }
        // Ensure the current window is active
        Window.Current.Activate();
    }
}

/// <summary>
/// Invoked when Navigation to a certain page fails
/// </summary>
/// <param name="sender">The Frame which failed navigation</param>
/// <param name="e">Details about the navigation failure</param>
void OnNavigationFailed(object sender, NavigationFailedEventArgs e)
{
    throw new Exception("Failed to load Page " + e.SourcePageType.FullName);
}

/// <summary>
/// Invoked when application execution is being suspended. Application state is saved
/// without knowing whether the application will be terminated or resumed with the contents
/// of memory still intact.
/// </summary>
/// <param name="sender">The source of the suspend request.</param>
/// <param name="e">Details about the suspend request.</param>
private void OnSuspending(object sender, SuspendingEventArgs e)
{
}

```

```
        }

        var deferral = e.SuspendingOperation.GetDeferral();
        //TODO: Save application state and stop any background activity
        deferral.Complete();
    }
}
```

TIP

W przypadku zmiany modelu użycia `Add-Migration` polecenia do tworzenia szkieletu nową migrację do zastosowania odpowiednich zmian w bazie danych. Wszystkie oczekujące migracje zostaną zastosowane do lokalnej bazy danych na każdym urządzeniu, podczas uruchamiania aplikacji.

Używa programu EF Core `__EFMigrationsHistory` tabeli w bazie danych, aby śledzić migracje, które zostały już zastosowane do bazy danych.

Przy użyciu modelu danych

Można teraz używać programu EF Core przeprowadzić dostępu do danych.

- Otwórz `MainPage.xaml`.
- Dodawanie obsługi ładowania strony i UI zawartości, które przedstawiono poniżej

```
<Page
    x:Class="Blogging.UWP.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:Blogging.UWP"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d"
    Loaded="Page_Loaded">

    <Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
        <StackPanel>
            <TextBox Name="NewBlogUrl"></TextBox>
            <Button Click="Add_Click">Add</Button>
            <ListView Name="Blogs">
                <ListView.ItemTemplate>
                    <DataTemplate>
                        <TextBlock Text="{Binding Url}" />
                    </DataTemplate>
                </ListView.ItemTemplate>
            </ListView>
        </StackPanel>
    </Grid>
</Page>
```

Teraz Dodaj kod, aby Podłączanie do interfejsu użytkownika z bazą danych

- Otwórz `MainPage.xaml.cs`.
- Dodaj wyróżniony kod z poniższej listy:

```

using Blogging.Model;
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Runtime.InteropServices.WindowsRuntime;
using Windows.Foundation;
using Windows.Foundation.Collections;
using Windows.UI.Xaml;
using Windows.UI.Xaml.Controls;
using Windows.UI.Xaml.Controls.Primitives;
using Windows.UI.Xaml.Data;
using Windows.UI.Xaml.Input;
using Windows.UI.Xaml.Media;
using Windows.UI.Xaml.Navigation;

// The Blank Page item template is documented at https://go.microsoft.com/fwlink/?LinkId=402352&clcid=0x409

namespace Blogging.UWP
{
    /// <summary>
    /// An empty page that can be used on its own or navigated to within a Frame.
    /// </summary>
    public sealed partial class MainPage : Page
    {
        public MainPage()
        {
            this.InitializeComponent();
        }

        private void Page_Loaded(object sender, RoutedEventArgs e)
        {
            using (var db = new BloggingContext())
            {
                Blogs.ItemsSource = db.Blogs.ToList();
            }
        }

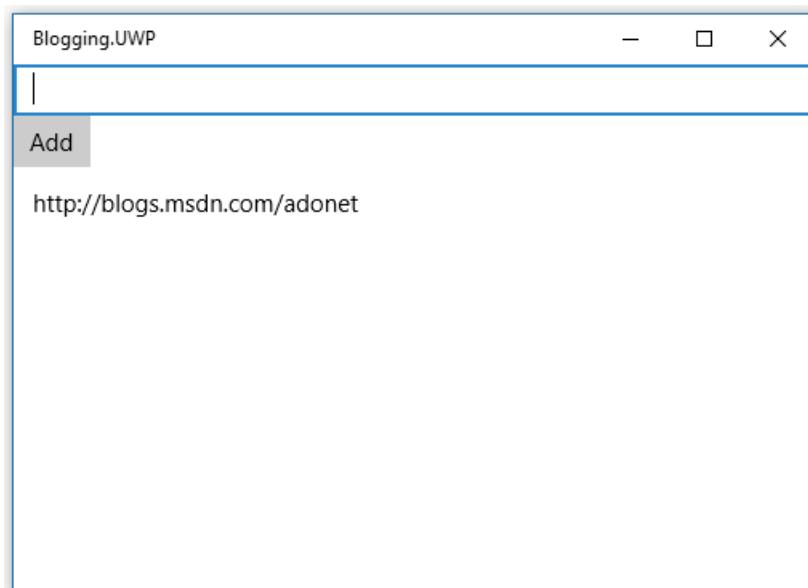
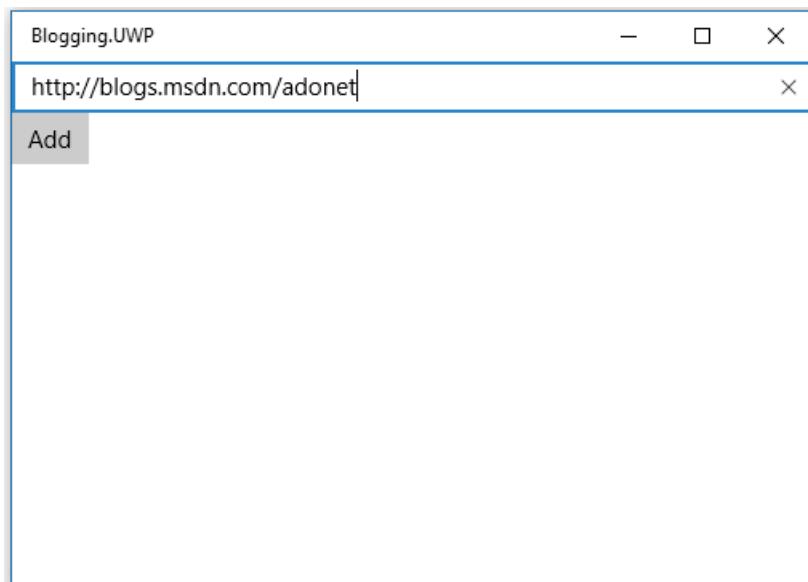
        private void Add_Click(object sender, RoutedEventArgs e)
        {
            using (var db = new BloggingContext())
            {
                var blog = new Blog { Url = NewBlogUrl.Text };
                db.Blogs.Add(blog);
                db.SaveChanges();

                Blogs.ItemsSource = db.Blogs.ToList();
            }
        }
    }
}

```

Teraz można uruchomić aplikacji, aby zobaczyć go w działaniu.

- W **Eksploratora rozwiązań**, kliknij prawym przyciskiem myszy *Blogging.UWP* projektu, a następnie wybierz pozycję **Wdroż**.
 - Ustaw *Blogging.UWP* jako projekt startowy.
 - **Debuguj > Uruchom bez debugowania**
- Aplikacja tworzy i uruchamia.
- Wprowadź adres URL, a następnie kliknij przycisk **Dodaj** przycisku



Tada! Masz teraz platformę Entity Framework Core uruchomioną prostą aplikacją platformy uniwersalnej systemu Windows.

Następne kroki

Uzyskać zgodności i wydajności, którą należy wiedzieć podczas korzystania z programu EF Core przy użyciu platformy uniwersalnej systemu Windows, zobacz [implementacji platformy .NET obsługiwanych przez platformę EF Core](#).

Zapoznaj się z innymi artykułami w tej dokumentacji, aby dowiedzieć się więcej na temat funkcji platformy Entity Framework Core.

Wprowadzenie do programu EF Core w programie .NET Framework

06.11.2018 • 2 minutes to read • [Edit Online](#)

Przedstawione samouczki nie wymagają wcześniejszej wiedzy dotyczącej korzystania z programów Entity Framework Core oraz Visual Studio. Pozwolą na przejście krok po kroku przez proces tworzenia prostej aplikacji do konsoli programu .NET Framework, która wykonuje kwerendę i zapisuje dane z bazy danych. Możesz skorzystać z samouczka dotyczącego tworzenia modelu w oparciu o istniejącą bazę danych lub dotyczącego tworzenia bazy danych na podstawie modelu.

Metody poznane z samouczków możesz wykorzystać w dowolnej aplikacji, która jest przeznaczona dla .NET Framework, w tym WPF oraz WinForms.

NOTE

Samouczki i towarzyszące im przykłady zostały zaktualizowane, aby korzystać z programu EF Core w wersji 2.1. Jednak w większości przypadków powinno być możliwe stworzenie aplikacji kompatybilnej z poprzednimi wersjami, wprowadzając zaledwie minimalne modyfikacje.

Wprowadzenie do programu EF Core w programie .NET Framework za pomocą nowej bazy danych

28.08.2018 • 5 minutes to read • [Edit Online](#)

W tym samouczku utworzysz aplikację konsolową, która wykonuje dostęp do podstawowych danych względem bazy danych programu Microsoft SQL Server przy użyciu platformy Entity Framework. Migracje umożliwia tworzenie bazy danych z modelu.

[Wyświetlanie przykładowych w tym artykule w witrynie GitHub.](#)

Wymagania wstępne

- [Visual Studio 2017 w wersji 15.7 lub nowszej](#)

Tworzenie nowego projektu

- Otwórz program Visual Studio 2017
- **Plik > Nowy > Projekt...**
- Z menu po lewej stronie wybierz **zainstalowane > Visual C# > Windows Desktop**
- Wybierz **Aplikacja konsoli (.NET Framework)** szablonu projektu
- Upewnij się, że projekt jest ukierunkowany **platformy .NET Framework 4.6.1** lub nowszej
- Nadaj projektowi nazwę *ConsoleApp.NewDb* i kliknij przycisk **OK**

Instalowanie programu Entity Framework

Aby korzystać z programu EF Core, należy zainstalować pakiet dla dostawców bazy danych, który ma pod kątem. Ten samouczek używa programu SQL Server. Aby uzyskać listę dostępnych dostawców zobacz [dostawcy baz danych](#).

- Narzędzia > Menedżer pakietów NuGet > Konsola Menedżera pakietów
- Uruchom `Install-Package Microsoft.EntityFrameworkCore.SqlServer`

W dalszej części tego samouczka użyjesz niektórych narzędzi Entity Framework Tools do obsługi bazy danych. Więc Zainstaluj również pakiet narzędzi.

- Uruchom `Install-Package Microsoft.EntityFrameworkCore.Tools`

Tworzenie modelu

Teraz nadszedł czas, do definiowania klas kontekstu i jednostek, które tworzą model.

- **Projekt > Dodaj klasę...**
- Wprowadź *Model.cs* jako nazwę i kliknij przycisk **OK**
- Zastąp zawartość pliku następującym kodem

```
using Microsoft.EntityFrameworkCore;
using System.Collections.Generic;

namespace ConsoleApp.NewDb
{
    public class BloggingContext : DbContext
    {
        public DbSet<Blog> Blogs { get; set; }
        public DbSet<Post> Posts { get; set; }

        protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
        {
            optionsBuilder.UseSqlServer(@"Server=
(localdb)\mssqllocaldb;Database=EFGetStarted.ConsoleApp.NewDb;Trusted_Connection=True;");
        }
    }

    public class Blog
    {
        public int BlogId { get; set; }
        public string Url { get; set; }

        public List<Post> Posts { get; set; }
    }

    public class Post
    {
        public int PostId { get; set; }
        public string Title { get; set; }
        public string Content { get; set; }

        public int BlogId { get; set; }
        public Blog Blog { get; set; }
    }
}
```

TIP

W rzeczywistej aplikacji można będzie umieścić każda klasa w oddzielnym pliku, a parametry połączenia w zmiennej środowisku lub pliku konfiguracji. Dla uproszczenia wszystko jest w pliku pojedynczego kodu na potrzeby tego samouczka.

Tworzenie bazy danych

Teraz, gdy model, można użyć migracje utworzyć bazę danych.

- **Narzędzia > Menedżer pakietów NuGet > Konsola Menedżera pakietów**
- Uruchom `Add-Migration InitialCreate` do tworzenia szkieletu migracji, aby utworzyć początkowy zestaw tabel dla modelu.
- Uruchom `Update-Database` zastosować nową migrację do bazy danych. Ponieważ baza danych nie istnieje, zostanie utworzony, przed zastosowaniem migracji.

TIP

Jeśli wprowadzisz zmiany w modelu, można użyć `Add-Migration` polecenia do tworzenia szkieletu nowej migracji w celu sprawdzenia odpowiedni schemat zmienia się z bazą danych. Po zaznaczeniu tej opcji utworzony szkielet kodu (i wprowadzone wymagane zmiany), można użyć `Update-Database` polecenie, aby zastosować zmiany do bazy danych.

Używa EF `__EFMigrationsHistory` tabeli w bazie danych, aby śledzić migracje, które zostały już zastosowane do bazy danych.

Użyj modelu

Można teraz używać modelu przeprowadzić dostępu do danych.

- Otwórz `Program.cs`
- Zastąp zawartość pliku następującym kodem

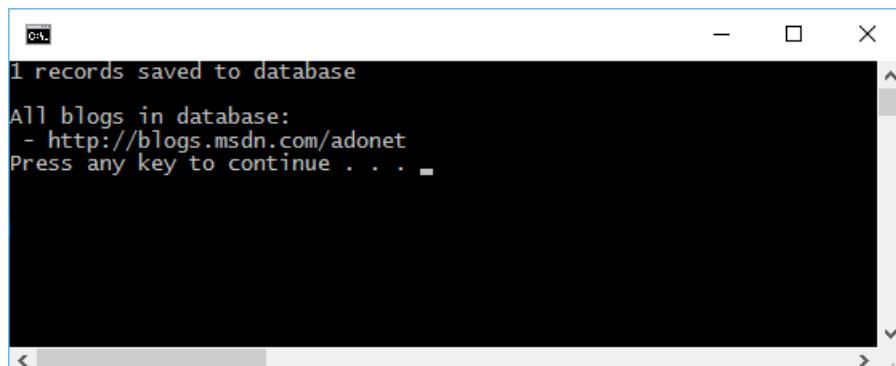
```
using System;

namespace ConsoleApp.NewDb
{
    class Program
    {
        static void Main(string[] args)
        {
            using (var db = new BloggingContext())
            {
                db.Blogs.Add(new Blog { Url = "http://blogs.msdn.com/adonet" });
                var count = db.SaveChanges();
                Console.WriteLine("{0} records saved to database", count);

                Console.WriteLine();
                Console.WriteLine("All blogs in database:");
                foreach (var blog in db.Blogs)
                {
                    Console.WriteLine(" - {0}", blog.Url);
                }
            }
        }
    }
}
```

- **Debuguj > Uruchom bez debugowania**

Zobaczysz, że blogami jest zapisywany w bazie danych, a następnie szczegółowe informacje o wszystkich blogów są drukowane w konsoli.



Dodatkowe zasoby

- [EF Core w programie .NET Framework przy użyciu istniejącej bazy danych](#)
- [EF Core na platformie .NET Core za pomocą nowej bazy danych — bazy danych SQLite — samouczek](#)
programu EF konsoli dla wielu platform.

Wprowadzenie do programu EF Core w programie .NET Framework przy użyciu istniejącej bazy danych

16.11.2018 • 6 minutes to read • [Edit Online](#)

W tym samouczku utworzysz aplikację konsolową, która wykonuje dostęp do podstawowych danych względem bazy danych programu Microsoft SQL Server przy użyciu platformy Entity Framework. Aby utworzyć model Entity Framework odtwarzanie istniejącej bazy danych.

[Wyświetlanie przykładowych w tym artykule w witrynie GitHub.](#)

Wymagania wstępne

- [Visual Studio 2017 w wersji 15.7 lub nowszej](#)

Utwórz bazę danych do obsługi blogów

W tym samouczku **do obsługi blogów** bazy danych w wystąpieniu LocalDb jako istniejącej bazy danych. Jeśli masz już utworzoną **do obsługi blogów** bazy danych jako część innego samouczek, należy pominąć tę procedurę.

- Otwórz program Visual Studio
- **Narzędzia > nawiązać połączenie z bazą danych...**
- Wybierz **programu Microsoft SQL Server** i kliknij przycisk **Kontynuuj**
- Wprowadź **(localdb) \mssqllocaldb** jako **nazwy serwera**
- Wprowadź **wzorca** jako **Nazwa bazy danych** i kliknij przycisk **OK**
- Baza danych master jest teraz wyświetlany w obszarze **połączeń danych** w **Eksploratora serwera**
- Kliknij prawym przyciskiem myszy w bazie danych w **Eksploratora serwera** i wybierz **nowe zapytanie**
- Skopiuj skrypt w edytorze zapytań
- Kliknij prawym przyciskiem myszy w edytorze zapytań, a następnie wybierz pozycję **wykonania**

```

CREATE DATABASE [Blogging];
GO

USE [Blogging];
GO

CREATE TABLE [Blog] (
    [BlogId] int NOT NULL IDENTITY,
    [Url] nvarchar(max) NOT NULL,
    CONSTRAINT [PK_Blog] PRIMARY KEY ([BlogId])
);
GO

CREATE TABLE [Post] (
    [PostId] int NOT NULL IDENTITY,
    [BlogId] int NOT NULL,
    [Content] nvarchar(max),
    [Title] nvarchar(max),
    CONSTRAINT [PK_Post] PRIMARY KEY ([PostId]),
    CONSTRAINT [FK_Post_Blog_BlogId] FOREIGN KEY ([BlogId]) REFERENCES [Blog] ([BlogId]) ON DELETE CASCADE
);
GO

INSERT INTO [Blog] (Url) VALUES
('http://blogs.msdn.com/dotnet'),
('http://blogs.msdn.com/webdev'),
('http://blogs.msdn.com/visualstudio')
GO

```

Tworzenie nowego projektu

- Otwórz program Visual Studio 2017
- **Plik > Nowy > Projekt...**
- Z menu po lewej stronie wybierz **zainstalowane > Visual C# > Windows Desktop**
- Wybierz **Aplikacja konsoli (.NET Framework)** szablonu projektu
- Upewnij się, że projekt jest ukierunkowany **platformy .NET Framework 4.6.1** lub nowszej
- Nadaj projektowi nazwę *ConsoleApp.ExistingDb* i kliknij przycisk **OK**

Instalowanie programu Entity Framework

Aby korzystać z programu EF Core, należy zainstalować pakiet dla dostawców bazy danych, który ma pod kątem. Ten samouczek używa programu SQL Server. Aby uzyskać listę dostępnych dostawców zobacz [dostawcy baz danych](#).

- **Narzędzia > Menedżer pakietów NuGet > Konsola Menedżera pakietów**
- Uruchom `Install-Package Microsoft.EntityFrameworkCore.SqlServer`

W następnym kroku użyjesz niektórych narzędzi Entity Framework Tools do odtworzenia bazy danych. Więc Zainstaluj również pakiet narzędzi.

- Uruchom `Install-Package Microsoft.EntityFrameworkCore.Tools`

Odtwarzanie modelu

Teraz nadszedł czas na tworzenie modelu platformy EF, w oparciu o istniejącą bazę danych.

- **Narzędzia -> pakietu NuGet Manager -> Konsola Menedżera pakietów**

- Uruchom następujące polecenie, aby utworzyć model z istniejącej bazy danych

```
Scaffold-DbContext "Server=(localdb)\mssqllocaldb;Database=Blogging;Trusted_Connection=True;"  
Microsoft.EntityFrameworkCore.SqlServer
```

TIP

Można określić tabel w celu wygenerowania jednostki dla, dodając `-Tables` argument polecenia powyżej. Na przykład `-Tables Blog,Post`.

Proces odtwarzania utworzony klas jednostek (`Blog` i `Post`) i pochodnej kontekstu (`BloggingContext`) na podstawie schematu istniejącej bazy danych.

Klasy jednostki są proste obiekty języka C#, które reprezentują będziesz zapytań i zapisywanie danych. Poniżej przedstawiono `Blog` i `Post` klas jednostek:

```
using System;  
using System.Collections.Generic;  
  
namespace ConsoleApp.ExistingDb  
{  
    public partial class Blog  
    {  
        public Blog()  
        {  
            Post = new HashSet<Post>();  
        }  
  
        public int BlogId { get; set; }  
        public string Url { get; set; }  
  
        public ICollection<Post> Post { get; set; }  
    }  
}
```

```
using System;  
using System.Collections.Generic;  
  
namespace ConsoleApp.ExistingDb  
{  
    public partial class Post  
    {  
        public int PostId { get; set; }  
        public int BlogId { get; set; }  
        public string Content { get; set; }  
        public string Title { get; set; }  
  
        public Blog Blog { get; set; }  
    }  
}
```

TIP

Aby włączyć ładowanie z opóźnieniem, należy wybrać właściwości nawigacji `virtual` (`Blog.Post` i `Post.Blog`).

Kontekst reprezentuje sesję z bazą danych. Zawiera metody, które służy do wykonywania zapytań i Zapisz wystąpień klas jednostek.

```
using System;
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Metadata;

namespace ConsoleApp.ExistingDb
{
    public partial class BloggingContext : DbContext
    {
        public BloggingContext()
        {

        }

        public BloggingContext(DbContextOptions<BloggingContext> options)
            : base(options)
        {
        }

        public virtual DbSet<Blog> Blog { get; set; }
        public virtual DbSet<Post> Post { get; set; }

        protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
        {
            if (!optionsBuilder.IsConfigured)
            {
#warning To protect potentially sensitive information in your connection string, you should move it out of
source code. See http://go.microsoft.com/fwlink/?LinkId=723263 for guidance on storing connection strings.
                optionsBuilder.UseSqlServer("Server=
(localdb)\mssqllocaldb;Database=Blogging;Trusted_Connection=True;");
            }
        }

        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            modelBuilder.Entity<Blog>(entity =>
            {
                entity.Property(e => e.Url).IsRequired();
            });

            modelBuilder.Entity<Post>(entity =>
            {
                entity.HasOne(d => d.Blog)
                    .WithMany(p => p.Post)
                    .HasForeignKey(d => d.BlogId);
            });
        }
    }
}
```

Użyj modelu

Można teraz używać modelu przeprowadzić dostępu do danych.

- Otwórz *Program.cs*
- Zastąp zawartość pliku następującym kodem

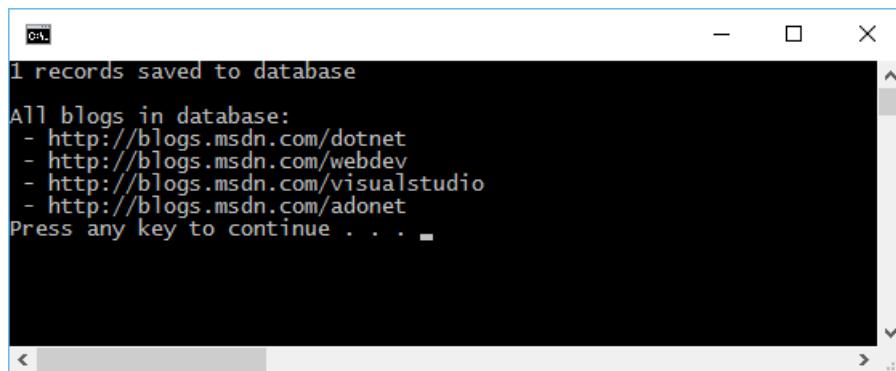
```
using System;

namespace ConsoleApp.ExistingDb
{
    class Program
    {
        static void Main(string[] args)
        {
            using (var db = new BloggingContext())
            {
                db.Blog.Add(new Blog { Url = "http://blogs.msdn.com/adonet" });
                var count = db.SaveChanges();
                Console.WriteLine("{0} records saved to database", count);

                Console.WriteLine();
                Console.WriteLine("All blogs in database:");
                foreach (var blog in db.Blog)
                {
                    Console.WriteLine("- {0}", blog.Url);
                }
            }
        }
    }
}
```

- Debuguj > Uruchom bez debugowania

Zobaczysz, że blogami jest zapisywany w bazie danych, a następnie szczegółowe informacje o wszystkich blogów są drukowane w konsoli.



```
1 records saved to database
All blogs in database:
- http://blogs.msdn.com/dotnet
- http://blogs.msdn.com/webdev
- http://blogs.msdn.com/visualstudio
- http://blogs.msdn.com/adonet
Press any key to continue . . .
```

Następne kroki

Aby uzyskać więcej informacji na temat sposobu tworzenia szkieletu klasy kontekstu i jednostek, zobacz następujące artykuły:

- [Odtwarzanie](#)
- [Entity Framework Core odnoszą się narzędzia — interfejs wiersza poleceń platformy .NET](#)
- [Entity Framework Core odnoszą się narzędzia — Konsola Menedżera pakietów](#)

Parametry połączenia

29.10.2018 • 3 minutes to read • [Edit Online](#)

Większość dostawców bazy danych wymaga pewnego rodzaju parametry połączenia do łączenia z bazą danych. Czasami te parametry połączenia zawiera poufne informacje, które muszą być chronione. Również może być konieczna zmiana parametrów połączenia, ponieważ przenoszenie aplikacji między środowiskami, takie jak programowania, testowania i produkcji.

Aplikacji .NET framework

Aplikacji .NET framework, takich jak WinForms, WPF, konsola i platformy ASP.NET 4 ma połączenie i przetestowanej wzorzec ciągu. Parametry połączenia należy dodać kod do pliku App.config aplikacji (Web.config, jeśli używasz programu ASP.NET). Jeśli parametry połączenia zawiera poufne informacje, takie jak nazwa użytkownika i hasło, chronić zawartość przy użyciu pliku konfiguracji [konfiguracji chronionej](#).

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>

    <connectionStrings>
        <add name="BloggingDatabase"
            connectionString="Server=(localdb)\mssqllocaldb;Database=Blogging;Trusted_Connection=True;" />
    </connectionStrings>
</configuration>
```

TIP

`providerName` Ustawienie nie jest wymagane na parametry połączenia programu EF Core przechowywane w pliku App.config, ponieważ dostawca bazy danych jest skonfigurowana za pomocą kodu.

Możesz odczytywać parametry połączenia za pomocą `ConfigurationManager` interfejsu API w sieci w kontekście `OnConfiguring` metody. Może być konieczne dodanie odwołania do `System.Configuration` zestawu struktury, aby można było używać tego interfejsu API.

```
public class BloggingContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }
    public DbSet<Post> Posts { get; set; }

    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    {

        optionsBuilder.UseSqlServer(ConfigurationManager.ConnectionStrings["BloggingDatabase"].ConnectionString);
    }
}
```

Platforma uniwersalna systemu Windows (UWP)

Parametry połączenia w aplikacji platformy uniwersalnej systemu Windows są zazwyczaj połączenia bazy danych SQLite, po prostu określający nazwę pliku lokalnego. Są zazwyczaj nie zawierają informacji poufnych, a nie powinny być można zmienić, ponieważ aplikacja jest wdrażana. W efekcie te parametry połączenia są zwykle wystarczające pozostać w kodzie, jak pokazano poniżej. Jeśli chcesz przenieść je poza kod to platformy

uniwersalnej systemu Windows obsługuje pojęcie ustawień, zobacz [ustawienia aplikacji w sekcji dokumentacji platformy uniwersalnej systemu Windows](#) Aby uzyskać szczegółowe informacje.

```
public class BloggingContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }
    public DbSet<Post> Posts { get; set; }

    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    {
        optionsBuilder.UseSqlite("Data Source=blogging.db");
    }
}
```

ASP.NET Core

W programie ASP.NET Core jest bardzo elastyczny system konfiguracji i parametry połączenia mogą być przechowywane w `appsettings.json`, zmienną środowiskową, magazynu wpisów tajnych użytkownika lub innego źródła konfiguracji. Zobacz [konfiguracji w sekcji dokumentacji platformy ASP.NET Core](#) Aby uzyskać więcej informacji. W poniższym przykładzie przedstawiono parametry połączenia przechowywana w `appsettings.json`.

```
{
  "ConnectionStrings": {
    "BloggingDatabase": "Server=(localdb)\\mssqllocaldb;Database=EFGetStarted.ConsoleApp.NewDb;Trusted_Connection=True;"
  },
}
```

Kontekst jest zazwyczaj skonfigurowany w `Startup.cs` przy użyciu parametrów połączenia jest odczytywana z konfiguracji. Uwaga `GetConnectionString()` metoda szuka wartości konfiguracji, w których kluczem jest `ConnectionStrings:<connection string name>`. Należy zaimportować [Microsoft.Extensions.Configuration](#) przestrzeni nazw w celu używania tej metody rozszerzenia.

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddDbContext<BloggingContext>(options =>
        options.UseSqlServer(Configuration.GetConnectionString("BloggingDatabase")));
}
```

Rejestrowanie

25.10.2018 • 2 minutes to read • [Edit Online](#)

TIP

Przykład użyty w tym artykule można zobaczyć w witrynie GitHub.

Aplikacje platformy ASP.NET Core

EF Core automatycznie integruje się z mechanizmami rejestracji programu ASP.NET Core przy każdym `AddDbContext` lub `AddDbContextPool` jest używany. W związku z tym, korzystając z platformy ASP.NET Core, rejestrowanie należy skonfigurować zgodnie z opisem w [dokumentacji platformy ASP.NET Core](#).

Inne aplikacje

EF Core rejestrowania obecnie wymaga element `ILoggerFactory`, która sama skonfigurowane z co najmniej jeden `ILoggerProvider`. Typowe dostawcy są dostarczane w następujących pakietów:

- [Microsoft.Extensions.Logging.Console](#): rejestratora konsoli proste.
- [Microsoft.Extensions.Logging.AzureAppServices](#): Obsługa usługi Azure App Services "Dzienniki diagnostyczne" i "Log strumienia" funkcji.
- [Microsoft.Extensions.Logging.Debug](#): dzienniki, aby monitor debugera za pomocą `System.Diagnostics.Debug.WriteLine()`.
- [Microsoft.Extensions.Logging.EventLog](#): dzienniki w dzienniku zdarzeń Windows.
- [Microsoft.Extensions.Logging.EventSource](#): obsługuje `EventSource/EventListener`.
- [Microsoft.Extensions.Logging.TraceSource](#): dzienniki, aby za pomocą `System.Diagnostics.TraceSource.TraceEvent()` odbiornik śledzenia.

Po zainstalowaniu odpowiednich pakietów aplikacji powinien utworzyć pojedyncze/globalnego wystąpienia `LoggerFactory`. Na przykład za pomocą rejestratora konsoli:

```
public static readonly LoggerFactory MyLoggerFactory  
    = new LoggerFactory(new[] {new ConsoleLoggerProvider(_,_ => true, true)});
```

To wystąpienie singleton/globalne następnie powinny być rejestrowane z programem EF Core na `DbContextOptionsBuilder`. Na przykład:

```
protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)  
=> optionsBuilder  
    .UseLoggerFactory(MyLoggerFactory) // Warning: Do not create a new ILoggerFactory instance each time  
    .UseSqlServer(  
        @"Server=(localdb)\mssqllocaldb;Database=EFLogging;Trusted_Connection=True;ConnectRetryCount=0");
```

WARNING

Jest to bardzo ważne jest, że aplikacji należy tworzyć nowe wystąpienie element `ILoggerFactory` dla każdego wystąpienia kontekstu. Ten sposób spowoduje przeciek pamięci i niską wydajnością.

Filtrowanie, co jest rejestrowane

Najprostszym sposobem filtrowania, co jest rejestrowane jest skonfigurowane podczas rejestrowania `ILoggerProvider`. Na przykład:

```
public static readonly LoggerFactory MyLoggerFactory
    = new LoggerFactory(new[]
    {
        new ConsoleLoggerProvider((category, level)
            => category == DbLoggerCategory.Database.Command.Name
                && level == LogLevel.Information, true)
    });
}
```

W tym przykładzie dziennik jest filtrowana w celu zwraca tylko wiadomości:

- w kategorii "Microsoft.EntityFrameworkCore.Database.Command"
- na poziomie "Informacje"

Dla platformy EF Core rejestratora kategorie są definiowane w `DbLoggerCategory` klasy, aby ułatwić znalezienie kategorii, ale one rozpoznać zwykłe ciągi.

Szczegółowe informacje na temat podstawowej infrastruktury rejestrowania można znaleźć w [dokumentacji rejestracji platformy ASP.NET Core](#).

Elastyczność połączenia

25.10.2018 • 8 minutes to read • [Edit Online](#)

Elastyczność połączenia automatycznie ponawia próbę polecenia bazy danych nie powiodło się. Funkcja może być używana z żadną bazą danych, podając "strategię wykonywania", która hermetyzuje logikę niezbędną do wykrywania błędów i ponów próbę wykonania polecenia. EF Core dostawców można podać strategię wykonywania dopasowane do swoich warunków błędów konkretnej bazy danych i zasady ponawiania optymalne.

Na przykład dostawca programu SQL Server zawiera strategię wykonywania, który jest w szczególny sposób dopasowane do programu SQL Server (w tym usługi SQL Azure). Ją rozpoznaje rodzaje wyjątków, które mogą być ponawiane i posiada odpowiednie ustawienia domyślne, aby uzyskać maksymalną liczbę ponownych prób, opóźnienie między ponownych prób.

Strategia wykonywania jest określony podczas konfigurowania opcji dla kontekstu. Jest to zazwyczaj w `OnConfiguring` metody pochodzącej kontekstu lub w `Startup.cs` dla aplikacji platformy ASP.NET Core.

```
protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
{
    optionsBuilder
        .UseSqlServer(
            @"Server=
(localdb)\mssqllocaldb;Database=EFMiscellaneous.ConnectionResiliency;Trusted_Connection=True;ConnectRetryCount=0",
            options => options.EnableRetryOnFailure());
}
```

Strategia wykonywania niestandardowych

Istnieje mechanizm do zarejestrowania strategii wykonywania niestandardowych samodzielnie, jeśli chcesz zmienić dowolne z ustawień domyślnych.

```
protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
{
    optionsBuilder
        .UseMyProvider(
            "<connection string>",
            options => options.ExecutionStrategy(...));
}
```

Strategii wykonywania i transakcji

Strategii wykonywania, który automatycznie ponawia próbę na błędy musi być w stanie odtworzyć każdej operacji w bloku ponawiania, która kończy się niepowodzeniem. Po włączeniu ponownych prób każdej operacji wykonywanych przy użyciu programu EF Core staje się własną wywoływaną operacją. Oznacza to, każde zapytanie i każde wywołanie `SaveChanges()` zostanie ponowione jako jednostki, jeśli wystąpi błąd przejściowy.

Jednakże jeżeli Twój kod inicjuje transakcję przy użyciu `BeginTransaction()` definiujesz własną grupę działań, które muszą być traktowane jako jednostka i wszystko wewnętrznych transakcji musi być odtwarzane ma miejsce awaria. Jeśli spróbujesz to zrobić, korzystając z strategii wykonywania, zostanie wyświetlony następujący wyjątek:

```
InvalidOperationException: Strategia wykonywania skonfigurowany "SqlServerRetryingExecutionStrategy" nie obsługuje transakcji zainicjowanej przez użytkownika. Strategia wykonywania zwróconych przez
```

"DbContext.Database.CreateExecutionStrategy()" umożliwia wykonywanie wszystkich operacji w transakcji jako jednostka z możliwością ponowienia próby.

To rozwiązanie jest ręcznie wywołać strategii wykonywania z delegatem reprezentującym wszystko, co ma zostać wykonana. Jeśli wystąpi błąd przejściowy, strategia wykonywania wywoła delegata ponownie.

```
using (var db = new BloggingContext())
{
    var strategy = db.Database.CreateExecutionStrategy();

    strategy.Execute(() =>
    {
        using (var context = new BloggingContext())
        {
            using (var transaction = context.Database.BeginTransaction())
            {
                context.Blogs.Add(new Blog {Url = "http://blogs.msdn.com/dotnet"});
                context.SaveChanges();

                context.Blogs.Add(new Blog {Url = "http://blogs.msdn.com/visualstudio"});
                context.SaveChanges();

                transaction.Commit();
            }
        });
    });
}
```

To podejście można również otoczenia transakcji.

```
using (var context1 = new BloggingContext())
{
    context1.Blogs.Add(new Blog { Url = "http://blogs.msdn.com/visualstudio" });

    var strategy = context1.Database.CreateExecutionStrategy();

    strategy.Execute(() =>
    {
        using (var context2 = new BloggingContext())
        {
            using (var transaction = new TransactionScope())
            {
                context2.Blogs.Add(new Blog { Url = "http://blogs.msdn.com/dotnet" });
                context2.SaveChanges();

                context1.SaveChanges();

                transaction.Complete();
            }
        });
    });
}
```

Błąd zatwierdzania transakcji i problem idempotentności

Ogólnie rzecz biorąc gdy wystąpi awaria połączenia bieżąca transakcja zostanie wycofana. Jednakże, jeśli połączenie zostało przerwane, gdy transakcja jest zwrócenia zatwierdzone Wynikowy stan transakcji jest nieznany. Zobacz ten [wpis w blogu](#) Aby uzyskać więcej informacji.

Domyślnie strategii wykonywania ponowi operację tak, jakby transakcja została wycofana, ale jeśli nie jest to wynikiem będzie wyjątek nowy stan bazy danych jest niezgodny lub może prowadzić do **uszkodzenie danych**

Jeśli Operacja nie zależą od określonego stanu, na przykład podczas wstawiania nowego wiersza przy użyciu automatycznego generowania wartości klucza.

Istnieje kilka sposobów, aby poradzić sobie z tym.

Opcja 1 — czy (prawie) nothing

Prawdopodobieństwo awarii połączenia podczas zatwierdzania transakcji jest niska, dlatego może być akceptowalne, aby aplikacja została właśnie się niepowodzeniem, jeśli rzeczywiście występuje ten problem.

Jednak należy unikać używania generowane przez magazyn kluczy w celu zapewnienia, że zamiast opcji dodawania zduplikowany wiersz jest zgłoszany wyjątek. Należy rozważyć użycie wartości identyfikatora GUID generowany przez klienta lub generator wartości po stronie klienta.

Opcja 2 — stan aplikacji ponownej komplikacji

1. Odrzucenie aktualnego `DbContext`.
2. Utwórz nową `DbContext` i przywrócenia stanu aplikacji z bazy danych.
3. Informuje użytkownika, że ostatnia operacja może nie zostały zakończone pomyślnie.

Opcja 3 — Dodawanie weryfikacji do stanu

Dla większości działań, które zmieniają stan bazy danych jest można dodać kod, który sprawdza, czy powiodła się. EF udostępnia metodę rozszerzenia, aby to ułatwić - `IExecutionStrategy.ExecuteInTransaction`.

Ta metoda rozpoczyna się i zatwierdzeń transakcji i akceptuje także funkcję `verifySucceeded` parametrów, które jest wywoływanie, gdy wystąpi błęd przejściowy podczas zatwierdzania transakcji.

```
using (var db = new BloggingContext())
{
    var strategy = db.Database.CreateExecutionStrategy();

    var blogToAdd = new Blog {Url = "http://blogs.msdn.com/dotnet"};
    db.Blogs.Add(blogToAdd);

    strategy.ExecuteInTransaction(db,
        operation: context =>
    {
        context.SaveChanges(acceptAllChangesOnSuccess: false);
    },
    verifySucceeded: context => context.Blogs.AsNoTracking().Any(b => b.BlogId == blogToAdd.BlogId));

    db.ChangeTracker.AcceptAllChanges();
}
```

NOTE

W tym miejscu `SaveChanges` jest wywoływana z `acceptAllChangesOnSuccess` równa `false`. Aby uniknąć zmieniania stanu `Blog` jednostki do `Unchanged`. Jeśli `SaveChanges` zakończy się pomyślnie. Dzięki temu, aby ponowić próbę wykonania tej samej operacji, jeśli zatwierdzenie zakończy się niepowodzeniem, a transakcja zostanie wycofana.

Opcja 4 — ręcznie śledzić transakcji

Jeśli musisz używać generowane przez magazyn kluczy lub potrzebujesz ogólny sposób obsługi błędów zatwierdzania, który nie są zależne od operacji wykonywanych każdej transakcji można przypisać Identyfikatora, który jest sprawdzany, jeśli zatwierdzenie zakończy się niepowodzeniem.

1. Dodaj tabelę w bazie danych używane do śledzenia stanu transakcji.
2. Wstaw wiersz do tabeli na początku każdej transakcji.
3. Jeśli połączenie nie powiedzie się podczas zatwierdzania, sprawdź, czy obecność odpowiedni wiersz w bazie

danych.

- Jeśli zatwierdzenie zakończy się pomyślnie, usuń odpowiedni wiersz w celu uniknięcia wzrostu tabeli.

```
using (var db = new BloggingContext())
{
    var strategy = db.Database.CreateExecutionStrategy();

    db.Blogs.Add(new Blog { Url = "http://blogs.msdn.com/dotnet" });

    var transaction = new TransactionRow { Id = Guid.NewGuid() };
    db.Transactions.Add(transaction);

    strategy.ExecuteInTransaction(db,
        operation: context =>
    {
        context.SaveChanges(acceptAllChangesOnSuccess: false);
    },
    verifySucceeded: context => context.Transactions.AsNoTracking().Any(t => t.Id == transaction.Id));

    db.ChangeTracker.AcceptAllChanges();
    db.Transactions.Remove(transaction);
    db.SaveChanges();
}
```

NOTE

Upewnij się, że kontekst użyty w celu weryfikacji ma zdefiniowany jako połączenie jest prawdopodobnie nastąpi ich awaria ponownie podczas weryfikacji w przypadku niepowodzenia podczas zatwierdzania transakcji strategii wykonywania.

Testowanie

28.08.2018 • 2 minutes to read • [Edit Online](#)

Można przetestować składników za pomocą coś, co przybliża z bazą danych rzeczywistych, bez konieczności operacji We/Wy bazy danych.

Istnieją dwie główne opcje w ten sposób:

- [Tryb w pamięci SQLite](#) pozwala na zapis efektywne testy względem dostawcy, który zachowuje się jak relacyjnej bazy danych.
- [Dostawca InMemory](#) jest uproszczone dostawcę, który ma minimalne zależności, ale nie zawsze zachowują się jak relacyjnej bazy danych.

Testowanie za pomocą SQLite

28.08.2018 • 4 minutes to read • [Edit Online](#)

Bazy danych SQLite ma trybie w pamięci, która pozwala na potrzeby pisania testów przeciwko relacyjnej bazy danych bez konieczności operacji bazy danych SQLite.

TIP

Można wyświetlić w tym artykule [przykładowe](#) w witrynie GitHub

Przykładowy scenariusz testowania

Należy wziąć pod uwagę następujące usługę, która umożliwia wykonywanie operacji związań z blogów kodu aplikacji. Używa wewnętrznie `DbContext` łączący się z bazą danych programu SQL Server. Należało by wymiany tego kontekstu, aby połączyć się z bazą danych SQLite w pamięci, możemy napisać efektywne testy dla tej usługi bez konieczności modyfikowania kodu lub wykonać dużo pracy, aby utworzyć test podwójnego kontekstu.

```
using System.Collections.Generic;
using System.Linq;

namespace BusinessLogic
{
    public class BlogService
    {
        private BloggingContext _context;

        public BlogService(BloggingContext context)
        {
            _context = context;
        }

        public void Add(string url)
        {
            var blog = new Blog { Url = url };
            _context.Blogs.Add(blog);
            _context.SaveChanges();
        }

        public IEnumerable<Blog> Find(string term)
        {
            return _context.Blogs
                .Where(b => b.Url.Contains(term))
                .OrderBy(b => b.Url)
                .ToList();
        }
    }
}
```

Przygotuj kontekstu

Należy unikać konfigurowania dwóch dostawców bazy danych

W testach ma zewnętrznie skonfigurować kontekst do użycia dostawcy InMemory. Jeśli konfigurujesz dostawcy bazy danych przez zastąpienie `OnConfiguring` w kontekście, następnie należy dodać niektórych kod warunkowy, aby upewnić się, można tylko skonfigurować dostawcy bazy danych, gdy nie została już skonfigurowana.

TIP

Jeśli używasz platformy ASP.NET Core, następnie nie należy tego kodu od dostawcy bazy danych skonfigurowano poza kontekstem (w pliku Startup.cs).

```
protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
{
    if (!optionsBuilder.IsConfigured)
    {
        optionsBuilder.UseSqlServer(@"Server=
(localdb)\mssqllocaldb;Database=EFProviders.InMemory;Trusted_Connection=True;ConnectRetryCount=0");
    }
}
```

Dodaj Konstruktor do testowania

Najprostszym sposobem, aby umożliwić testowanie z inną bazą danych jest zmodyfikowanie kontekstu ujawniać Konstruktor, który akceptuje `DbContextOptions<TContext>`.

```
public class BloggingContext : DbContext
{
    public BloggingContext()
    { }

    public BloggingContext(DbContextOptions<BloggingContext> options)
        : base(options)
    { }
```

TIP

`DbContextOptions<TContext>` informuje kontekstu, wszystkie jego ustawienia, takie jak której bazy danych, aby nawiązać połaczenie. Jest to ten sam obiekt, który jest wbudowana, uruchamiając metodę `OnConfiguring` w kontekście usługi.

Pisanie testów

Kluczem do testowania przy użyciu tego dostawcy jest możliwość opowiadania kontekstu do używania bazy danych SQLite i kontrolowanie zakresu bazy danych w pamięci. Zakres bazy danych jest kontrolowana przez otwierające i zamykające połączenia. Baza danych jest ograniczony do czasu trwania, że połączenie jest otwarte. Zazwyczaj chcesz czyste bazy danych dla każdej metody testowej.

```
using BusinessLogic;
using Microsoft.Data.Sqlite;
using Microsoft.EntityFrameworkCore;
using Microsoft.VisualStudio.TestTools.UnitTesting;
using System.Linq;

namespace TestProject.SQLite
{
    [TestClass]
    public class BlogServiceTests
    {
        [TestMethod]
        public void Add_writes_to_database()
        {
            // In-memory database only exists while the connection is open
            var connection = new SqliteConnection("DataSource=:memory:");
            connection.Open();
```

```

try
{
    var options = new DbContextOptionsBuilder<BloggingContext>()
        .UseSqlite(connection)
        .Options;

    // Create the schema in the database
    using (var context = new BloggingContext(options))
    {
        context.Database.EnsureCreated();
    }

    // Run the test against one instance of the context
    using (var context = new BloggingContext(options))
    {
        var service = new BlogService(context);
        service.Add("http://sample.com");
    }

    // Use a separate instance of the context to verify correct data was saved to database
    using (var context = new BloggingContext(options))
    {
        Assert.AreEqual(1, context.Blogs.Count());
        Assert.AreEqual("http://sample.com", context.Blogs.Single().Url);
    }
}
finally
{
    connection.Close();
}
}

[TestMethod]
public void Find_searches_url()
{
    // In-memory database only exists while the connection is open
    var connection = new SqliteConnection("DataSource=:memory:");
    connection.Open();

    try
    {
        var options = new DbContextOptionsBuilder<BloggingContext>()
            .UseSqlite(connection)
            .Options;

        // Create the schema in the database
        using (var context = new BloggingContext(options))
        {
            context.Database.EnsureCreated();
        }

        // Insert seed data into the database using one instance of the context
        using (var context = new BloggingContext(options))
        {
            context.Blogs.Add(new Blog { Url = "http://sample.com/cats" });
            context.Blogs.Add(new Blog { Url = "http://sample.com/catfish" });
            context.Blogs.Add(new Blog { Url = "http://sample.com/dogs" });
            context.SaveChanges();
        }

        // Use a clean instance of the context to run the test
        using (var context = new BloggingContext(options))
        {
            var service = new BlogService(context);
            var result = service.Find("cat");
            Assert.AreEqual(2, result.Count());
        }
    }
}
finally
{
}

```

```
        .----,
    {
        connection.Close();
    }
}
```

Testowanie za pomocą InMemory

28.08.2018 • 5 minutes to read • [Edit Online](#)

Dostawca InMemory jest przydatne, gdy chcesz przetestować składników za pomocą coś, co przybliża z bazą danych rzeczywistych, bez konieczności operacji bazy danych.

TIP

[Przykład](#) użyty w tym artykule można zobaczyć w witrynie GitHub.

InMemory nie jest relacyjnej bazy danych

Dostawcy baz danych programu EF Core nie trzeba być relacyjnych baz danych. InMemory została zaprojektowana jako bazy danych ogólnego przeznaczenia do testowania i nie jest przeznaczony do naśladowania relacyjnej bazy danych.

Niektóre przykłady to między innymi:

- InMemory pozwoli na zapisywanie danych, który mógłby naruszyć ograniczenia integralności referencyjnej w relacyjnej bazie danych.
- Jeśli używasz `DefaultValueSql(string)` właściwości w modelu jest interfejs API relacyjnej bazy danych i nie odniesie skutku przy uruchamianiu InMemory.
- [Współbieżność za pośrednictwem wiersza i znacznik czasu: wersja](#) (`[Timestamp]` lub `IsRowVersion`) nie jest obsługiwane. Nie `DbUpdateConcurrencyException` zostanie zgłoszony, jeśli aktualizacja jest wykonywane przy użyciu starego tokenu współbieżności.

TIP

Różnice te nie będą znaczenia, dla wielu celów testowych. Jednak jeśli chcesz przetestować względem elementu, który zachowuje się bardziej jak true relacyjnej bazy danych, rozważ użycie [trybie w pamięci z bazy danych SQLite](#).

Przykładowy scenariusz testowania

Należy wziąć pod uwagę następujące usługę, która umożliwia wykonywanie operacji związanych z blogów kodu aplikacji. Używa wewnętrznie `DbContext` łączący się z bazą danych programu SQL Server. Należałyby wymiany z tym kontekstem do nawiązania połączenia z bazą InMemory tak, aby firma Microsoft mogła zapisać efektywne testy dla tej usługi bez konieczności modyfikowania kodu lub wykonać dużo pracy, aby utworzyć test podwójnego kontekstu.

```

using System.Collections.Generic;
using System.Linq;

namespace BusinessLogic
{
    public class BlogService
    {
        private BloggingContext _context;

        public BlogService(BloggingContext context)
        {
            _context = context;
        }

        public void Add(string url)
        {
            var blog = new Blog { Url = url };
            _context.Blogs.Add(blog);
            _context.SaveChanges();
        }

        public IEnumerable<Blog> Find(string term)
        {
            return _context.Blogs
                .Where(b => b.Url.Contains(term))
                .OrderBy(b => b.Url)
                .ToList();
        }
    }
}

```

Przygotuj kontekstu

Należy unikać konfigurowania dwóch dostawców bazy danych

W testach ma zewnętrznie skonfigurować kontekst do użycia dostawcy InMemory. Jeśli konfigurujesz dostawcy bazy danych przez zastąpienie `OnConfiguring` w kontekście, następnie należy dodać niektórych kod warunkowy, aby upewnić się, można tylko skonfigurować dostawcy bazy danych, gdy nie została już skonfigurowana.

```

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
{
    if (!optionsBuilder.IsConfigured)
    {
        optionsBuilder.UseSqlServer(@"Server=
(localdb)\mssqllocaldb;Database=EFProviders.InMemory;Trusted_Connection=True;ConnectRetryCount=0");
    }
}

```

TIP

Jeśli używasz platformy ASP.NET Core, następnie nie należy tego kodu od dostawcy bazy danych jest już skonfigurowana poza kontekstem (w pliku Startup.cs).

Dodaj Konstruktor do testowania

Najprostszym sposobem, aby umożliwić testowanie z inną bazą danych jest zmodyfikowanie kontekst ujawniać Konstruktor, który akceptuje `DbContextOptions<TContext>`.

```
public class BloggingContext : DbContext
{
    public BloggingContext()
    { }

    public BloggingContext(DbContextOptions<BloggingContext> options)
        : base(options)
    { }
```

TIP

`DbContextOptions<TContext>` informuje kontekstu, wszystkie jego ustawienia, takie jak której bazy danych, aby nawiązać połączenie. Jest to ten sam obiekt, który jest wbudowana, uruchamiając metoda `OnConfiguring` w kontekście usługi.

Pisanie testów

Kluczem do testowania przy użyciu tego dostawcy jest możliwość opowiadania kontekst do użycia dostawcy `InMemory` i kontrolowanie zakresu bazy danych w pamięci. Zazwyczaj chcesz czyste bazy danych dla każdej metody testowej.

Oto przykład klasy testowej, korzystającej z bazy danych w pamięci. Każdej metody testowej Określa unikatową nazwę bazy danych, co oznacza, że każda metoda charakteryzuje się własną bazą danych w pamięci.

TIP

Aby użyć `.UseInMemoryDatabase()` metodę rozszerzenia, odwołanie do pakietu NuGet
`Microsoft.EntityFrameworkCore.InMemory`.

```

using BusinessLogic;
using Microsoft.EntityFrameworkCore;
using Microsoft.VisualStudio.TestTools.UnitTesting;
using System.Linq;

namespace TestProject.InMemory
{
    [TestClass]
    public class BlogServiceTests
    {
        [TestMethod]
        public void Add_writes_to_database()
        {
            var options = new DbContextOptionsBuilder<BloggingContext>()
                .UseInMemoryDatabase(databaseName: "Add_writes_to_database")
                .Options;

            // Run the test against one instance of the context
            using (var context = new BloggingContext(options))
            {
                var service = new BlogService(context);
                service.Add("http://sample.com");
            }

            // Use a separate instance of the context to verify correct data was saved to database
            using (var context = new BloggingContext(options))
            {
                Assert.AreEqual(1, context.Blogs.Count());
                Assert.AreEqual("http://sample.com", context.Blogs.Single().Url);
            }
        }

        [TestMethod]
        public void Find_searches_url()
        {
            var options = new DbContextOptionsBuilder<BloggingContext>()
                .UseInMemoryDatabase(databaseName: "Find_searches_url")
                .Options;

            // Insert seed data into the database using one instance of the context
            using (var context = new BloggingContext(options))
            {
                context.Blogs.Add(new Blog { Url = "http://sample.com/cats" });
                context.Blogs.Add(new Blog { Url = "http://sample.com/catfish" });
                context.Blogs.Add(new Blog { Url = "http://sample.com/dogs" });
                context.SaveChanges();
            }

            // Use a clean instance of the context to run the test
            using (var context = new BloggingContext(options))
            {
                var service = new BlogService(context);
                var result = service.Find("cat");
                Assert.AreEqual(2, result.Count());
            }
        }
    }
}

```

Konfigurowanie typu DbContext

03.11.2018 • 6 minutes to read • [Edit Online](#)

W tym artykule przedstawiono podstawowe wzorce dotyczące konfigurowania `DbContext` za pośrednictwem `DbContextOptions` nawiązać połączenia z bazą danych, używając określonego dostawcy programu EF Core i opcjonalnie zachowania.

Konfiguracja typu DbContext w czasie projektowania

EF Core z czasu projektowania narzędzi, takich jak [migracje](#) muszą mieć możliwość odnajdywania i utworzyć wystąpienie pracy `DbContext` typu, aby można było zbierać szczegółowe informacje dotyczące typów jednostek i sposobu mapowania ich na schemat bazy danych aplikacji. Ten proces może być automatyczna, tak dugo, jak łatwo można utworzyć narzędzie `DbContext` w taki sposób, że zostanie on skonfigurowany podobnie jak może zostać skonfigurowane w czasie wykonywania.

Podczas gdy wszelkie wzorzec, który dostarcza informacje o konfiguracji niezbędne do `DbContext` może pracować w czasie wykonywania, narzędzi, które wymagają przy użyciu `DbContext` w czasie projektowania może pracować tylko z ograniczonej liczby wzorców. Są one objęte bardziej szczegółowo w [czasie projektowania, tworzenia kontekstu](#) sekcji.

Konfigurowanie DbContextOptions

`DbContext` musi mieć instancję `DbContextOptions` aby można było wykonać wszelkie prace. `DbContextOptions` Wystąpienia niesie ze sobą informacje o konfiguracji takich jak:

- Dostawca bazy danych do użycia, zwykle wybrane przez wywołanie metody, takie jak `UseSqlServer` lub `UseSqlite`. Te metody rozszerzenia wymagają odpowiedni pakiet dostawcy, takich jak `Microsoft.EntityFrameworkCore.SqlServer` lub `Microsoft.EntityFrameworkCore.Sqlite`. Te metody są zdefiniowane w `Microsoft.EntityFrameworkCore` przestrzeni nazw.
- Wszelkie wymagane parametry lub identyfikator wystąpienia bazy danych zazwyczaj przekazywany jako argument do podanej powyżej metody wyboru dostawcy
- Żadnych selektorów poziom dostawcy, zachowanie opcjonalne, również są zazwyczaj powiązane wewnętrz wywołania metody wyboru dostawcy
- Wszelkie ogólne selektorów zachowanie programu EF Core, zwykle połączonymi w łańcuch po lub przed metodą selektor dostawcy

Poniższy przykład umożliwia skonfigurowanie `DbContextOptions` do używania dostawcy programu SQL Server, połączenie jest zawarty w `connectionString` zmienną, limit czasu polecenia poziom dostawcy i selektor zachowanie programu EF Core, która sprawia, że wszystkie zapytania wykonywane w `DbContext` śledzenia nie domyślnie:

```
optionsBuilder
    .UseSqlServer(connectionString, providerOptions=>providerOptions.CommandTimeout(60))
    .UseQueryTrackingBehavior(QueryTrackingBehavior.NoTracking);
```

NOTE

Dostawca selektora metody i innych metod selektor zachowanie wymienione powyżej są metody rozszerzenia na `DbContextOptions` lub klasy opcji właściwe dla dostawcy. Aby uzyskać dostęp do tych metod rozszerzenia może być konieczne przestrzeń nazw (zazwyczaj `Microsoft.EntityFrameworkCore`) w zakres i obejmują zależności dodatkowych pakietów w projekcie.

`DbContextOptions` Mogą być dostarczane do `DbContext` przez zastąpienie `OnConfiguring` metody lub zewnętrznie za pośrednictwem argumentu konstruktora.

Jeśli używane są obie, `OnConfiguring` ostatnio zastosowane i mogą zastąpić opcje przekazana do argumentu konstruktora.

Argument konstruktora

Kontekst kodu za pomocą konstruktora:

```
public class BloggingContext : DbContext
{
    public BloggingContext(DbContextOptions<BloggingContext> options)
        : base(options)
    { }

    public DbSet<Blog> Blogs { get; set; }
}
```

TIP

Podstawowy Konstruktor typu `DbContext` akceptuje także nieogólnego wersję `DbContextOptions`, ale przy użyciu wersji nieogólnego nie jest zalecane w przypadku aplikacji z wieloma typami kontekstu.

Kod aplikacji można zainicjować na podstawie argumentu konstruktora:

```
var optionsBuilder = new DbContextOptionsBuilder<BloggingContext>();
optionsBuilder.UseSqlite("Data Source=blog.db");

using (var context = new BloggingContext(optionsBuilder.Options))
{
    // do stuff
}
```

OnConfiguring

Kontekst kodu za pomocą `OnConfiguring`:

```
public class BloggingContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }

    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    {
        optionsBuilder.UseSqlite("Data Source=blog.db");
    }
}
```

Kod aplikacji, aby zainicjować `DbContext`, który używa `OnConfiguring`:

```
using (var context = new BloggingContext())
{
    // do stuff
}
```

TIP

To podejście nie jest przystosowany do testowania, chyba że próby docelowe pełnej bazy danych.

Przy użyciu iniekcji zależności typu DbContext

EF Core obsługuje korzystanie z `DbContext` z kontenera iniekcji zależności. Typu `DbContext` można dodać do kontenera usługi za pomocą `AddDbContext<TContext>` metody.

`AddDbContext<TContext>` spowoduje, że oba danego typu `DbContext` `TContext` i odpowiedni `DbContextOptions<TContext>` dostępne do iniekcji z kontenera usług.

Zobacz [więcej odczytu](#) poniżej dodatkowe informacje na temat iniekcji zależności.

Dodawanie `DbContext` do wstrzykiwania zależności:

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddDbContext<BloggingContext>(options => options.UseSqlite("Data Source=blog.db"));
}
```

To wymaga dodania [argumentu konstruktora](#) do danego typu `DbContext`, który akceptuje `DbContextOptions<TContext>`.

Kontekst kodu:

```
public class BloggingContext : DbContext
{
    public BloggingContext(DbContextOptions<BloggingContext> options)
        :base(options)
    { }

    public DbSet<Blog> Blogs { get; set; }
}
```

Kod aplikacji (w programie ASP.NET Core):

```
public class MyController
{
    private readonly BloggingContext _context;

    public MyController(BloggingContext context)
    {
        _context = context;
    }

    ...
}
```

Kod aplikacji (przy użyciu elementu `ServiceProvider` bezpośrednio, mniej typowe):

```
using (var context = serviceProvider.GetService<BlogginContext>())
{
    // do stuff
}

var options = serviceProvider.GetService<DbContextOptions<BlogginContext>>();
```

Odczytywanie więcej

- Odczyt [wprowadzenie do programu ASP.NET Core](#) Aby uzyskać więcej informacji na temat korzystania z programów EF z platformą ASP.NET Core.
- Odczyt [wstrzykiwanie zależności](#) Aby dowiedzieć się więcej o korzystaniu z DI.
- Odczyt [testowania](#) Aby uzyskać więcej informacji.

Tworzenie i konfigurowanie modelu

13.09.2018 • 2 minutes to read • [Edit Online](#)

Entity Framework używa zestawu Konwencji do zbudowania modelu oparte na kształt klas jednostek. Możesz określić dodatkowej konfiguracji w celu uzupełnienia i/lub zastąpić, co zostało wykryte przez Konwencję.

W tym artykule opisano konfiguracji, które mogą być stosowane do modelu, przeznaczone dla dowolnego magazynu danych i tych, które można zastosować w przypadku przeznaczone dla dowolnej relacyjnej bazy danych. Dostawców może też umożliwiać konfiguracji, które są specyficzne dla określonego magazynu danych. Dokumentację dotyczącą konfiguracji określonego dostawcy znaleźć [dostawcy baz danych](#) sekcji.

TIP

Można wyświetlić w tym artykule [przykładowe](#) w witrynie GitHub.

Użyj interfejsu API fluent, aby skonfigurować model

Można zastąpić `OnModelCreating` metodę w pochodnej kontekstu i użyj `ModelBuilder API` do skonfigurowania modelu. To jest najbardziej zaawansowane metody konfiguracji i umożliwia konfigurację można określić bez konieczności wprowadzania zmian w Twoich zajęciach jednostki. Konfiguracja interfejsu API Fluent ma najwyższy priorytet i spowoduje zastąpienie danych i konwencje adnotacji.

```
class MyContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Blog>()
            .Property(b => b.Url)
            .IsRequired();
    }
}
```

Umożliwia konfigurowanie modelu adnotacji danych

Atrybuty (znanych jako adnotacje danych) można zastosować także do klas i właściwości. Adnotacje danych spowoduje zastąpienie Konwencji, ale zostaną zastąpione przez interfejs Fluent API konfiguracji.

```
public class Blog
{
    public int BlogId { get; set; }
    [Required]
    public string Url { get; set; }
}
```

Uwzględnianie i wykluczanie typów

28.08.2018 • 2 minutes to read • [Edit Online](#)

W tym typu w modelu oznacza, że EF ma metadane o typ, który podejmie próbę odczytu i zapisu wystąpienia z i do bazy danych.

Konwencje

Zgodnie z Konwencją, typy, które są widoczne w `DbSet` właściwości kontekstu znajdują się w modelu. Ponadto, typy, które są wymienione w `OnModelCreating` metody dostępne są również. Ponadto wszystkie typy, które znajdują się przez rekursywnie Eksplorowanie właściwości nawigacji odnalezionych typów znajdują się również w modelu.

Na przykład w poniższym fragmencie kodu wszystkich trzech typów zostaną wykryte:

- `Blog` ponieważ jest ona uwidoczniona w `DbSet` właściwości w kontekście
- `Post` ponieważ jest odnalezione `Blog.Posts` właściwość nawigacji
- `AuditEntry` ponieważ są one wymienione w `OnModelCreating`

```
class MyContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<AuditEntry>();
    }
}

public class Blog
{
    public int BlogId { get; set; }
    public string Url { get; set; }

    public List<Post> Posts { get; set; }
}

public class Post
{
    public int PostId { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }

    public Blog Blog { get; set; }
}

public class AuditEntry
{
    public int AuditEntryId { get; set; }
    public string Username { get; set; }
    public string Action { get; set; }
}
```

Adnotacje danych

Korzystanie z adnotacji danych, aby wyłączyć typ z modelu.

```
public class Blog
{
    public int BlogId { get; set; }
    public string Url { get; set; }

    public BlogMetadata Metadata { get; set; }
}

[NotMapped]
public class BlogMetadata
{
    public DateTime LoadedFromDatabase { get; set; }
}
```

Interfejs Fluent API

Aby wyłączyć typ z modelu, można użyć Fluent API.

```
class MyContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Ignore<BlogMetadata>();
    }
}

public class Blog
{
    public int BlogId { get; set; }
    public string Url { get; set; }

    public BlogMetadata Metadata { get; set; }
}

public class BlogMetadata
{
    public DateTime LoadedFromDatabase { get; set; }
}
```

Uwzględnianie i wykluczanie właściwości

28.08.2018 • 2 minutes to read • [Edit Online](#)

W tym właściwości w modelu oznacza, że EF ma metadane dotyczące tej właściwości i podejmie próbę odczytu i zapisu wartości z i do bazy danych.

Konwencje

Zgodnie z Konwencją właściwości publicznej metody pobierającej i ustawiającej zostaną uwzględnione w modelu.

Adnotacje danych

Korzystanie z adnotacji danych, które mają zostać wykluczone z właściwością z modelu.

```
public class Blog
{
    public int BlogId { get; set; }
    public string Url { get; set; }

    [NotMapped]
    public DateTime LoadedFromDatabase { get; set; }
}
```

Interfejs Fluent API

Interfejs Fluent API umożliwia wykluczać właściwość z modelu.

```
class MyContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Blog>()
            .Ignore(b => b.LoadedFromDatabase);
    }
}

public class Blog
{
    public int BlogId { get; set; }
    public string Url { get; set; }

    public DateTime LoadedFromDatabase { get; set; }
}
```

Klucze (podstawowe)

28.08.2018 • 2 minutes to read • [Edit Online](#)

Klucz służy jako podstawowy identyfikator unikatowy dla każdego wystąpienia jednostki. Korzystając z relacyjnej bazy danych to mapuje do koncepcji *klucz podstawowy*. Można również skonfigurować unikatowy identyfikator, który nie jest kluczem podstawowym (zobacz [klucze alternatywne](#) Aby uzyskać więcej informacji).

Konwencje

Zgodnie z Konwencją, właściwość o nazwie `Id` lub `<type name>Id` zostaną skonfigurowane jako klucza jednostki.

```
class Car
{
    public string Id { get; set; }

    public string Make { get; set; }
    public string Model { get; set; }
}
```

```
class Car
{
    public string CarId { get; set; }

    public string Make { get; set; }
    public string Model { get; set; }
}
```

Adnotacje danych

Korzystanie z adnotacji danych, aby skonfigurować jedną właściwość jako klucz jednostki.

```
class Car
{
    [Key]
    public string LicensePlate { get; set; }

    public string Make { get; set; }
    public string Model { get; set; }
}
```

Interfejs Fluent API

Interfejs Fluent API umożliwia skonfigurowanie jednej właściwości klucza jednostki.

```
class MyContext : DbContext
{
    public DbSet<Car> Cars { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Car>()
            .HasKey(c => c.LicensePlate);
    }
}

class Car
{
    public string LicensePlate { get; set; }

    public string Make { get; set; }
    public string Model { get; set; }
}
```

Można również skonfigurować wiele właściwości, aby być kluczem jednostki (znanych jako klucz złożony) za pomocą Fluent interfejsu API. Klucze złożone można skonfigurować tylko przy użyciu interfejsu API Fluent — konwencje nigdy nie skonfiguruje klucz złożony i nie umożliwia adnotacje danych konfiguracji.

```
class MyContext : DbContext
{
    public DbSet<Car> Cars { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Car>()
            .HasKey(c => new { c.State, c.LicensePlate });
    }
}

class Car
{
    public string State { get; set; }
    public string LicensePlate { get; set; }

    public string Make { get; set; }
    public string Model { get; set; }
}
```

Generowane wartości

10.09.2018 • 7 minutes to read • [Edit Online](#)

Wzorce Generowanie wartości

Istnieją trzy wzorce Generowanie wartości, które mogą być używane dla właściwości:

- Generowanie wartości nie jest
- Dodaj wartość wygenerowaną
- Wartość wygenerowaną na dodawanie lub aktualizowanie

Generowanie wartości nie jest

Generowanie wartość nie oznacza, że zawsze będzie podać prawidłową wartość do zapisania w bazie danych. To prawidłową wartość można przypisać nowe jednostki, zanim zostaną one dodane do kontekstu.

Dodaj wartość wygenerowaną

Wartość wygenerowaną Dodaj oznacza, że wygenerowaną wartość dla nowych jednostek.

W zależności od używanego dostawcy bazy danych wartości mogą być generowane po stronie klienta lub w bazie danych EF. Jeśli wartość jest generowana przez bazę danych, następnie EF może przypisać wartości tymczasowej po dodaniu elementu do kontekstu. Tej wartości tymczasowej następnie zostanie zastąpiona przez wartości bazy danych, wygenerowane podczas `SaveChanges()`.

Jeśli dodasz jednostki do kontekstu, który ma wartość przypisana do właściwości EF będzie podejmować próbę Wstaw tę wartość, zamiast generować nową. Właściwość została uznana za wartości przypisanej, jeśli nie jest przypisana wartość domyślna CLR (`null` dla `string`, `0` dla `int`, `Guid.Empty` dla `Guid` itp.). Aby uzyskać więcej informacji, zobacz [jawne wartości dla wygenerowanych właściwości](#).

WARNING

Jak wartość jest generowana dla jednostek dodano będzie zależeć od używanego dostawcy bazy danych. Dostawcy baz danych może automatycznie skonfigurować Generowanie wartości dla niektórych typów właściwości, ale innych może być konieczne, należy ręcznie skonfigurować, jak jest generowany wartość.

Na przykład podczas korzystania z programu SQL Server, wartości będą automatycznie generowane dla `GUID` właściwości (przy użyciu programu SQL Server algorytm sekwencyjny identyfikatora GUID). Jednak jeśli określono `DateTime` właściwość jest generowany na dodać, a następnie należy skonfigurować sposób wartości do wygenerowania. Jednym ze sposobów, aby to zrobić, to aby skonfigurować domyślną wartość `GETDATE()`, zobacz [wartości domyślne](#).

Wartość wygenerowaną na dodawanie lub aktualizowanie

Wartość generowane przy dodawaniu lub aktualizacji oznacza, że nowa wartość jest generowany za każdym razem, gdy rekord zostanie zapisany (insert nebo update).

Podobnie jak `value generated on add`, jeśli określona wartość właściwości na nowo dodane wystąpienie jednostki, że wartość zostanie wstawiony zamiast wartości generowanych. Jest również możliwość określenia jawną wartość podczas aktualizacji. Aby uzyskać więcej informacji, zobacz [jawne wartości dla wygenerowanych właściwości](#).

WARNING

Jak wartość jest generowana dla dodanych i zaktualizowanych jednostek będzie zależeć od używanego dostawcy bazy danych. Dostawcy baz danych mogą automatycznie skonfigurować Generowanie wartości dla niektórych typów właściwości, podczas gdy inne będą wymagały ręcznego konfigurowania, jak jest generowany wartość.

Na przykład w przypadku korzystania z programu SQL Server `byte[]` właściwości, które są ustawione, tak jak w dodać lub zaktualizować i oznaczone jako tokeny współbieżności będzie instalacji z użyciem `rowversion` typ danych — tak, aby wartości, zostanie wygenerowany w bazie danych. Jednak jeśli określono `DateTime` właściwość jest generowany na dodać lub zaktualizować, a następnie należy skonfigurować sposób wartości do wygenerowania. Jednym ze sposobów, aby to zrobić, to aby skonfigurować domyślną wartość `GETDATE()` (zobacz [wartości domyślne](#)) do generowania wartości dla nowych wierszy. Następnie można użyć wyzwalacza bazy danych do generowania wartości podczas aktualizacji (na przykład następujący wyzwalacz przykład).

```
CREATE TRIGGER [dbo].[Blogs_UPDATE] ON [dbo].[Blogs]
    AFTER UPDATE
AS
BEGIN
    SET NOCOUNT ON;

    IF ((SELECT TRIGGER_NESTLEVEL()) > 1) RETURN;

    DECLARE @Id INT

    SELECT @Id = INSERTED.BlogId
    FROM INSERTED

    UPDATE dbo.Blogs
    SET LastUpdated = GETDATE()
    WHERE BlogId = @Id
END
```

Konwencje

Zgodnie z Konwencją kluczy podstawowych innych niż złożone typu short, int, long lub identyfikator Guid będzie instalacji mogły mieć wartości wygenerowane Dodaj. Wszystkie pozostałe właściwości będzie Instalatora z Generowanie nie wartość.

Adnotacje danych

Generowanie nie wartość (adnotacje danych)

```
public class Blog
{
    [DatabaseGenerated(DatabaseGeneratedOption.None)]
    public int BlogId { get; set; }
    public string Url { get; set; }
}
```

Wartość wygenerowano Dodaj (adnotacje danych)

```
public class Blog
{
    public int BlogId { get; set; }
    public string Url { get; set; }
    [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
    public DateTime Inserted { get; set; }
}
```

WARNING

Dzięki temu wystarczy wiedzieć, że wartości są generowane dla jednostek dodano, nie gwarantuje, że EF skonfiguruje konkretny mechanizm do generowania wartości EF. Zobacz [Dodaj wartość wygenerowano](#) sekcji, aby uzyskać więcej informacji.

Dodaj wartość wygenerowano lub aktualizacji (adnotacje danych)

```
public class Blog
{
    public int BlogId { get; set; }
    public string Url { get; set; }
    [DatabaseGenerated(DatabaseGeneratedOption.Computed)]
    public DateTime LastUpdated { get; set; }
}
```

WARNING

Dzięki temu wystarczy wiedzieć, że wartości są generowane dla jednostek dodane lub zaktualizowane, nie gwarantuje, że EF skonfiguruje konkretny mechanizm do generowania wartości EF. Zobacz [wartość wygenerowano dodania lub zaktualizowania](#) sekcji, aby uzyskać więcej informacji.

Interfejs Fluent API

Interfejs Fluent API umożliwia zmianianie wzorca Generowanie wartości danej właściwości.

Generowanie nie wartość (Fluent API)

```
modelBuilder.Entity<Blog>()
    .Property(b => b.BlogId)
    .ValueGeneratedNever();
```

Wartość wygenerowano Dodaj (Fluent API)

```
modelBuilder.Entity<Blog>()
    .Property(b => b.Inserted)
    .ValueGeneratedOnAdd();
```

WARNING

`ValueGeneratedOnAdd()` po prostu umożliwia EF wiedzieć, że wartości są generowane dla jednostek dodano, nie gwarantuje, że EF skonfiguruje konkretny mechanizm do generowania wartości. Zobacz [Dodaj wartość wygenerowano](#) sekcji, aby uzyskać więcej informacji.

Dodaj wartość wygenerowano lub aktualizacji (Fluent API)

```
modelBuilder.Entity<Blog>()
    .Property(b => b.LastUpdated)
    .ValueGeneratedOnAddOrUpdate();
```

WARNING

Dzięki temu wystarczy wiedzieć, że wartości są generowane dla jednostek dodane lub zaktualizowane, nie gwarantuje, że EF skonfiguruje konkretny mechanizm do generowania wartości EF. Zobacz [wartość wygenerowano dodania lub zaktualizowania](#) sekcji, aby uzyskać więcej informacji.

Wymagane i opcjonalne właściwości

28.08.2018 • 2 minutes to read • [Edit Online](#)

Właściwości są traktowane jako opcjonalne, jeśli jest on prawidłowy, aby mogła zawierać `null`. Jeśli `null` nie jest prawidłową wartością ma być przypisane do właściwości, a następnie jest on uznawany za jest właściwością wymaganą.

Konwencje

Zgodnie z Konwencją, właściwość, której typ CLR może zawierać wartości null zostanie skonfigurowany jako opcjonalny (`string`, `int?`, `byte[]` itp.). Właściwości, których typ CLR nie może zawierać wartości null zostanie skonfigurowany zgodnie z wymogami (`int`, `decimal`, `bool` itp.).

NOTE

Właściwość, której typ CLR nie może zawierać wartości null nie można skonfigurować jako opcjonalną. Właściwość będzie zawsze być brana pod uwagę wymagane przez program Entity Framework.

Adnotacje danych

Korzystanie z adnotacji danych, aby wskazać, że właściwość jest wymagana.

```
public class Blog
{
    public int BlogId { get; set; }
    [Required]
    public string Url { get; set; }
}
```

Interfejs Fluent API

Aby wskazać, że właściwość jest wymagana, można użyć interfejsu API Fluent.

```
class MyContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Blog>()
            .Property(b => b.Url)
            .IsRequired();
    }
}

public class Blog
{
    public int BlogId { get; set; }
    public string Url { get; set; }
}
```

Maksymalna długość

28.08.2018 • 2 minutes to read • [Edit Online](#)

Konfigurowanie maksymalnej długości stanowi wskazówkę z magazynem danych o typie danych odpowiednich dla danej właściwości. Maksymalna długość ma zastosowanie tylko do tablicy typów danych, takich jak `string` i `byte[]`.

NOTE

Platformy Entity Framework wykonaj wszelkie weryfikacji o maksymalnej długości przed przekazaniem do dostawcy. Jest magazynu dostawcy lub danych, sprawdź, czy jest to odpowiednie. Na przykład gdy przeznaczonych dla programu SQL Server, która przekracza maksymalną długość spowodują wyjątek jako typ danych kolumny źródłowej nie zezwoli nadmiarowych danych mają być przechowywane.

Konwencje

Zgodnie z Konwencją pozostało do dostawcy bazy danych, aby wybrać odpowiedni typ danych właściwości. Dla właściwości o długości dostawca bazy danych zazwyczaj wybierz typ danych, który umożliwia najdłuższy długość danych. Na przykład Microsoft SQL Server będzie używać `nvarchar(max)` dla `string` właściwości (lub `nvarchar(450)`). Jeśli kolumna jest używana jako klucz).

Adnotacje danych

Korzystanie z adnotacji danych, aby skonfigurować maksymalną długość dla właściwości. W tym przykładzie przeznaczonych dla programu SQL Server, spowodowałoby `nvarchar(500)` typ danych jest używany.

```
public class Blog
{
    public int BlogId { get; set; }
    [MaxLength(500)]
    public string Url { get; set; }
}
```

Interfejs Fluent API

Interfejs Fluent API umożliwiają skonfigurowanie maksymalnej długości dla właściwości. W tym przykładzie przeznaczonych dla programu SQL Server, spowodowałoby `nvarchar(500)` typ danych jest używany.

```
class MyContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Blog>()
            .Property(b => b.Url)
            .HasMaxLength(500);
    }
}

public class Blog
{
    public int BlogId { get; set; }
    public string Url { get; set; }
}
```

Tokeny współbieżności

28.08.2018 • 2 minutes to read • [Edit Online](#)

NOTE

Ta strona dokumenty, jak skonfigurować tokeny współbieżności. Zobacz [Obsługa konfliktów współbieżności](#) uzyskać szczegółowy opis sposobu działania kontroli współbieżności na programu EF Core i przykłady sposobów obsługi konfliktów współbieżności w aplikacji.

Właściwości skonfigurowane jako tokeny współbieżności są używane do implementowania kontroli optymistycznej współbieżności.

Konwencje

Zgodnie z Konwencją właściwości nigdy nie są skonfigurowane jako tokeny współbieżności.

Adnotacje danych

Korzystanie z adnotacji danych, aby skonfigurować właściwości jako tokenem współbieżności.

```
public class Person
{
    public int PersonId { get; set; }

    [ConcurrencyCheck]
    public string LastName { get; set; }

    public string FirstName { get; set; }
}
```

Interfejs Fluent API

Interfejs Fluent API umożliwia skonfigurowanie właściwości jako tokenem współbieżności.

```
class MyContext : DbContext
{
    public DbSet<Person> People { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Person>()
            .Property(p => p.LastName)
            .IsConcurrencyToken();
    }
}

public class Person
{
    public int PersonId { get; set; }
    public string LastName { get; set; }
    public string FirstName { get; set; }
}
```

Wersja sygnatury czasowej/wiersza

Sygnatura czasowa jest właściwością, gdzie nowa wartość jest generowana przez bazę danych, za każdym razem, gdy wstawieniu lub zaktualizowaniu wiersza. Właściwość jest również traktowane jako tokenem współbieżności. Dzięki temu uzyskasz wyjątek, jeśli osobę zmodyfikował wiersza, który próbujesz zaktualizować, ponieważ zapytania dla danych.

Jak odbywa się to zależy od używanego dostawcy bazy danych. Dla programu SQL Server sygnatura czasowa jest zazwyczaj używany na *byte[]* właściwość, która będzie można skonfigurować jako *ROWVERSION* kolumny w bazie danych.

Konwencje

Zgodnie z Konwencją właściwości nigdy nie są skonfigurowane jako sygnatury czasowych.

Adnotacje danych

Korzystanie z adnotacji danych, aby skonfigurować właściwości jako sygnaturę czasową.

```
public class Blog
{
    public int BlogId { get; set; }

    public string Url { get; set; }

    [Timestamp]
    public byte[] Timestamp { get; set; }
}
```

Interfejs Fluent API

Interfejs Fluent API umożliwia skonfigurowanie właściwości jako sygnaturę czasową.

```
class MyContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Blog>()
            .Property(p => p.Timestamp)
            .IsRowVersion();
    }
}

public class Blog
{
    public int BlogId { get; set; }
    public string Url { get; set; }
    public byte[] Timestamp { get; set; }
}
```

Właściwości w tle

28.08.2018 • 3 minutes to read • [Edit Online](#)

Właściwości w tle są właściwości, które nie są zdefiniowane w klasie jednostki .NET, ale są zdefiniowane dla tego typu jednostki w modelu platformy EF Core. Wartość i stan tych właściwości jest obsługiwane wyłącznie w śledzeniu zmian.

Właściwości w tle są przydatne w przypadku danych w bazie danych, które nie powinny zostać ujawnione w typach zamapowanego jednostki. W większości przypadków są one używane dla właściwości klucza obcego, których relację między dwiema jednostkami jest reprezentowany przez wartość klucza obcego w bazie danych, lecz relację odbywa się na typy jednostek przy użyciu właściwości nawigacji między typami encji.

Wartości właściwości w tle, które mogą być uzyskane i zmienić za pomocą `ChangeTracker` interfejsu API.

```
context.Entry(myBlog).Property("LastUpdated").CurrentValue = DateTime.Now;
```

Właściwości w tle mogą być przywoływane w zapytaniach LINQ za pośrednictwem `EF.Property` metody statycznej.

```
var blogs = context.Blogs
    .OrderBy(b => EF.Property<DateTime>(b, "LastUpdated"));
```

Konwencje

Właściwości w tle mogą być tworzone przez Konwencję, gdy relacji został odnaleziony, ale nie właściwość klucza obcego znajduje się w klasie jednostki zależne. W takim przypadku zostaną wprowadzone właściwości klucza obcego w tle. Właściwość klucza obcego w tle będą miały nazwę nadaną

`<navigation property name><principal key property name>` (nawigacji na jednostki zależne wskazuje główną jednostki, używany na potrzeby nazywania). Jeśli nazwa właściwości klucza podmiotu zabezpieczeń zawiera nazwę właściwości nawigacji, a następnie po prostu nazwą będzie `<principal key property name>`. Jeśli nie ma właściwości nawigacji jednostki zależne, Nazwa Typ podmiotu zabezpieczeń jest używana w tym miejscu.

Na przykład w poniższym fragmencie kodu spowoduje `BlogId` właściwości w tle są wprowadzane do `Post` jednostki.

```

class MyContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }
    public DbSet<Post> Posts { get; set; }
}

public class Blog
{
    public int BlogId { get; set; }
    public string Url { get; set; }

    public List<Post> Posts { get; set; }
}

public class Post
{
    public int PostId { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }

    public Blog Blog { get; set; }
}

```

Adnotacje danych

Nie można utworzyć właściwości w tle przy użyciu adnotacji danych.

Interfejs Fluent API

Interfejs Fluent API umożliwiają skonfigurowanie właściwości w tle. Gdy wywołujesz przeciążenie ciągu `Property` można połączyć w łańcuchach dowolne wywołania konfiguracji, jak w przypadku innych właściwości.

Jeśli nazwa dostarczona do `Property` metody jest zgodna z nazwą istniejącej właściwości (właściwość w tle lub jeden zdefiniowany w klasie jednostek), a następnie kod służy do konfigurowania właściwości istniejących, zamiast wprowadzać nowe właściwości w tle.

```

class MyContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Blog>()
            .Property<DateTime>("LastUpdated");
    }
}

public class Blog
{
    public int BlogId { get; set; }
    public string Url { get; set; }
}

```

Relacje

28.08.2018 • 21 minutes to read • [Edit Online](#)

Relacja definiuje sposób dwie jednostki powiązane są ze sobą. W relacyjnej bazie danych jest reprezentowane przez ograniczenie klucza obcego.

NOTE

Większość przykładów w tym artykule umożliwiają zademonstrowania pojęć relacji jeden do wielu. Zobacz przykłady relacji jeden do jednego i wiele do wielu [inne wzorce relacji](#) sekcji na końcu tego artykułu.

Definicje terminów

Istnieje wiele terminów używane do opisywania relacji

- **Jednostki zależne:** jest to obiekt, który zawiera właściwości kluczy obcych. Czasami określane jako "podrzędne" w relacji.
- **Jednostka główna:** to jednostka, która zawiera właściwości klucza podstawowego/alternatywnego. Czasami określane jako "parent" w relacji.
- **Klucz obcy:** właściwości w jednostce zależne, który jest używany do przechowywania wartości właściwości klucza jednostki powiązanej jednostki.
- **Klucz jednostki:** właściwości, który unikatowo identyfikuje główną jednostkę. Może to być kluczem podstawowym lub unikatowym.
- **Właściwość nawigacji:** właściwości zdefiniowane w jednostce podmiotu zabezpieczeń i/lub zależne, który zawiera odwołania do powiązanych entity(s).
 - **Właściwości nawigacji kolekcji:** właściwość nawigacji, który zawiera odwołania do wielu powiązanych jednostek.
 - **Odwoływać się do właściwości nawigacji:** właściwość nawigacji, która zawiera odwołanie do pojedynczego obiektu pokrewnego.
 - **Właściwość nawigacji odwrotność:** Omawiając właściwości określonej nawigacji, określenie to odnosi się do właściwości nawigacji na końcu relacji.

W poniższym fragmencie kodu przedstawiono relację jeden do wielu między `Blog` i `Post`

- `Post` to jednostka zależna
- `Blog` to jednostka główna
- `Post.BlogId` jest to klucz obcy
- `Blog.BlogId` to klucz jednostki (w tym przypadku jest kluczem podstawowym, a nie klucza alternatywnego)
- `Post.Blog` jest to właściwość nawigacji odwołania
- `Blog.Posts` jest to właściwość nawigacji kolekcji
- `Post.Blog` jest właściwość nawigacji odwrotność `Blog.Posts` (i na odwrót)

```

public class Blog
{
    public int BlogId { get; set; }
    public string Url { get; set; }

    public List<Post> Posts { get; set; }
}

public class Post
{
    public int PostId { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }

    public int BlogId { get; set; }
    public Blog Blog { get; set; }
}

```

Konwencje

Zgodnie z Konwencją będzie można utworzyć relacji, po wykryte na typ właściwości nawigacji. Właściwość jest traktowana jako właściwość nawigacji, jeśli nie można zamapować typu, który wskazuje jako typ skalarny przez bieżącego dostawcę bazy danych.

NOTE

Relacje, które zostały wykryte przez Konwencję zawsze będą ukierunkowane na klucz podstawowy jednostki głównej. Pod kątem klucza alternatywnego, należy wykonać dodatkowe czynności konfiguracyjne przy użyciu interfejsu API Fluent.

W pełni zdefiniowanych relacji

Najczęstszym wzorcem dla relacji jest zdefiniowane na obu końcach relacji i właściwości klucza obcego zdefiniowanej w klasie jednostki zależne właściwości nawigacji.

- Jeśli para właściwości nawigacji znajduje się między dwoma typami, następnie one zostaną skonfigurowane jako właściwości nawigacji odwrotność tej samej relacji.
- Jeśli jednostka zależna zawiera właściwość o nazwie <primary key property name>, <navigation property name><primary key property name>, lub <principal entity name><primary key property name>, a następnie zostaną skonfigurowane jako klucza obcego.

```

public class Blog
{
    public int BlogId { get; set; }
    public string Url { get; set; }

    public List<Post> Posts { get; set; }
}

public class Post
{
    public int PostId { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }

    public int BlogId { get; set; }
    public Blog Blog { get; set; }
}

```

WARNING

Czy wiele właściwości nawigacji zdefiniowane między dwoma typami (czyli więcej niż jedną parę distinct źródłem, które wskazują na siebie nawzajem), następnie relacje nie zostanie utworzony zgodnie z Konwencją i trzeba będzie ręcznie skonfigurować je do identyfikowania sposob, w jaki pary właściwości nawigacji.

Nie właściwości klucza obcego

Mimo że zaleca się mieć właściwość klucza obcego zdefiniowanej w klasie jednostki zależne, nie jest wymagana. Jeśli zostanie znalezione nie właściwość klucza obcego, zostaną wprowadzone właściwości klucza obcego w tle o nazwie `<navigation property name><principal key property name>` (zobacz [właściwości w tle](#) Aby uzyskać więcej informacji).

```
public class Blog
{
    public int BlogId { get; set; }
    public string Url { get; set; }

    public List<Post> Posts { get; set; }
}

public class Post
{
    public int PostId { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }

    public Blog Blog { get; set; }
}
```

Właściwości pojedynczej wartości

Łącznie z tylko jedną właściwość nawigacji (nie odwrotność nawigacji i nie właściwości klucza obcego) jest wystarczający, aby mieć relacji zdefiniowanych przez Konwencję. Można również mieć właściwości pojedynczej wartości i właściwości klucza obcego.

```
public class Blog
{
    public int BlogId { get; set; }
    public string Url { get; set; }

    public List<Post> Posts { get; set; }
}

public class Post
{
    public int PostId { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }
}
```

Usuwanie kaskadowe

Zgodnie z Konwencją, usuwanie kaskadowe zostanie ustawiony *Cascade* dla relacji wymagane i *ClientSetNull* dla relacji opcjonalne. *Kaskadowe* oznacza, że jednostki zależne również zostaną usunięte. *ClientSetNull* oznacza, że jednostki zależne, które nie są ładowane do pamięci pozostaną bez zmian i muszą być ręcznie usunięty lub poinformowani o prawidłową jednostkę główną. W przypadku jednostek, które są ładowane do pamięci programu EF Core będzie podejmować próby równa właściwości klucza obcego o wartości null.

Zobacz [relacje wymaganych i opcjonalnych](#) sekcji różnicę między relacjami wymagającymi i opcjonalnymi.

Zobacz [usuwanie kaskadowe](#) więcej szczegółów na temat różnych Usuń zachowania i wartości domyślne używane przez Konwencję.

Anotacje danych

Istnieją dwa anotacji danych, które mogą być używane do konfigurowania relacji `[ForeignKey]` i `[InverseProperty]`.

[Klucza obcego]

Korzystanie z anotacji danych, aby skonfigurować, których właściwość powinna być używana jako właściwość klucza obcego dla danej relacji. Zazwyczaj jest to wykonywane, gdy właściwość klucza obcego nie został odnaleziony przez Konwencję.

```
public class Blog
{
    public int BlogId { get; set; }
    public string Url { get; set; }

    public List<Post> Posts { get; set; }
}

public class Post
{
    public int PostId { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }

    public int BlogForeignKey { get; set; }

    [ForeignKey("BlogForeignKey")]
    public Blog Blog { get; set; }
}
```

TIP

`[ForeignKey]` Anotacji można umieścić we właściwości nawigacji, albo w relacji. Nie musi przejść na właściwość nawigacji w klasie jednostki zależne.

[InverseProperty]

Korzystanie z anotacji danych, aby skonfigurować sposób pary właściwości nawigacji jednostki zależnej i głównej. Zazwyczaj jest to wykonywane, gdy istnieje więcej niż jedną parę właściwości nawigacji między dwoma typami encji.

```
public class Post
{
    public int PostId { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }

    public int AuthorUserId { get; set; }
    public User Author { get; set; }

    public int ContributorUserId { get; set; }
    public User Contributor { get; set; }
}

public class User
{
    public string UserId { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }

    [InverseProperty("Author")]
    public List<Post> AuthoredPosts { get; set; }

    [InverseProperty("Contributor")]
    public List<Post> ContributedToPosts { get; set; }
}
```

Interfejs Fluent API

Aby skonfigurować relację w interfejsie API Fluent, możesz rozpocząć, określając właściwości nawigacji, które tworzą relacji. `HasOne` lub `HasMany` identyfikuje właściwość nawigacji typu jednostki, rozpoczynamy od konfiguracji na. Można następnie połączyć w łańcuch po wywołaniu `WithOne` lub `WithMany` do identyfikowania odwrotność nawigacji. `HasOne` / `WithOne` są używane dla właściwości nawigacji odwołania i `HasMany` / `WithMany` są używane dla właściwości nawigacji kolekcji.

```

class MyContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }
    public DbSet<Post> Posts { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Post>()
            .HasOne(p => p.Blog)
            .WithMany(b => b.Posts);
    }
}

public class Blog
{
    public int BlogId { get; set; }
    public string Url { get; set; }

    public List<Post> Posts { get; set; }
}

public class Post
{
    public int PostId { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }

    public Blog Blog { get; set; }
}

```

Właściwości pojedynczej wartości

Jeśli masz tylko jedną właściwość nawigacji, a następnie istnieją przeciążenia bez parametrów `WithOne` i `WithMany`. Oznacza to, że ma pod względem koncepcyjnym odwołanie lub kolekcji na końcu relacji, ale nie ma właściwości nawigacji zawarta w klasie jednostki.

```

class MyContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }
    public DbSet<Post> Posts { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Blog>()
            .HasMany(b => b.Posts)
            .WithOne();
    }
}

public class Blog
{
    public int BlogId { get; set; }
    public string Url { get; set; }

    public List<Post> Posts { get; set; }
}

public class Post
{
    public int PostId { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }
}

```

Klucz obcy

Interfejs Fluent API umożliwia skonfigurowanie, których właściwość powinna być używana jako właściwość klucza obcego dla danej relacji.

```
class MyContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }
    public DbSet<Post> Posts { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Post>()
            .HasOne(p => p.Blog)
            .WithMany(b => b.Posts)
            .HasForeignKey(p => p.BlogForeignKey);
    }
}

public class Blog
{
    public int BlogId { get; set; }
    public string Url { get; set; }

    public List<Post> Posts { get; set; }
}

public class Post
{
    public int PostId { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }

    public int BlogForeignKey { get; set; }
    public Blog Blog { get; set; }
}
```

W poniższym fragmencie kodu przedstawiono sposób konfigurowania złożonego klucza obcego.

```

class MyContext : DbContext
{
    public DbSet<Car> Cars { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Car>()
            .HasKey(c => new { c.State, c.LicensePlate });

        modelBuilder.Entity<RecordOfSale>()
            .HasOne(s => s.Car)
            .WithMany(c => c.SaleHistory)
            .HasForeignKey(s => new { s.CarState, s.CarLicensePlate });
    }
}

public class Car
{
    public string State { get; set; }
    public string LicensePlate { get; set; }
    public string Make { get; set; }
    public string Model { get; set; }

    public List<RecordOfSale> SaleHistory { get; set; }
}

public class RecordOfSale
{
    public int RecordOfSaleId { get; set; }
    public DateTime DateSold { get; set; }
    public decimal Price { get; set; }

    public string CarState { get; set; }
    public string CarLicensePlate { get; set; }
    public Car Car { get; set; }
}

```

Można użyć ciągu przeciążenia `HasForeignKey(...)` do skonfigurowania właściwości w tle jako klucz obcy (zobacz [właściwości w tle](#) Aby uzyskać więcej informacji). Zalecane jest jawne dodanie właściwości w tle do modelu, zanim użyjesz go jako klucza obcego (jak pokazano poniżej).

```

class MyContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }
    public DbSet<Post> Posts { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        // Add the shadow property to the model
        modelBuilder.Entity<Post>()
            .Property<int>("BlogForeignKey");

        // Use the shadow property as a foreign key
        modelBuilder.Entity<Post>()
            .HasOne(p => p.Blog)
            .WithMany(b => b.Posts)
            .HasForeignKey("BlogForeignKey");
    }
}

public class Blog
{
    public int BlogId { get; set; }
    public string Url { get; set; }

    public List<Post> Posts { get; set; }
}

public class Post
{
    public int PostId { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }

    public Blog Blog { get; set; }
}

```

Klucz jednostki

Jeśli chcesz, aby klucz obcy, aby odwoływać się do właściwości innego niż klucz podstawowy, można użyć interfejsu API Fluent do skonfigurowania właściwości klucza podmiotu zabezpieczeń dla relacji. Właściwość, którą można skonfigurować jako głównej będą kluczowe automatycznie należy skonfigurować jako klucza alternatywnego (zobacz [klucze alternatywne](#) Aby uzyskać więcej informacji).

```
class MyContext : DbContext
{
    public DbSet<Car> Cars { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<RecordOfSale>()
            .HasOne(s => s.Car)
            .WithMany(c => c.SaleHistory)
            .HasForeignKey(s => s.CarLicensePlate)
            .HasPrincipalKey(c => c.LicensePlate);
    }
}

public class Car
{
    public int CarId { get; set; }
    public string LicensePlate { get; set; }
    public string Make { get; set; }
    public string Model { get; set; }

    public List<RecordOfSale> SaleHistory { get; set; }
}

public class RecordOfSale
{
    public int RecordOfSaleId { get; set; }
    public DateTime DateSold { get; set; }
    public decimal Price { get; set; }

    public string CarLicensePlate { get; set; }
    public Car Car { get; set; }
}
```

W poniższym fragmencie kodu przedstawiono sposób konfigurowania złożony klucz jednostki.

```

class MyContext : DbContext
{
    public DbSet<Car> Cars { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<RecordOfSale>()
            .HasOne(s => s.Car)
            .WithMany(c => c.SaleHistory)
            .HasForeignKey(s => new { s.CarState, s.CarLicensePlate })
            .HasPrincipalKey(c => new { c.State, c.LicensePlate });
    }
}

public class Car
{
    public int CarId { get; set; }
    public string State { get; set; }
    public string LicensePlate { get; set; }
    public string Make { get; set; }
    public string Model { get; set; }

    public List<RecordOfSale> SaleHistory { get; set; }
}

public class RecordOfSale
{
    public int RecordOfSaleId { get; set; }
    public DateTime DateSold { get; set; }
    public decimal Price { get; set; }

    public string CarState { get; set; }
    public string CarLicensePlate { get; set; }
    public Car Car { get; set; }
}

```

WARNING

Kolejność, w której zostaną określone jednostki właściwości klucza musi być zgodna kolejność, w którym są określone w kluczu obcym.

Relacje wymagane i opcjonalne

Aby określić, czy relacja jest wymagane lub opcjonalne, można użyć interfejsu API Fluent. Ostatecznie to określa, czy właściwość klucza obcego jest wymagane lub opcjonalne. Jest to najbardziej przydatne, korzystając z kluczem obcym stanu w tle. Jeśli masz właściwości klucza obcego w klasie jednostki, a następnie requiredness relacji jest określana na podstawie tego, czy właściwość klucza obcego jest wymagane lub opcjonalne (zobacz [wymagane i opcjonalne właściwości](#) Aby uzyskać więcej informacji informacje o).

```
class MyContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }
    public DbSet<Post> Posts { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Post>()
            .HasOne(p => p.Blog)
            .WithMany(b => b.Posts)
            .IsRequired();
    }
}

public class Blog
{
    public int BlogId { get; set; }
    public string Url { get; set; }

    public List<Post> Posts { get; set; }
}

public class Post
{
    public int PostId { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }

    public Blog Blog { get; set; }
}
```

Usuwanie kaskadowe

Interfejs Fluent API umożliwiają skonfigurowanie jawnie zachowanie usuwanie kaskadowe określonej relacji.

Zobacz [usuwanie kaskadowe](#) sekcji Zapisywanie danych szczegółowe omówienie każdej z nich.

```

class MyContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }
    public DbSet<Post> Posts { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Post>()
            .HasOne(p => p.Blog)
            .WithMany(b => b.Posts)
            .OnDelete(DeleteBehavior.Cascade);
    }
}

public class Blog
{
    public int BlogId { get; set; }
    public string Url { get; set; }

    public List<Post> Posts { get; set; }
}

public class Post
{
    public int PostId { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }

    public int? BlogId { get; set; }
    public Blog Blog { get; set; }
}

```

Inne wzorce relacji

Jeden do jednego

Relacje jeden-do-jednego ma odwołanie do właściwości nawigacji po obu stronach. Używają tej samej Konwencji co relacji jeden do wielu, ale unikatowy indeks został wprowadzony na właściwość klucza obcego, aby upewnić się, że tylko jeden zależny od powiązany jest każdy podmiot zabezpieczeń.

```

public class Blog
{
    public int BlogId { get; set; }
    public string Url { get; set; }

    public BlogImage BlogImage { get; set; }
}

public class BlogImage
{
    public int BlogImageId { get; set; }
    public byte[] Image { get; set; }
    public string Caption { get; set; }

    public int BlogId { get; set; }
    public Blog Blog { get; set; }
}

```

NOTE

EF wybrac jeden z jednostki, które jest zależne od oparte na możliwość wykrywania właściwości klucza obcego. Jeśli problem jednostki jest wybierany jako zależnych od ustawień lokalnych, można użyć Fluent API, aby rozwiązać ten problem.

Podczas konfigurowania relacji z interfejsem API Fluent, możesz użyć `HasOne` i `WithOne` metody.

Podczas konfigurowania klucza obcego, należy określić typ jednostki zależne - Zwróć uwagę, parametr generyczny udostępniane `HasForeignKey` na liście poniżej. W przypadku relacji jeden do wielu jest jasne, czy jednostka z nawigacją odwołanie jest zależnych od ustawień lokalnych, jak i z kolekcji jest podmiot zabezpieczeń. Ale to nie jest tak w przypadku relacji jeden do jednego — dlatego trzeba jawnie zdefiniować go.

```
class MyContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }
    public DbSet<BlogImage> BlogImages { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Blog>()
            .HasOne(p => p.BlogImage)
            .WithOne(i => i.Blog)
            .HasForeignKey<BlogImage>(b => b.BlogForeignKey);
    }
}

public class Blog
{
    public int BlogId { get; set; }
    public string Url { get; set; }

    public BlogImage BlogImage { get; set; }
}

public class BlogImage
{
    public int BlogImageId { get; set; }
    public byte[] Image { get; set; }
    public string Caption { get; set; }

    public int BlogForeignKey { get; set; }
    public Blog Blog { get; set; }
}
```

Wiele do wielu

Relacje wiele do wielu, bez klasę jednostki, do reprezentowania tabelę sprzężenia nie są jeszcze obsługiwane. Jednak może reprezentować relacji wiele do wielu, umieszczając klasę jednostki dla tabeli sprzężenia i dwie oddzielne relacje jeden do wielu mapowania.

```
class MyContext : DbContext
{
    public DbSet<Post> Posts { get; set; }
    public DbSet<Tag> Tags { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<PostTag>()
            .HasKey(t => new { t.PostId, t.TagId });

        modelBuilder.Entity<PostTag>()
            .HasOne(pt => pt.Post)
            .WithMany(p => p.PostTags)
            .HasForeignKey(pt => pt.PostId);

        modelBuilder.Entity<PostTag>()
            .HasOne(pt => pt.Tag)
            .WithMany(t => t.PostTags)
            .HasForeignKey(pt => pt.TagId);
    }
}

public class Post
{
    public int PostId { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }

    public List<PostTag> PostTags { get; set; }
}

public class Tag
{
    public string TagId { get; set; }

    public List<PostTag> PostTags { get; set; }
}

public class PostTag
{
    public int PostId { get; set; }
    public Post Post { get; set; }

    public string TagId { get; set; }
    public Tag Tag { get; set; }
}
```

Indeksy

28.08.2018 • 2 minutes to read • [Edit Online](#)

Indeksy są typowe pojęcia w wielu magazynach danych. Podczas ich wdrażania w magazynie danych mogą się różnić, są one używane do lepiej wyszukiwań na podstawie kolumny (lub zestaw kolumn) wydajne.

Konwencje

Zgodnie z Konwencją indeksu jest tworzony w każdej właściwości (lub zestaw właściwości), używany jako klucz obcego.

Adnotacje danych

Nie można utworzyć indeksów przy użyciu adnotacji danych.

Interfejs Fluent API

Interfejs Fluent API służy do określenia indeksu w jednej właściwości. Indeksy są domyślnie nie jest unikatowa.

```
class MyContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Blog>()
            .HasIndex(b => b.Url);
    }
}

public class Blog
{
    public int BlogId { get; set; }
    public string Url { get; set; }
}
```

Można również określić czy indeksu powinna być unikatowa, co oznacza, że nie dwie jednostki może mieć tej samej wartości dla danej właściwości.

```
modelBuilder.Entity<Blog>()
    .HasIndex(b => b.Url)
    .IsUnique();
```

Można również określić indeks przez więcej niż jedną kolumnę.

```
class MyContext : DbContext
{
    public DbSet<Person> People { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Person>()
            .HasIndex(p => new { p.FirstName, p.LastName });
    }
}

public class Person
{
    public int PersonId { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
}
```

TIP

Istnieje tylko jeden indeks na odrębnym zestawem właściwości. W przypadku konfigurowania indeksu na zbiór właściwości, które ma już zdefiniowane, indeks przy Konwencji lub poprzedniej konfiguracji, za pomocą interfejsu API Fluent następnie spowoduje zmianę definicji indeksu. Jest to przydatne, jeśli chcesz dodatkowo skonfigurować indeks, który został utworzony przez Konwencję.

Klucze alternatywne

28.08.2018 • 3 minutes to read • [Edit Online](#)

Klucz alternatywnego służy jako alternatywne Unikatowy identyfikator dla każdego wystąpienia jednostki, oprócz klucz podstawowy. Klucze alternatywne może służyć jako element docelowy relacji. Przy użyciu relacyjnej bazy danych to mapuje do koncepcji unikatowego indeksu/ograniczenia na alternatywny kolumn kluczy i jeden lub więcej ograniczeń klucza obcego odwołujące się do kolumn na liście.

TIP

Jeśli chcesz wymusić unikatowość kolumny, a następnie chcesz, aby zamiast klucza alternatywnego unikatowego indeksu, zobacz [indeksów](#). W programie EF klucze alternatywne zapewnić większą funkcjonalność niż indeksy unikatowe, ponieważ może służyć jako cel klucza obcego.

Klucze alternatywne są zwykle wprowadzane dla Ciebie w razie i nie trzeba ręcznie skonfigurować je. Zobacz [konwencje](#) Aby uzyskać więcej informacji.

Konwencje

Zgodnie z Konwencją klucza alternatywnego został wprowadzony dla Ciebie podczas określania właściwości, który nie jest kluczem podstawowym jako element docelowy relacji.

```
class MyContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }
    public DbSet<Post> Posts { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Post>()
            .HasOne(p => p.Blog)
            .WithMany(b => b.Posts)
            .HasForeignKey(p => p.BlogUrl)
            .HasPrincipalKey(b => b.Url);
    }
}

public class Blog
{
    public int BlogId { get; set; }
    public string Url { get; set; }

    public List<Post> Posts { get; set; }
}

public class Post
{
    public int PostId { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }

    public string BlogUrl { get; set; }
    public Blog Blog { get; set; }
}
```

Adnotacje danych

Klucze alternatywne nie można skonfigurować przy użyciu adnotacji danych.

Interfejs Fluent API

Interfejs Fluent API umożliwiają skonfigurowanie jednej właściwości klucza alternatywnego.

```
class MyContext : DbContext
{
    public DbSet<Car> Cars { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Car>()
            .HasAlternateKey(c => c.LicensePlate);
    }
}

class Car
{
    public int CarId { get; set; }
    public string LicensePlate { get; set; }
    public string Make { get; set; }
    public string Model { get; set; }
}
```

Można również skonfigurować wiele właściwości jako klucza alternatywnego (znanych jako alternatywne klucz złożony) za pomocą Fluent interfejsu API.

```
class MyContext : DbContext
{
    public DbSet<Car> Cars { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Car>()
            .HasAlternateKey(c => new { c.State, c.LicensePlate });
    }
}

class Car
{
    public int CarId { get; set; }
    public string State { get; set; }
    public string LicensePlate { get; set; }
    public string Make { get; set; }
    public string Model { get; set; }
}
```

Dziedziczenie

28.08.2018 • 2 minutes to read • [Edit Online](#)

Dziedziczenie w modelu platformy EF jest używane do kontrolowania, jak dziedziczenia klas jednostek jest reprezentowana w bazie danych.

Konwencje

Według Konwencji jest dostawca bazy danych, aby określić, jak dziedziczenie będzie reprezentowany w bazie danych. Zobacz [dziedziczenie \(relacyjna baza danych\)](#) dla jak jest to obsługiwane za pomocą dostawcy relacyjnej bazy danych.

EF skonfiguruje tylko dziedziczenie, jeśli co najmniej dwa dziedziczone typy są jawnie uwzględnione w modelu. EF nie będzie skanować dla typów podstawowych i pochodnych, które w przeciwnym razie nie zostały uwzględnione w modelu. Mogą zawierać typy w modelu, zapewniając *DbSet* dla każdego typu w hierarchii dziedziczenia.

```
class MyContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }
    public DbSet<RssBlog> RssBlogs { get; set; }
}

public class Blog
{
    public int BlogId { get; set; }
    public string Url { get; set; }
}

public class RssBlog : Blog
{
    public string RssUrl { get; set; }
}
```

Jeśli nie chcesz udostępnić *DbSet* dla co najmniej jednej jednostki w hierarchii można użyć Fluent API, aby upewnić się, że są one uwzględnione w modelu. A jeśli użytkownik nie należy polegać na konwencjach można określić typ podstawowy, w sposób jawny przy użyciu `HasBaseType`.

```
class MyContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<RssBlog>().HasBaseType<Blog>();
    }
}
```

NOTE

Mozesz użyć `.HasBaseType((Type)null)` można usunąć typu jednostki z hierarchii.

Adnotacje danych

Nie można użyć adnotacji danych, aby skonfigurować dziedziczenie.

Interfejs Fluent API

Interfejs Fluent API dziedziczenia zależy od dostawcy bazy danych, którego używasz. Zobacz [dziedziczenie \(relacyjna baza danych\)](#) konfiguracji można wykonać dla dostawcy relacyjnej bazy danych.

Pola zapasowe

28.08.2018 • 4 minutes to read • [Edit Online](#)

NOTE

Ta funkcja jest nowa w programie EF Core 1.1.

Pola zapasowego umożliwiają EF do odczytu lub zapisu do pola, a nie właściwości. Może to być przydatne, gdy hermetyzacji klasy jest używany do ograniczenia użytkowania i/lub zwiększa semantykę dostępu do danych przez kod aplikacji, ale wartość powinna być odczytywać i/lub zapisywanych w bazie danych bez korzystania z tych ograniczeń / ulepszenia.

Konwencje

Zgodnie z Konwencją następujące pola zostaną odnalezione jak kopie pól dla danej właściwości (wymienione w kolejność pierwszeństwa). Pola odnajdywane będą tylko dla właściwości, które znajdują się w modelu. Aby uzyskać więcej informacji, na którym właściwości znajdują się w modelu, zobacz [uwzględnianie i wykluczanie właściwości](#).

- `_<camel-cased property name>`
- `_<property name>`
- `m_<camel-cased property name>`
- `m_<property name>`

```
public class Blog
{
    private string _url;

    public int BlogId { get; set; }

    public string Url
    {
        get { return _url; }
        set { _url = value; }
    }
}
```

Po skonfigurowaniu polem zapasowym EF zapisze bezpośrednio do tego pola, gdy materializowanie wystąpień jednostek w bazie danych (zamiast przy użyciu metoda ustawaająca właściwości). Jeśli EF musi odczytać lub zapisać wartości w pozostałych godzinach, właściwość zostanie użyty, jeśli jest to możliwe. Na przykład jeśli EF należy zaktualizować wartość właściwości go użyje metoda ustawaająca właściwości, jeśli jest zdefiniowana. Jeśli właściwość jest tylko do odczytu, następnie go zapisuje do pola.

Adnotacje danych

Pola zapasowe nie można skonfigurować przy użyciu adnotacji danych.

Interfejs Fluent API

Interfejs Fluent API umożliwiają skonfigurowanie z polem zapasowym dla właściwości.

```

class MyContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Blog>()
            .Property(b => b.Url)
            .HasField("_validatedUrl");
    }
}

public class Blog
{
    private string _validatedUrl;

    public int BlogId { get; set; }

    public string Url
    {
        get { return _validatedUrl; }
    }

    public void SetUrl(string url)
    {
        using (var client = new HttpClient())
        {
            var response = client.GetAsync(url).Result;
            response.EnsureSuccessStatusCode();
        }

        _validatedUrl = url;
    }
}

```

Kontrolowanie, gdy pole jest używane

Można skonfigurować, kiedy EF używa pola lub właściwości. Zobacz [wyliczenia PropertyAccessMode](#) obsługiwanych opcjach.

```

modelBuilder.Entity<Blog>()
    .Property(b => b.Url)
    .HasField("_validatedUrl")
    .UsePropertyAccessMode(PropertyAccessMode.Field);

```

Pola bez właściwości

Ogólne właściwości można też utworzyć w modelu nie ma odpowiadającą właściwość CLR w klasie jednostki, ale zamiast tego używa pola do przechowywania danych w jednostce. To różni się od [właściwości w tle](#), której dane są przechowywane w śledzenie zmian. To wykorzystania, jeśli klasa jednostki używa metody do pobierania/ustawiania wartości.

EF można nadać nazwę pola w `Property(...)` interfejsu API. Jeśli nie ma właściwości o podanej nazwie, EF będzie wyglądać dla pola.

```

class MyContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Blog>()
            .Property(" _validatedUrl");
    }
}

public class Blog
{
    private string _validatedUrl;

    public int BlogId { get; set; }

    public string GetUrl()
    {
        return _validatedUrl;
    }

    public void SetUrl(string url)
    {
        using (var client = new HttpClient())
        {
            var response = client.GetAsync(url).Result;
            response.EnsureSuccessStatusCode();
        }

        _validatedUrl = url;
    }
}

```

Możesz również podać właściwość nazwę, inną niż nazwa pola. Ta nazwa jest następnie używana podczas tworzenia modelu, głównie będzie możliwa używać dla nazwy kolumny, który jest mapowany w bazie danych.

```

protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Blog>()
        .Property<string>("Url")
        .HasField("_validatedUrl");
}

```

Jeśli nie ma właściwości w klasie jednostki, możesz użyć `EF.Property(...)` metody w zapytaniu programu LINQ do odwoływanego się do właściwości, pod względem koncepcyjnym będącej częścią modelu.

```
var blogs = db.blogs.OrderBy(b => EF.Property<string>(b, "Url"));
```

Konwersje wartości

29.08.2018 • 7 minutes to read • [Edit Online](#)

NOTE

Ta funkcja jest nowa na platformie EF Core 2.1.

Konwertery wartości Zezwalaj na wartości właściwości, które ma zostać przekonwertowany, podczas odczytu / zapisu do bazy danych. Ta konwersja może być z jedną wartością na inny tego samego typu (na przykład ciągi, szyfrowanie) lub wartością jednego typu z wartością innego typu (na przykład konwertowania wartości wyliczenia do i z ciągów w bazie danych.)

Podstawy

Konwertery wartości są określane na podstawie `ModelClrType` i `ProviderClrType`. Typ modelu jest typem architektury .NET właściwości typu jednostki. Typ dostawcy jest typem architektury .NET zrozumiałą dla dostawcy bazy danych. Na przykład, aby zapisać wyliczenia jako ciągi w bazie danych, typ modelu jest typem wyliczenia, a typ dostawcy jest `String`. Te dwa typy mogą być takie same.

Konwersje są definiowane za pomocą dwóch `Func` drzew wyrażeń: jeden z `ModelClrType` do `ProviderClrType`, a druga z `ProviderClrType` do `ModelClrType`. Drzewa wyrażeń są używane i mogą być kompilowane do kodu dostępu do bazy danych podczas konwersji wydajne. W przypadku złożonych konwersji drzewa wyrażeń mogą być proste wywołanie metody, która wykonuje konwersję.

Konfigurowanie konwertera wartości

Konwersje wartości są zdefiniowane na właściwości w `OnModelCreating` Twojego `DbContext`. Na przykład należy wziąć pod uwagę typem wyliczenia i jednostkę zdefiniowany jako:

```
public class Rider
{
    public int Id { get; set; }
    public EquineBeast Mount { get; set; }
}

public enum EquineBeast
{
    Donkey,
    Mule,
    Horse,
    Unicorn
}
```

Następnie konwersje można zdefiniować w `OnModelCreating` do przechowywania wartości wyliczenia jako ciągi (na przykład "Ośłów", "Muł", ...) w bazie danych:

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder
        .Entity<Rider>()
        .Property(e => e.Mount)
        .HasConversion(
            v => v.ToString(),
            v => (EquineBeast)Enum.Parse(typeof(EquineBeast), v));
}
```

NOTE

A `null` wartość nigdy nie zostaną przekazane do konwertera wartości. To ułatwia wykonanie konwersji i umożliwia im być współużytkowane przez właściwości dopuszczają wartości null i nie dopuszcza wartości null.

Klasa elementu ValueConverter

Wywoływanie `HasConversion` jak wspomniano powyżej, spowoduje to utworzenie `ValueConverter` wystąpienia i ustaw jego właściwości. `ValueConverter` Zamiast tego mogą być tworzone w sposób jawnym. Na przykład:

```
var converter = new ValueConverter<EquineBeast, string>(
    v => v.ToString(),
    v => (EquineBeast)Enum.Parse(typeof(EquineBeast), v));

modelBuilder
    .Entity<Rider>()
    .Property(e => e.Mount)
    .HasConversion(converter);
```

Może to być przydatne, gdy wiele właściwości, użyj tej samej konwersji.

NOTE

Obecnie nie ma możliwości można określić w jednym miejscu, że dla każdej właściwości danego typu musi używać tego samego konwertera wartości. Ta funkcja zostanie uwzględniona w przyszłej wersji.

Wbudowane konwertery

EF Core dostarczany z zestawem wstępnie zdefiniowane `ValueConverter` znalezione w klasie `Microsoft.EntityFrameworkCore.Storage.ValueConversion` przestrzeni nazw. Są to:

- `BoolToZeroOneConverter` — Bool na zero i jeden
- `BoolToStringConverter` — Bool do ciągów, takich jak "Y" i "N"
- `BoolToTwoValuesConverter` — Bool do dowolnych dwóch wartości.
- `BytesToStringConverter` -Tablica bajtów na串 kodowany w formacie Base64
- `CastingConverter` -Konwersje, które wymagają rzutowanie typu
- `CharToStringConverter` -Char串 pojedynczy znak
- `DateTimeOffsetToBinaryConverter` — DateTimeOffset na wartość 64-bitowych zakodowane w formacie pliku binarnego
- `DateTimeOffsetToBytesConverter` — DateTimeOffset do tablicy typu byte
- `DateTimeOffsetToStringConverter` — DateTimeOffset串
- `DateTimeToBinaryConverter` — Data i godzina wartości 64-bitowe, w tym DateTimeKind

- `DateTimeToStringConverter` — Data i godzina ciąg
- `DateTimeToTicksConverter` -Data i godzina impulsów
- `EnumToNumberConverter` -Wyliczenie numer podstawowy
- `EnumToStringConverter` — Enum ciąg
- `GuidToBytesConverter` — Identyfikator Guid do tablicy typu byte
- `GuidToStringConverter` — Identyfikator Guid ciąg
- `NumberToBytesConverter` -Dowolna wartość liczbową do tablicy typu byte
- `NumberToStringConverter` -Dowolna wartość liczbową, ciąg
- `StringToBytesConverter` -Ciągu UTF8 bajtów
- `TimeSpanToStringConverter` — TimeSpan ciąg
- `TimeSpanToTicksConverter` -TimeSpan impulsów

Należy zauważyć, że `EnumToStringConverter` znajduje się na tej liście. Oznacza to, że nie ma potrzeby określić konwersji jawnie, jak pokazano powyżej. Zamiast tego można użyć konwertera wbudowane:

```
var converter = new EnumToStringConverter<EquineBeast>();

modelBuilder
    .Entity<Rider>()
    .Property(e => e.Mount)
    .HasConversion(converter);
```

Należy pamiętać, że wbudowane konwertery są bezstanowe, a więc pojedyncze wystąpienie może być bezpiecznie współużytkowane przez wiele właściwości.

Wstępnie zdefiniowane konwersje

W przypadku typowych konwersji dla których istnieje wbudowany konwerter nie ma potrzeby jawnie określić konwertera. Zamiast tego po prostu skonfigurować należy używać typu dostawcy i platforma EF automatycznie użyje odpowiedni konwerter wbudowanych. Wyliczenie w celu konwersji ciągów są używane jako przykład powyżej, ale EF faktycznie wykona to automatycznie, jeśli skonfigurowano typ dostawcy:

```
modelBuilder
    .Entity<Rider>()
    .Property(e => e.Mount)
    .HasConversion<string>();
```

Ten sam efekt można osiągnąć przez jawnie określenie typu kolumny. Na przykład, jeśli nie zdefiniowano typu jednostki, takie jak tak:

```
public class Rider
{
    public int Id { get; set; }

    [Column(TypeName = "nvarchar(24)")]
    public EquineBeast Mount { get; set; }
}
```

Następnie wartości wyliczenia, które zostaną zapisane jako ciągi w bazie danych bez dalszej konfiguracji w `OnModelCreating`.

Ograniczenia

Istnieje kilka znanych bieżących ograniczeń systemu konwersji wartości:

- Jak wspomniano powyżej, `null` nie może zostać przekonwertowany.
- Obecnie nie ma możliwości się konwersję z jedną właściwość wiele kolumn lub na odwrót.
- Korzystanie z konwersji wartości mogą mieć wpływ na możliwość translacji wyrażeń do bazy danych SQL platformy EF Core. W takich przypadkach zostanie zarejestrowane ostrzeżenie. Usunięcie tych ograniczeń jest rozważane w przyszłej wersji.

Wstępne wypełnianie danych

16.01.2019 • 6 minutes to read • [Edit Online](#)

Wstępne wypełnianie danych polega na wypełnianie bazy danych za pomocą początkowego zestawu danych.

Istnieje kilka sposobów, które można to zrobić w programie EF Core:

- Modelowanie danych inicjatora
- Dostosowywanie ręcznej migracji
- Logikę niestandardową inicjalizację

Modelowanie danych inicjatora

NOTE

Ta funkcja jest nowa na platformie EF Core 2.1.

W odróżnieniu od w EF6 w programie EF Core wstępne wypełnianie danych może być skojarzony z typem jednostki jako część konfiguracji modelu. Następnie programu EF Core [migracje](#) można automatycznie obliczyć co Wstawianie, aktualizowanie lub usuwanie potrzebę operacji mają być stosowane podczas uaktualniania bazy danych do nowej wersji modelu.

NOTE

Podczas określania, jakie operacja powinna być wykonana pobierać dane inicjatora do żądanego stanu, migracje analizuje tylko zmiany modelu. Dlatego wszelkie zmiany w danych poza migracje mogą zostać utracone lub nie powodują wystąpienie błędu.

Na przykład dane zostaną skonfigurowane `Blog` w `OnModelCreating`:

```
modelBuilder.Entity<Blog>().HasData(new Blog { BlogId = 1, Url = "http://sample.com" });
```

Aby dodać jednostki, które mają relację wartości klucza obcego muszą być określone:

```
modelBuilder.Entity<Post>().HasData(  
    new Post() { BlogId = 1, PostId = 1, Title = "First post", Content = "Test 1" });
```

Jeśli typ jednostki ma wszystkie właściwości w stanie w tle klasa anonimowa może służyć do Podaj wartości:

```
modelBuilder.Entity<Post>().HasData(  
    new { BlogId = 1, PostId = 2, Title = "Second post", Content = "Test 2" });
```

Należące do jednostki, który może zostać rozpoczęta typów w podobny sposób:

```
modelBuilder.Entity<Post>().OwnsOne(p => p.AuthorName).HasData(  
    new { PostId = 1, First = "Andriy", Last = "Svyryd" },  
    new { PostId = 2, First = "Diego", Last = "Vega" });
```

Zobacz pełny przykład projektu Aby uzyskać dodatkowy kontekst.

Po dodaniu danych do modelu, [migracje](#) powinien być używany do zastosowania zmian.

TIP

Jeśli konieczne jest zastosowanie migracji w ramach zautomatyzowanego wdrażania możesz [Utwórz skrypt SQL](#) można je przeglądać przed wykonaniem.

Alternatywnie, można użyć `context.Database.EnsureCreated()` do utworzenia nowej bazy danych zawierającej dane inicjatora, na przykład w przypadku bazy danych testów lub za pomocą dostawcy w pamięci lub dowolnej-relation bazy danych. Należy pamiętać, że jeśli baza danych już istnieje, `EnsureCreated()` zaktualizuje żadnego schematu ani inicjatora w bazie danych. Relacyjne bazy danych nie należy wywoływać `EnsureCreated()`. Jeśli planujesz użyć migracje.

Ograniczenia danych inicjatora modelu

Inicjatora danych tego typu jest zarządzana przez migracje i skrypt do aktualizowania danych, który jest już w bazie danych musi zostać wygenerowane bez połączenia z bazą danych. To nakłada pewne ograniczenia:

- Wartość klucza podstawowego nie trzeba określać, nawet jeśli zwykle jest generowany przez bazę danych. Będzie służyć do wykrywania zmian danych między migracjami.
- Uprzednio wprowadzonych danych zostanie usunięty, zmiana klucza podstawowego w dowolny sposób.

W związku z tym ta funkcja jest najbardziej przydatny w przypadku danych statycznych, który nie ma powinna się zmieniać poza migracje i nie zależy od niczego więcej w bazie danych, na przykład kody pocztowe.

Jeśli scenariusz zawiera następujące zalecane jest używana logika inicjowania niestandardowego opisanego w ostatniej sekcji:

- Dane tymczasowe na potrzeby testowania
- Dane, które jest zależny od stanu bazy danych
- Dane, które wymaga wartości klucza, zostanie wygenerowany przez bazę danych, w tym jednostki używające klucze alternatywne jako tożsamość
- Dane, które wymaga niestandardowej transformacji (nie jest obsługiwany przez [wartość konwersje](#)), takie jak niektóre tworzenia skrótów haseł
- Dane, które wymaga wywołania funkcji API zewnętrznych, takich jak tworzenie ról i użytkowników tożsamości platformy ASP.NET Core

Dostosowywanie ręcznej migracji

Po dodaniu zmiany w danych określony za pomocą migracji `HasData` są przekształcane do wywołania `InsertData()`, `UpdateData()`, i `DeleteData()`. Jednym ze sposobów obejścia niektórych ograniczeń `HasData` jest, aby ręcznie dodać te wywołania lub [operacje niestandardowe](#) migracji zamiast tego.

```
migrationBuilder.InsertData(  
    table: "Blogs",  
    columns: new[] { "Url" },  
    values: new object[] { "http://generated.com" });
```

Logikę niestandardową inicjalizacji

Prosty, zaawansowany sposób wykonania wstępne wypełnianie danych jest użycie `DbContext.SaveChanges()` przed głównej aplikacji logiki rozpoczęcia wykonywanie.

```
using (var context = new DataSeedingContext())
{
    context.Database.EnsureCreated();

    var testBlog = context.Blogs.FirstOrDefault(b => b.Url == "http://test.com");
    if (testBlog == null)
    {
        context.Blogs.Add(new Blog { Url = "http://test.com" });
    }
    context.SaveChanges();
}
```

WARNING

Rozmieszczania kod nie powinien być częścią wykonywania zwykłej aplikacji, ponieważ może to spowodować problemy ze współbieżnością, gdy wiele wystąpień działają, a także wymagałoby aplikacji mających uprawnienia do modyfikowania schematu bazy danych.

W zależności od ograniczeń wdrożenia kod inicjujący mogą być wykonywane na różne sposoby:

- Uruchamianie aplikacji inicjowania lokalnie
- Wdrażanie aplikacji inicjowanie przy użyciu głównej aplikacji wywoływanie procedury inicjowania i wyłączenie lub usunięcie aplikacji inicjowania.

Zazwyczaj można to zautomatyzować za pomocą [profilów publikowania](#).

Typy jednostek za pomocą konstruktorów

29.08.2018 • 10 minutes to read • [Edit Online](#)

NOTE

Ta funkcja jest nowa na platformie EF Core 2.1.

Począwszy od programu EF Core 2.1 jest teraz możliwa zdefiniowanie konstruktora z parametrami, a programem EF Core wywołania tego konstruktora, podczas tworzenia wystąpienia jednostki. Parametry Konstruktora może być powiązana z właściwości zamapowanych lub do różnych rodzajów usług w celu ułatwienia zachowania, takich jak ładowanych z opóźnieniem.

NOTE

Począwszy od programu EF Core 2.1 wszystkie powiązania Konstruktor jest przez Konwencję. Konfiguracja określonego konstruktora do użycia jest planowana w przyszłej wersji.

Powiązywanie właściwości zamapowanych

Należy rozważyć typowy modelu/wpis w blogu:

```
public class Blog
{
    public int Id { get; set; }

    public string Name { get; set; }
    public string Author { get; set; }

    public ICollection<Post> Posts { get; } = new List<Post>();
}

public class Post
{
    public int Id { get; set; }

    public string Title { get; set; }
    public string Content { get; set; }
    public DateTime PostedOn { get; set; }

    public Blog Blog { get; set; }
}
```

Jeśli programu EF Core tworzy wystąpienia tych typów, takich jak wyniki zapytania, jest będzie pierwsze wywołanie domyślnego konstruktora bez parametrów i następnie ustawi wartość każdej właściwości z bazy danych. Jednak jeśli programu EF Core znajdzie sparametryzowania konstruktora przy użyciu nazwy i typy, które pasują do właściwości parametrów mapowane właściwości, a następnie zamiast tego będzie wywoływać sparametryzowanego konstruktora o wartości dla tych właściwości, a nie ustawi każdej właściwości jawnie. Na przykład:

```

public class Blog
{
    public Blog(int id, string name, string author)
    {
        Id = id;
        Name = name;
        Author = author;
    }

    public int Id { get; set; }

    public string Name { get; set; }
    public string Author { get; set; }

    public ICollection<Post> Posts { get; } = new List<Post>();
}

public class Post
{
    public Post(int id, string title, DateTime postedOn)
    {
        Id = id;
        Title = title;
        PostedOn = postedOn;
    }

    public int Id { get; set; }

    public string Title { get; set; }
    public string Content { get; set; }
    public DateTime PostedOn { get; set; }

    public Blog Blog { get; set; }
}

```

Niektóre kwestie, należy zwrócić uwagę:

- Nie wszystkie właściwości konieczne parametry konstruktora. Na przykład właściwość Post.Content nie jest ustawiany przez żadnych parametrów konstruktora, więc programu EF Core zostanie ustawiony po wywołaniu konstruktora w normalny sposób.
- Nazwy i typy parametrów muszą być zgodne typy właściwości i nazw, z tą różnicą, że właściwości mogą być Pascal — z uwzględnieniem wielkości liter podczas, gdy parametry są w formacie camelcase.
- EF Core nie można ustawić właściwości nawigacji (na przykład blogu lub wpisy powyżej) przy użyciu konstruktora.
- Konstruktor może być publiczne, prywatne, lub innych ułatwień dostępu.

Właściwości tylko do odczytu

Gdy właściwości są ustawiane za pośrednictwem konstruktora sensowne może się niektóre z nich w trybie tylko do odczytu. Obsługuje tę funkcję programu EF Core, ale jest kilka rzeczy, zwróć uwagę na:

- Właściwości bez metod ustawiających nie są zamapowane przez Konwencję. (To zwykle właściwości, które nie powinny być mapowane, takich jak obliczone właściwości mapy.)
- Przy użyciu automatycznie generowanego klucza wartości wymaga właściwości klucza, która jest odczytu i zapisu, ponieważ wartość klucza musi być ustawiona przez generator kluczy, podczas wstawiania nowych jednostek.

Prosty sposób, aby uniknąć tych rzeczy jest używać prywatnych metod ustawiających. Na przykład:

```
public class Blog
{
    public Blog(int id, string name, string author)
    {
        Id = id;
        Name = name;
        Author = author;
    }

    public int Id { get; private set; }

    public string Name { get; private set; }
    public string Author { get; private set; }

    public ICollection<Post> Posts { get; } = new List<Post>();
}

public class Post
{
    public Post(int id, string title, DateTime postedOn)
    {
        Id = id;
        Title = title;
        PostedOn = postedOn;
    }

    public int Id { get; private set; }

    public string Title { get; private set; }
    public string Content { get; set; }
    public DateTime PostedOn { get; private set; }

    public Blog Blog { get; set; }
}
```

EF Core uznaje prywatnej setter właściwości odczytu / zapisu, co oznacza, że wszystkie właściwości są mapowane tak jak poprzednio, i klawisz mogą nadal być generowane przez Magazyn.

Alternatywa dla użycia prywatnych metod ustawiających jest właściwości tak naprawdę tylko do odczytu i Dodaj mapowanie dokładniejsze w OnModelCreating. Podobnie niektóre właściwości mogą całkowicie usunięty i zastąpiony tylko pola. Na przykład należy wziąć pod uwagę te typy jednostek:

```

public class Blog
{
    private int _id;

    public Blog(string name, string author)
    {
        Name = name;
        Author = author;
    }

    public string Name { get; }
    public string Author { get; }

    public ICollection<Post> Posts { get; } = new List<Post>();
}

public class Post
{
    private int _id;

    public Post(string title, DateTime postedOn)
    {
        Title = title;
        PostedOn = postedOn;
    }

    public string Title { get; }
    public string Content { get; set; }
    public DateTime PostedOn { get; }

    public Blog Blog { get; set; }
}

```

I tej konfiguracji w OnModelCreating:

```

protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Blog>(
        b =>
        {
            b.HasKey("_id");
            b.Property(e => e.Author);
            b.Property(e => e.Name);
        });

    modelBuilder.Entity<Post>(
        b =>
        {
            b.HasKey("_id");
            b.Property(e => e.Title);
            b.Property(e => e.PostedOn);
        });
}

```

Kwestii, które należy zwrócić uwagę:

- Klucz "property" jest polem. Nie jest `readonly` tak, aby klucze generowane przez Magazyn mogły być używane.
- Inne właściwości są właściwości tylko do odczytu ustawiana tylko w konstruktorze.
- Jeśli wartość klucza podstawowego tylko nigdy nie jest ustawiony przez EF lub odczytu z bazy danych, następnie nie ma potrzeby umieszczenie go w konstruktorze. Pozostawia klucz "property" jako prostym polem i sprawia, że jasne, że jej nie powinna być ustawiona jawnie podczas tworzenia nowego blogów i wpisów.

NOTE

Ten kod będzie skutkować kompilatora ostrzeżenie "169" wskazujący, że pole nigdy nie jest używany. Może być zostało zignorowane, ponieważ w rzeczywistości programu EF Core jest za pomocą pola w sposób extralinguistic.

Wprowadzanie usługi

EF Core mogą umożliwić również iniekcję "usługi" do konstruktora typu jednostki. Na przykład że może wprowadzone:

- `DbContext` -wystąpienie kontekstu bieżącego można także wpisać jako pochodny typu DbContext
- `ILazyLoader` -Usługa ładowanych z opóźnieniem — zobacz [dokumentacji powolne ładowanie](#) Aby uzyskać więcej informacji
- `Action<object, string>` — obiekt delegowany ładowanych z opóźnieniem — zobacz [dokumentacji powolne ładowanie](#) Aby uzyskać więcej informacji
- `IEntityType` -metadanych programu EF Core skojarzonych z tym typem jednostki

NOTE

Począwszy od programu EF Core 2.1 może zostać wprowadzona tylko usługi znane przez platformę EF Core. Obsługa wprowadzanie usługi aplikacji jest rozważane w przyszłej wersji.

Na przykład można wprowadzonego typu DbContext na selektywny dostęp do bazy danych, aby uzyskać informacje o powiązanych jednostek bez ładowania ich wszystkich. W poniższym przykładzie służy do uzyskania liczby wpisów w blogu bez ładowania wpisy:

```

public class Blog
{
    public Blog()
    {
    }

    private Blog(BloggingContext context)
    {
        Context = context;
    }

    private BloggingContext Context { get; set; }

    public int Id { get; set; }
    public string Name { get; set; }
    public string Author { get; set; }

    public ICollection<Post> Posts { get; set; }

    public int PostsCount
        => Posts?.Count
        ?? Context?.Set<Post>().Count(p => Id == EF.Property<int?>(p, "BlogId"))
        ?? 0;
}

public class Post
{
    public int Id { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }
    public DateTime PostedOn { get; set; }

    public Blog Blog { get; set; }
}

```

Kilka istotnych kwestii na ten temat:

- Konstruktor jest prywatny, ponieważ tylko nigdy nie jest wywoływana przez platformę EF Core i ma innego konstruktora publicznego do użytku ogólnego.
- Kod przy użyciu usługi wprowadzonego (kontekstu) jest obrony przed nim są `null` sposób obsługiwać przypadki, gdzie programu EF Core nie jest tworzone wystąpienie.
- Ponieważ usługa jest przechowywana we właściwości odczytu/zapisu zostaną zresetowane, gdy jednostka jest dołączony do nowego wystąpienia kontekstu.

WARNING

Wprowadzanie `DbContext` następujące jest często uznawana za zapobieganie wzorca od czasu jej couplek typów jednostek bezpośrednio do programu EF Core. Zastanów się uważnie wszystkie opcje przed przy użyciu iniekcji usługi w następujący sposób.

Posiadane typy jednostek

08.01.2019 • 11 minutes to read • [Edit Online](#)

NOTE

Ta funkcja jest nowa w programie EF Core 2.0.

EF Core umożliwia modelu typów jednostek, które mogą być wyświetlane tylko dla właściwości nawigacji innych typów jednostek. Są to tak zwane *posiadane typy jednostek*. Obiekt zawierający typ jednostki należące do firmy jest jego *właściciela*.

Jawne konfiguracji

Własność jednostki typy nigdy nie znajdują się przez platformę EF Core w modelu przez Konwencję. Można użyć `OwesOne` method in Class metoda `OnModelCreating` lub dodawać adnotacje do typu z `ownedAttribute` (nowe w programie EF Core 2.1) Aby skonfigurować typ jako typ należące do firmy.

W tym przykładzie `StreetAddress` jest typem przy użyciu żadnej właściwości tożsamości. Aby określić adres wysyłkowy dla określonej kolejności służy jako właściwość typu zamówienia.

Możemy użyć `OwnedAttribute` traktowanie jej jako należących do jednostki, gdy występuje do innego typu jednostki:

```
[Owned]
public class StreetAddress
{
    public string Street { get; set; }
    public string City { get; set; }
}
```

```
public class Order
{
    public int Id { get; set; }
    public StreetAddress ShippingAddress { get; set; }
}
```

Istnieje również możliwość użycia `OwesOne` method in Class metoda `OnModelCreating` Aby określić, że `ShippingAddress` właściwość jest własnością jednostki `Order` typu jednostki i skonfigurować dodatkowe aspekty, jeśli to konieczne.

```
modelBuilder.Entity<Order>().OwesOne(p => p.ShippingAddress);
```

Jeśli `ShippingAddress` właściwość jest prywatna w `Order` typu, można użyć ciągu wersję `OwesOne` metody:

```
modelBuilder.Entity<Order>().OwesOne(typeof(StreetAddress), "ShippingAddress");
```

Zobacz [pełny przykład projektu](#) Aby uzyskać dodatkowy kontekst.

Niejawne kluczy

Posiadane typy skonfigurowano `OwesOne` lub odnalezione za pomocą nawigacji odwołania zawsze mieć relację jeden do jednego z właścicielem, dlatego nie potrzebują własne wartości klucza jako wartości klucza obcego są unikatowe. W poprzednim przykładzie `StreetAddress` typu nie trzeba zdefiniować właściwość klucza.

Aby dowiedzieć się, jak EF Core śledzi te obiekty, warto Pomyśl, czy klucz podstawowy został utworzony jako [w tle właściwość](#) należących do typu. Wartość klucza wystąpienia typu należące do firmy będzie taka sama jak wartość klucza wystąpienia właściciela.

Kolekcji typów będących własnością

NOTE

Ta funkcja jest nowa w programie EF Core 2.2.

Aby skonfigurować kolekcji typów będących własnością `OwesMany` powinny być używane w `OnModelCreating`. Jednak klucza podstawowego nie zostanie skonfigurowany zgodnie z Konwencją, dlatego musi być jawnie określony. Powszechnie jest wprowadzanie klucza złożonego na użytek tych typów jednostek, zawierające klucz obcy do właściciela i dodatkowe unikatowa właściwość, która również może być w stanie w tle:

```
modelBuilder.Entity<Distributor>().OwesMany(p => p.ShippingCenters, a =>
{
    a.HasForeignKey("DistributorId");
    a.Property<int>("Id");
    a.HasKey("DistributorId", "Id");
});
```

Posiadane typy z dzielenia tabeli mapowania

Korzystając z relacyjnych baz danych, według Konwencji odwołania należące do typów są mapowane na tej samej tabeli jako właściciel. Ta migracja wymaga dzielenia tabeli na dwie: niektóre kolumny będą używane do przechowywania danych właściciela, a niektóre kolumny będą używane do przechowywania danych należących do jednostki. Jest to typowe funkcje znane jako dzielenie tabeli.

TIP

Posiadane typy przechowywane z dzielenia tabeli mogą być używane, podobnie jak złożonych typów są używane w EF6.

Zgodnie z Konwencją programu EF Core będzie nazwa kolumny bazy danych dla właściwości typu jednostki należące do firmy, zgodnie ze wzorcem `Navigation_OwnedEntityType`. W związku z tym `StreetAddress` właściwości będą wyświetlane w tabeli "Zamówienia" o nazwach "ShippingAddress_Street" i "ShippingAddress_City".

Możesz dołączyć `HasColumnName` metodę, aby zmienić te kolumny:

```
modelBuilder.Entity<Order>().OwesOne(
    o => o.ShippingAddress,
    sa =>
{
    sa.Property(p => p.Street).HasColumnName("ShipsToStreet");
    sa.Property(p => p.City).HasColumnName("ShipsToCity");
});
```

Udostępnianie tego samego typu .NET wśród wielu typów należące do firmy

Typ jednostki należące do firmy może być tego samego typu .NET jako inny typ jednostki należące do firmy, w związku z tym, że typ architektury .NET może nie być wystarczająca do identyfikowania typu należące do firmy.

W takich przypadkach staje się właściwość wskazuje od właściciela do jednostki należące do firmy *Definiowanie nawigacji* typu jednostki należące do firmy. Z perspektywy programu EF Core definiujące Nawigacja jest częścią tożsamości typu wraz z typem platformy .NET.

Na przykład w następującej klasie `ShippingAddress` i `BillingAddress` są tego samego typu .NET `StreetAddress`:

```
public class OrderDetails
{
    public DetailedOrder Order { get; set; }
    public StreetAddress BillingAddress { get; set; }
    public StreetAddress ShippingAddress { get; set; }
}
```

Aby dowiedzieć się, jak EF Core pomoże wyróżnić śledzonych wystąpień tych obiektów, może być wyobrazić sobie, że definiujące nawigacji stał się częścią klucza wystąpienia obok wartości klucza właściciela i typ architektury .NET typu należące do firmy.

Zagnieżdżone typy należące do firmy

W tym przykładzie `OrderDetails` właścicielem `BillingAddress` i `ShippingAddress`, które są zarówno `StreetAddress` typów. Następnie `OrderDetails` jest właściwą `DetailedOrder` typu.

```
public class DetailedOrder
{
    public int Id { get; set; }
    public OrderDetails OrderDetails { get; set; }
    public OrderStatus Status { get; set; }
}
```

```
public enum OrderStatus
{
    Pending,
    Shipped
}
```

Oprócz zagnieżdżonych typów należące do firmy typem należące do firmy można odwoływać się do regularnego jednostki, tak długo, jak należących do jednostki znajduje się na stronie zależne może być właścicielem lub innej jednostki. Ta funkcja ustawia EF6 posiadane typy jednostek oprócz typów złożonych.

```
public class OrderDetails
{
    public DetailedOrder Order { get; set; }
    public StreetAddress BillingAddress { get; set; }
    public StreetAddress ShippingAddress { get; set; }
}
```

Istnieje możliwość łańcucha `OwnsOne` metody w wywołaniu fluent, aby skonfigurować ten model:

```
modelBuilder.Entity<DetailedOrder>().OwnsOne(p => p.OrderDetails, od =>
{
    od.OwnsOne(c => c.BillingAddress);
    od.OwnsOne(c => c.ShippingAddress);
});
```

Istnieje również możliwość można osiągnąć używając tej samej kolejności `OwnedAttribute` zarówno `OrderDetails` i `StreetAddress`.

Przechowywanie należące do typów w oddzielnych tabelach

Również inaczej niż w przypadku typów złożonych EF6 posiadane typy mogą być przechowywane w osobnej tabeli od właściciela. Aby zastąpić Konwencji, która mapuje typ należących do tej samej tabeli jako właściciela, możesz po prostu wywołać `ToTable` i podaj inną nazwę tabeli. Poniższy przykład zmapuje `OrderDetails` i jego dwa adresy do osobnej tabeli z `DetailedOrder`:

```
modelBuilder.Entity<DetailedOrder>().OwnsOne(p => p.OrderDetails, od =>
{
    od.OwnsOne(c => c.BillingAddress);
    od.OwnsOne(c => c.ShippingAddress);
    od.ToTable("OrderDetails");
});
```

Wykonywanie zapytania dotyczącego typów należące do firmy

Podczas wykonywania zapytań dotyczących właściciela należących do typów będą uwzględniane domyślnie. Nie jest konieczne użycie `Include` metody, nawet jeśli posiadane typy są przechowywane w osobnej tabeli. Na podstawie modelu opisany wcześniej, następujące zapytanie zwróci `Order`, `OrderDetails` a dwa należące do `StreetAddresses` z bazy danych:

```
var order = context.DetailedOrders.First(o => o.Status == OrderStatus.Pending);
Console.WriteLine($"First pending order will ship to: {order.OrderDetails.ShippingAddress.City}");
```

Ograniczenia

Niektórych z tych ograniczeń mają zasadnicze znaczenie jak należących do pracy typów jednostek, ale niektóre inne ograniczenia, że firma Microsoft może być niemożliwe do usunięcia w przyszłych wersjach:

Ograniczenia według projektu

- Nie można utworzyć `DbSet<T>` dla typu należące do firmy
- Nie można wywołać `Entity<T>()` z typem należące do firmy na `ModelBuilder`

Bieżący wad

- Hierarchii dziedziczenia, które obejmują posiadane typy jednostek nie są obsługiwane.
- Odwołanie do tego celu posiadane typy jednostek nie może być pusty, chyba że jawnie są mapowane na tabelę oddzielnych od właściciela
- Wystąpienia elementu posiadane typy jednostek nie może być współużytkowana przez wiele właścicieli (jest to dobrze znanych scenariusz obiektów wartości, które nie mogą zostać zaimplementowane przy użyciu posiadane typy jednostek)

Braków w poprzednich wersjach

- W programie EF Core 2.0 tego celu należące do typów jednostek nie można zadeklarować w typach pochodny

jednostki, chyba że jednostki należące do firmy są jawnie mapowane do osobnej tabeli z hierarchii właściciela. To ograniczenie zostało usunięte w programie EF Core 2.1

- W programie EF Core 2.0 i 2.1 odwoływać się tylko były obsługiwane tego typom należące do firmy. To ograniczenie zostało usunięte w programie EF Core 2.2

Typy zapytań

08.12.2018 • 6 minutes to read • [Edit Online](#)

NOTE

Ta funkcja jest nowa w programie EF Core 2.1

Oprócz typów jednostek, mogą zawierać model programu EF Core typy zapytań, który może służyć do przeprowadzania zapytań bazy danych dla danych, które nie są mapowane na typy jednostek.

Porównanie typów zapytań do typów jednostek

Typy zapytań są podobne do typów jednostek, w tym ich:

- Można dodać do modelu albo w `OnModelCreating` lub za pośrednictwem "set" właściwości pochodnej `DbContext`.
- Obsługuje wiele tych samych funkcji mapowania, takich jak właściwości mapowania i nawigację dziedziczenia. W sklepach relacyjnych można skonfigurować obiektów bazy danych docelowych i kolumn za pomocą metody interfejsu API fluent lub adnotacji danych.

Jednak różnią się one od jednostki typy w tym ich:

- Nie wymagają klucza do zdefiniowania.
- Nigdy nie są śledzone zmiany na `DbContext` i w związku z tym są nigdy nie wstawione, zaktualizowane lub usunięte w bazie danych.
- Nigdy nie są odnajdywane przez Konwencję.
- Tylko obsługują podzbior funkcji map nawigacji — w szczególności:
 - Nigdy nie mogą one działać jako główny koniec relacji.
 - Może zawierać tylko właściwości nawigacji odwołania, wskazując jednostek.
 - Jednostki nie może zawierać właściwości nawigacji, aby typy zapytań.
- Szybkiego reagowania na element `ModelBuilder` przy użyciu `Query` metody zamiast `Entity` metody.
- Są mapowane na `DbContext` za pośrednictwem właściwości typu `DbQuery<T>` zamiast `DbSet<T>`.
- Są mapowane na obiekty bazy danych przy użyciu `ToView` metody, zamiast `ToTable`.
- Mogą być mapowane na *kwerendy* — Definiowanie zapytanie jest zapytaniem dodatkowej zadeklarowanych w modelu, który działa źródła danych dla typu zapytania.

Scenariusze użytkowania

Niektóre scenariusze użycia główne typy zapytań to:

- Służy jako typ zwracany dla ad hoc `FromSql()` zapytania.
- Mapowanie na widoki baz danych.
- Mapowania tabel, które nie mają zdefiniowany klucz podstawowy.
- Mapowanie do zapytań zdefiniowanych w modelu.

Mapowanie na obiekty bazy danych

Mapowanie typu zapytania do obiektu bazy danych jest osiągane przy użyciu `ToView` wygodnego interfejsu API. Z

perspektywy programu EF Core jest podany w tej metodzie obiekt bazy danych *widoku*, co oznacza, że jest ona traktowana jako źródła zapytań tylko do odczytu i nie może być elementem docelowym aktualizacji, wstawiania lub operacje usuwania. Jednak oznacza to, że obiekt bazy danych jest faktycznie wymagana do widoku bazy danych — może też być tabeli bazy danych, które będą traktowane jako tylko do odczytu. Z drugiej strony, dla typów jednostek programu EF Core przyjęto założenie, że obiektu bazy danych określony w `ToTable` metoda może być traktowana jako *tabeli*, co oznacza, że mogą być używane jako źródło zapytania, ale wskazywane przez aktualizację, usuwanie i wstawianie operacje. W rzeczywistości, można określić nazwy widoku bazy danych w `ToTable` i wszystko powinno działać prawidłowo tak długo, jak widok jest skonfigurowany jako nadaje się do aktualizacji w bazie danych.

Przykład

Poniższy przykład pokazuje, jak zapytania widoku bazy danych za pomocą typu zapytania.

TIP

Przykład użyty w tym artykule można zobaczyć w witrynie GitHub.

Najpierw należy zdefiniować prosty model blogu i Post:

```
public class Blog
{
    public int BlogId { get; set; }
    public string Name { get; set; }
    public string Url { get; set; }
    public ICollection<Post> Posts { get; set; }
}

public class Post
{
    public int PostId { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }
    public int BlogId { get; set; }
}
```

Następnie zdefiniuj widok prostej bazy danych, który umożliwi nam zapytania liczby stanowisk skojarzonych z każdym blogiem:

```
db.Database.ExecuteSqlCommand(
    @"CREATE VIEW View_BlogPostCounts AS
        SELECT b.Name, Count(p.PostId) as PostCount
        FROM Blogs b
        JOIN Posts p on p.BlogId = b.BlogId
        GROUP BY b.Name");
```

Następnie zdefiniuj klasę zawierającą wyniki z widoku bazy danych:

```
public class BlogPostsCount
{
    public string BlogName { get; set; }
    public int PostCount { get; set; }
}
```

Firma Microsoft Skonfiguruj typ zapytania w `OnModelCreating` przy użyciu `modelBuilder.Query<T>` interfejsu API. Używamy standardowej konfiguracji fluent API do skonfigurowania mapowania dla typu zapytania:

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder
        .Query<BlogPostsCount>().ToView("View_BlogPostCounts")
        .Property(v => v.BlogName).HasColumnName("Name");
}
```

Na koniec mamy zapytania widoku bazy danych w sposób standardowy:

```
var postCounts = db.BlogPostCounts.ToList();

foreach (var postCount in postCounts)
{
    Console.WriteLine($"{postCount.BlogName} has {postCount.PostCount} posts.");
    Console.WriteLine();
}
```

TIP

Należy zauważyć, że również zdefiniowaliśmy właściwości zapytania poziom kontekstu (DbQuery) do działania jako główne zapytań dla tego typu.

Przełączanie pomiędzy wiele modeli za pomocą tego samego typu DbContext

28.08.2018 • 2 minutes to read • [Edit Online](#)

Wbudowany model `OnModelCreating` wystarczą właściwości dla kontekstu można zmienić, jak zbudowane jest model. Na przykład może zostać wykorzystana do wykluczenia niektórych właściwości:

```
public class DynamicContext : DbContext
{
    public bool? IgnoreIntProperty { get; set; }

    public DbSet<ConfigurableEntity> Entities { get; set; }

    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
        => optionsBuilder
            .UseInMemoryDatabase("DynamicContext")
            .ReplaceService<IModelCacheKeyFactory, DynamicModelCacheKeyFactory>();

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        if (IgnoreIntProperty.HasValue)
        {
            if (IgnoreIntProperty.Value)
            {
                modelBuilder.Entity<ConfigurableEntity>().Ignore(e => e.IntProperty);
            }
            else
            {
                modelBuilder.Entity<ConfigurableEntity>().Ignore(e => e.StringProperty);
            }
        }
    }
}
```

IModelCacheKeyFactory

Jednak jeśli chcesz wypróbować, wykonując powyższe bez wprowadzania dodatkowych zmian otrzymamy ten sam model za każdym razem, gdy nowy kontekst jest tworzona dla dowolnej wartości `IgnoreIntProperty`. Jest to spowodowane przez model używa EF, aby zwiększyć wydajność, wywołując tylko mechanizmu buforowania `OnModelCreating` raz i buforowanie modelu.

Domyślnie EF przyjęto założenie, że we wszystkich kontekstach danego typu modelu będą takie same. Aby to osiągnąć, domyślna implementacja klasy `IModelCacheKeyFactory` zwraca klucz, zawierający tylko typ kontekstu. Aby zmienić to musisz zastąpić `IModelCacheKeyFactory` usługi. Nowa implementacja musi zwracać obiekt, który można porównać do innych kluczy modelu przy użyciu `Equals` metodę, która uwzględnia wszystkie zmienne, które wpływają na modelu:

```
public class DynamicModelCacheKeyFactory : IModelCacheKeyFactory
{
    public object Create(DbContext context)
    {
        if (context is DynamicContext dynamicContext)
        {
            return (context.GetType(), dynamicContext.IgnoreIntProperty);
        }
        return context.GetType();
    }
}
```

Dane przestrzenne

17.11.2018 • 12 minutes to read • [Edit Online](#)

NOTE

Ta funkcja jest nowa w programie EF Core 2.2.

Dane przestrzenne reprezentują lokalizację fizyczną i kształt obiektów. Wiele baz danych zapewniają obsługę tego rodzaju danych, więc może być indeksowana i zapytania wraz z innymi danymi. Typowe scenariusze obejmują tworzenie zapytań dotyczących obiektów w określonej odległości od lokalizacji lub zaznaczając ten obiekt, którego obramowanie zawiera danej lokalizacji. EF Core obsługuje Mapowanie typów danych przestrzennych przy użyciu [NetTopologySuite](#) przestrzenne biblioteki.

Instalowanie programu

Aby można było używać danych przestrzennych z programem EF Core, musisz zainstalować odpowiedni pakiet NuGet pomocniczych. Pakiet, który należy zainstalować zależy od dostawcy, którego używasz.

EF CORE DOSTAWCY	PRZESTRZENNE PAKIETU NUGET
Microsoft.EntityFrameworkCore.SqlServer	Microsoft.EntityFrameworkCore.SqlServer.NetTopologySuite
Microsoft.EntityFrameworkCore.Sqlite	Microsoft.EntityFrameworkCore.Sqlite.NetTopologySuite
Microsoft.EntityFrameworkCore.InMemory	NetTopologySuite
Npgsql.EntityFrameworkCore.PostgreSQL	Npgsql.EntityFrameworkCore.PostgreSQL.NetTopologySuite

Odtwarzanie

Przestrzenne NuGet również pakietów Włącz [odtwarzanie](#) modeli przy użyciu właściwości przestrzennych, ale trzeba zainstalować pakiet **przed** systemem `Scaffold-DbContext` lub `dotnet ef dbcontext scaffold`. Jeśli nie istnieje, otrzymasz ostrzeżenia dotyczące nie możesz znaleźć mapowania typów kolumn i kolumny zostaną pominięte.

NetTopologySuite (NTS)

NetTopologySuite jest biblioteką przestrzennych dla platformy .NET. EF Core umożliwia mapowanie danych przestrzennych typów w bazie danych przy użyciu typów nktę przerwania w modelu.

Aby włączyć mapowanie typów przestrzennych za pośrednictwem nktę przerwania, należy wywołać metodę `UseNetTopologySuite` dla dostawcy typu `DbContext` opcje konstruktora. Na przykład z programem SQL Server możesz wywołać go następująco.

```
optionsBuilder.UseSqlServer(  
    @"Data Source=(localdb)\MSSQLLocalDB;Initial Catalog=WideWorldImporters",  
    x => x.UseNetTopologySuite());
```

Istnieją różne typy danych przestrzennych. Jakiego typu, możesz użyć zależy od typów kształtami, które chcesz

zezwolić. Oto hierarchii typów który przerwania, które służy do właściwości w modelu. Są one umieszczone w ramach `NetTopologySuite.Geometries` przestrzeni nazw. Odpowiednich interfejsów w pakiecie GeoAPI (`GeoAPI.Geometries` przestrzeni nazw) może również służyć.

- Geometrii
 - Punkt
 - LineString
 - Wielokąt
 - GeometryCollection
 - MultiPoint
 - MultiLineString
 - MultiPolygon

WARNING

`CircularString`, `CompoundCurve` i `CurvePolygon` nie są obsługiwane przez który przerwania.

Przy użyciu podstawowego typu Geometria umożliwia dowolny typ kształtu do określonych we właściwości.

Następujących klas jednostek może być używany do mapowania z tabelami w [Wide World Importers przykładowej bazy danych](#).

```
[Table("Cities", Schema = "Application")]
class City
{
    public int CityID { get; set; }

    public string CityName { get; set; }

    public IPoint Location { get; set; }
}

[Table("Countries", Schema = "Application")]
class Country
{
    public int CountryID { get; set; }

    public string CountryName { get; set; }

    // Database includes both Polygon and MultiPolygon values
    public IGeometry Border { get; set; }
}
```

Tworzenie wartości

Można używać konstruktorów do tworzenia obiektów geometrii; jednak który przerwania zaleca używanie fabrykę geometrii zamiast tego. To pozwala określić domyślną SRID (używane przez współrzędne system odwołania przestrzennego) i zapewnia kontrolę nad bardziej zaawansowanych np. model precyzji (używane podczas obliczeń) i sekwencji współrzędnych (określa, które rzędne — wymiarów i miary — są dostępne).

```
var geometryFactory = NtsGeometryServices.Instance.CreateGeometryFactory(srid: 4326);
var currentLocation = geometryFactory.CreatePoint(-122.121512, 47.6739882);
```

NOTE

4326 odnosi się do WGS 84 standard używany w GPS i innych systemach geograficznego.

Długość i szerokość geograficzną

Współrzędne nkty przerwania są na podstawie wartości X i Y. Do reprezentowania długości i szerokości geograficznej, na użytku X dla współrzędnych i Y szerokości geograficznej. Należy zauważać, że jest to **Wstecz z latitude, longitude** format, w którym zwykle widzieć te wartości.

SRID ignorowany podczas operacji klienta

Nkty przerwania ignoruje wartości SRID podczas wykonywania operacji. Przyjęto założenie, współrzędnych planarnych. Oznacza to, że w przypadku określenia współrzędnych pod względem długości i szerokości geograficznej, niektóre wartości takie jak odległości, długość i obszar będzie w stopniach, liczniki nie oceniane przez klienta. Dla bardziej zrozumiałe wartości, należy najpierw projektu wspólczędne miejsca inny układ współrzędnych przy użyciu biblioteki, takich jak [ProjNet4GeoAPI](#) przed obliczeniem te wartości.

W przypadku operacji serwera oceniane przez platformę EF Core przy użyciu programu SQL, jednostka wynik będzie określana przez bazę danych.

Oto przykład użycia ProjNet4GeoAPI, aby obliczyć odległość między dwoma miast.

```

static class GeometryExtensions
{
    static readonly IGeometryServices _geometryServices = NtsGeometryServices.Instance;
    static readonly ICoordinateSystemServices _coordinateSystemServices
        = new CoordinateSystemServices(
            new CoordinateSystemFactory(),
            new CoordinateTransformationFactory(),
            new Dictionary<int, string>
            {
                // Coordinate systems:

                // (3857 and 4326 included automatically)

                // This coordinate system covers the area of our data.
                // Different data requires a different coordinate system.
                [2855] =
                @"
                    PROJCS[""NAD83(HARN) / Washington North"",
                        GEOGCS[""NAD83(HARN)"",
                            DATUM[""NAD83_High_Accuracy_Regional_Network"",
                                SPHEROID[""GRS 1980"",6378137,298.257222101,
                                    AUTHORITY[""EPSG"",""7019""}],
                                AUTHORITY[""EPSG"",""6152""]],
                            PRIMEM[""Greenwich"",0,
                                AUTHORITY[""EPSG"",""8901""]],
                            UNIT[""degree"",0.01745329251994328,
                                AUTHORITY[""EPSG"",""9122""]],
                                AUTHORITY[""EPSG"",""4152""]],
                            PROJECTION[""Lambert_Conformal_Conic_2SP""],
                            PARAMETER[""standard_parallel_1"",48.73333333333333],
                            PARAMETER[""standard_parallel_2"",47.5],
                            PARAMETER[""latitude_of_origin"",47],
                            PARAMETER[""central_meridian"",-120.833333333333],
                            PARAMETER[""false_easting"",500000],
                            PARAMETER[""false_northing"",0],
                            UNIT[""metre"",1,
                                AUTHORITY[""EPSG"",""9001""]],
                                AUTHORITY[""EPSG"",""2855""]]
                            "
                );
            }
        }

    public static IGeometry ProjectTo(this IGeometry geometry, int srid)
    {
        var geometryFactory = _geometryServices.CreateGeometryFactory(srid);
        var transformation = _coordinateSystemServices.CreateTransformation(geometry.SRID, srid);

        return GeometryTransform.TransformGeometry(
            geometryFactory,
            geometry,
            transformation.MathTransform);
    }
}

```

```

var seattle = new Point(-122.333056, 47.609722) { SRID = 4326 };
var redmond = new Point(-122.123889, 47.669444) { SRID = 4326 };

var distance = seattle.ProjectTo(2855).Distance(redmond.ProjectTo(2855));

```

Wykonanie zapytania o dane

W programie LINQ niektóre przerwania metody i właściwości dostępne jako funkcje bazy danych ma być tłumaczone na SQL. Na przykład odległość i zawiera metody są tłumaczone w następujących zapytań. Tabeli na końcu tego artykułu przedstawiono składniki, które są obsługiwane przez różnych dostawców programu EF Core.

```
var nearestCity = db.Cities
    .OrderBy(c => c.Location.Distance(currentLocation))
    .FirstOrDefault();

var currentCountry = db.Countries
    .FirstOrDefault(c => c.Border.Contains(currentLocation));
```

SQL Server

Jeśli używasz programu SQL Server, istnieją pewne dodatkowe rzeczy, na których należy wiedzieć.

Położenia geograficznego lub geometrii

Domyślnie przestrzenne właściwości są mapowane na `geography` kolumny w programie SQL Server. Aby użyć `geometry`, [skonfigurować typ kolumny](#) w modelu.

Pierścienie wielokąta lokalizacji geograficznej

Korzystając z `geography` typ kolumny, programu SQL Server nakłada dodatkowe wymagania dotyczące pierścienia zewnętrznego (lub powłoki) i posługiwianie się nimi pierścieniami (lub otworów). Pierścieniem musi być zorientowana zegara i wewnętrznych pierścieni zgodnie ze wskazówkami zegara. NTFS sprawdza poprawność to przed wysłaniem wartości w bazie danych.

FullGlobe

Program SQL Server ma typ geometrii niestandardowych do reprezentowania pełną całym świecie, korzystając z `geography` typ kolumny. Ma również sposób reprezentacji wielokątów oparte na świecie pełny (bez pierścieniem). Oba te są obsługiwane przez nkty przerwania.

WARNING

FullGlobe i wielokątów na jego podstawie nie są obsługiwane przez nkty przerwania.

Bazy danych SQLite

Poniżej przedstawiono niektóre dodatkowe informacje dotyczące tych przy użyciu systemu SQLite.

Instalowanie SpatiaLite

W Windows biblioteki natywnej mod_spatialite jest rozpowszechniany jako zależność pakietu NuGet. Platformami, trzeba go zainstalować osobno. Zazwyczaj jest to wykonywane przy użyciu Menedżera pakietów oprogramowania. Na przykład można użyć APT w systemie Ubuntu i Homebrew w systemie MacOS.

```
# Ubuntu
apt-get install libsqlite3-mod-spatialite

# macOS
brew install libspatialite
```

Konfigurowanie SRID

Kolumny SpatiaLite, muszą określić SRID według kolumny. Wartość domyślna to SRID `0`. Określ inną SRID, przy użyciu metody `ForSqliteHasSrid`.

```
modelBuilder.Entity<City>().Property(c => c.Location)
    .ForSqliteHasSrid(4326);
```

Wymiar

Podobnie jak SRID, wymiaru (lub kolumny rzędne) jest również określona jako część kolumny. Rzędne domyślne są X i Y. Włącz dodatkowe rzędne (Z i M) przy użyciu metody ForSqliteHasDimension.

```
modelBuilder.Entity<City>().Property(c => c.Location)
    .ForSqliteHasDimension(Ordinates.XYZ);
```

Operacje tłumaczenia

W poniższej tabeli przedstawiono, które elementy członkowskie metody przerwania są tłumaczone na SQL przez każdego dostawca programu EF Core.

NETTOPOLOGYSUITE	PROGRAM SQL SERVER (GEOMETRII)	PROGRAM SQL SERVER (GEOGRAFICZNE)	BAZY DANYCH SQLITE	NPGSQL
Geometry.Area	✓	✓	✓	✓
Geometry.AsBinary()	✓	✓	✓	✓
Geometry.AsText()	✓	✓	✓	✓
Geometry.Boundary	✓		✓	✓
Geometry.Buffer(double)	✓	✓	✓	✓
Geometry.Buffer(double, int)			✓	
Geometry.Centroid	✓		✓	✓
Geometry.Contains(Geometry)	✓	✓	✓	✓
Geometry.ConvexHull()	✓	✓	✓	✓
Geometry.CoveredBy(Geometry)			✓	✓
Geometry.Covers(Geometry)			✓	✓
Geometry.Crosses(Geometry)	✓		✓	✓
Geometry.Difference(Geometry)	✓	✓	✓	✓
Geometry.Dimension	✓	✓	✓	✓
Geometry.Disjoint(Geometry)	✓	✓	✓	✓

NET TOPOLOGYSUITE	PROGRAM SQL SERVER (GEOMETRII)	PROGRAM SQL SERVER (GEOGRAFICZNE)	BAZY DANYCH SQLITE	NPGSQL
Geometry.Distance(Geometry)	✓	✓	✓	✓
Geometry.Envelope	✓		✓	✓
Geometry.EqualsExact(Geometry)				✓
Geometry.EqualsTopologically(Geometry)	✓	✓	✓	✓
Geometry.GeometryType	✓	✓	✓	✓
Geometry.GetGeometryN(int)	✓		✓	✓
Geometry.InteriorPoint	✓		✓	
Geometry.Intersection(Geometry)	✓	✓	✓	✓
Geometry.Intersects(Geometry)	✓	✓	✓	✓
Geometry.IsEmpty	✓	✓	✓	✓
Geometry.IsSimple	✓		✓	✓
Geometry.IsValid	✓	✓	✓	✓
Geometry.IsWithinDistance(Geometry, double)	✓		✓	
Geometry.Length	✓	✓	✓	✓
Geometry.NumGeometries	✓	✓	✓	✓
Geometry.NumPoints	✓	✓	✓	✓
Geometry.OgcGeometryType	✓	✓	✓	
Geometry.Overlaps(Geometry)	✓	✓	✓	✓
Geometry.PointOnSurface	✓		✓	✓

NET TOPOLOGY SUITE	PROGRAM SQL SERVER (GEOMETRII)	PROGRAM SQL SERVER (GEOGRAFICZNE)	BAZY DANYCH SQLITE	NPGSQL
Geometry.Relate (Geometry, ciąg)	✓		✓	✓
Geometry.Reverse()			✓	✓
Geometry.SRID	✓	✓	✓	✓
Geometry.Symmetric Difference(Geometry)	✓	✓	✓	✓
Geometry.ToBinary()	✓	✓	✓	✓
Geometry.ToString()	✓	✓	✓	✓
Geometry.Touches(Ge ometry)	✓		✓	✓
Geometry.Union()			✓	
Geometry.Union(Geo metry)	✓	✓	✓	✓
Geometry.Within(Geo metry)	✓	✓	✓	✓
GeometryCollection.C ount	✓	✓	✓	✓
GeometryCollection [int]	✓	✓	✓	✓
LineString.Count	✓	✓	✓	✓
LineStringEndPoint	✓	✓	✓	✓
LineString.GetPointN(i nt)	✓	✓	✓	✓
LineString.IsClosed	✓	✓	✓	✓
LineString.IsRing	✓		✓	✓
LineString.StartPoint	✓	✓	✓	✓
MultiLineString.IsClos ed	✓	✓	✓	✓
Point.M	✓	✓	✓	✓
Point.X	✓	✓	✓	✓

NET TOPOLOGY SUITE	PROGRAM SQL SERVER (GEOMETRII)	PROGRAM SQL SERVER (GEOGRAFICZNE)	BAZY DANYCH SQLITE	NPGSQL
Point.Y	✓	✓	✓	✓
Point.Z	✓	✓	✓	✓
Polygon.ExteriorRing	✓	✓	✓	✓
Polygon.GetInteriorRingN(int)	✓	✓	✓	✓
Polygon.NumInteriorRings	✓	✓	✓	✓

Dodatkowe zasoby

- [Dane przestrzenne programu SQL Server](#)
- [Strona główna SpatiaLite](#)
- [Dokumentacja przestrzenne Npgsql](#)
- [Dokumentacja PostGIS](#)

Modelowanie relacyjnej bazy danych

28.08.2018 • 2 minutes to read • [Edit Online](#)

Ogólnie rzecz biorąc jest odpowiednie dla relacyjnych baz danych konfiguracji w tej sekcji. Metody rozszerzenia, pokazane tutaj staną się dostępne po zainstalowaniu dostawcy relacyjnej bazy danych (z powodu udostępnionej *Microsoft.EntityFrameworkCore.Relational* pakietu).

Mapowanie tabeli

28.08.2018 • 2 minutes to read • [Edit Online](#)

NOTE

Ogólnie rzecz biorąc jest odpowiednie dla relacyjnych baz danych konfiguracji w tej sekcji. Metody rozszerzenia, pokazane tutaj staną się dostępne po zainstalowaniu dostawcy relacyjnej bazy danych (z powodu udostępnionej `Microsoft.EntityFrameworkCore.Relational` pakietu).

Mapowanie tabeli identyfikuje dane, które tabeli powinien być odpytywane i zapisywane w bazie danych.

Konwencje

Zgodnie z Konwencją każdej jednostki będą konfigurowane tak, aby zamapować na tabelę z taką samą nazwą jak `DbSet< TEntity >` właściwość, która udostępnia jednostki w kontekście pochodnych. Jeśli nie `DbSet< TEntity >` jest dołączony do danej jednostki, nazwa klasy jest używana.

Adnotacje danych

Korzystanie z adnotacji danych, aby skonfigurować tabelę, która mapuje typu.

```
using System.ComponentModel.DataAnnotations.Schema;
```

```
[Table("blogs")]
public class Blog
{
    public int BlogId { get; set; }
    public string Url { get; set; }
}
```

Można również określić schemat, który należy do tabeli.

```
[Table("blogs", Schema = "blogging")]
public class Blog
{
    public int BlogId { get; set; }
    public string Url { get; set; }
}
```

Interfejs Fluent API

Interfejs Fluent API umożliwia skonfigurowanie tabeli, która mapuje typu.

```
using Microsoft.EntityFrameworkCore;
```

```
class MyContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Blog>()
            .ToTable("blogs");
    }
}

public class Blog
{
    public int BlogId { get; set; }
    public string Url { get; set; }
}
```

Można również określić schemat, który należy do tabeli.

```
modelBuilder.Entity<Blog>()
    .ToTable("blogs", schema: "blogging");
```

Mapowanie kolumn

28.08.2018 • 2 minutes to read • [Edit Online](#)

NOTE

Ogólnie rzecz biorąc jest odpowiednie dla relacyjnych baz danych konfiguracji w tej sekcji. Metody rozszerzenia, pokazane tutaj staną się dostępne po zainstalowaniu dostawcy relacyjnej bazy danych (z powodu udostępnionej `Microsoft.EntityFrameworkCore.Relational` pakietu).

Mapowanie kolumny określa dane, które kolumny powinien być odpytywane i zapisywane w bazie danych.

Konwencje

Zgodnie z Konwencją każda właściwość zostanie skonfigurowana do mapowania kolumny z taką samą nazwą jak właściwość.

Adnotacje danych

Korzystanie z adnotacji danych, aby skonfigurować kolumny, z którą właściwość jest zamapowana.

```
public class Blog
{
    [Column("blog_id")]
    public int BlogId { get; set; }
    public string Url { get; set; }
}
```

Interfejs Fluent API

Interfejs Fluent API umożliwia skonfigurowanie kolumny, z którą właściwość jest zamapowana.

```
class MyContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Blog>()
            .Property(b => b.BlogId)
            .HasColumnName("blog_id");
    }
}

public class Blog
{
    public int BlogId { get; set; }
    public string Url { get; set; }
}
```

Typy danych

28.08.2018 • 2 minutes to read • [Edit Online](#)

NOTE

Ogólnie rzecz biorąc jest odpowiednie dla relacyjnych baz danych konfiguracji w tej sekcji. Metody rozszerzenia, pokazane tutaj staną się dostępne po zainstalowaniu dostawcy relacyjnej bazy danych (z powodu udostępnionej `Microsoft.EntityFrameworkCore.Relational` pakietu).

Typ danych odnosi się do bazy danych określonego typu kolumny, z którą właściwość jest zamapowana.

Konwencje

Zgodnie z Konwencją dostawcy bazy danych wybiera typ danych, w zależności od typu CLR właściwości. On uwzględnia również inne metadane, takie jak skonfigurowanych [maksymalną długość](#), czy właściwość jest częścią klucza podstawowego, itp.

Na przykład program SQL Server używa `datetime2(7)` dla `DateTime` właściwości, a `nvarchar(max)` dla `string` właściwości (lub `nvarchar(450)` dla `string` właściwości, które są używane jako klucz).

Adnotacje danych

Korzystanie z adnotacji danych, aby określić dokładny typ danych dla kolumny.

Na przykład poniższy kod służy do konfigurowania `Url` jako ciąg znaków innego niż unicode o maksymalnej długości `200` i `Rating` jako dziesiętna z dokładnością do `5` i skalowanie z `2`.

```
public class Blog
{
    public int BlogId { get; set; }
    [Column(TypeName = "varchar(200)")]
    public string Url { get; set; }
    [Column(TypeName = "decimal(5, 2)")]
    public decimal Rating { get; set; }
}
```

Interfejs Fluent API

Można również określić ten sam typ danych dla kolumn za pomocą Fluent interfejsu API.

```
class MyContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Blog>(eb =>
        {
            eb.Property(b => b.Url).HasColumnType("varchar(200)");
            eb.Property(b => b.Rating).HasColumnType("decimal(5, 2)");
        });
    }
}

public class Blog
{
    public int BlogId { get; set; }
    public string Url { get; set; }
    public decimal Rating { get; set; }
}
```

Klucze podstawowe

28.08.2018 • 2 minutes to read • [Edit Online](#)

NOTE

Ogólnie rzecz biorąc jest odpowiednie dla relacyjnych baz danych konfiguracji w tej sekcji. Metody rozszerzenia, pokazane tutaj staną się dostępne po zainstalowaniu dostawcy relacyjnej bazy danych (z powodu udostępnionej `Microsoft.EntityFrameworkCore.Relational` pakietu).

Wprowadzono ograniczenie klucza podstawowego klucza dla każdego typu jednostki.

Konwencje

Zgodnie z Konwencją, klucz podstawowy w bazie danych będą miały nazwę nadaną `PK_<type name>`.

Adnotacje danych

Konkretnych aspektów relacyjnej bazy danych klucza podstawowego można skonfigurować przy użyciu adnotacji danych.

Interfejs Fluent API

Interfejs Fluent API umożliwiają skonfigurowanie nazwę ograniczenia klucza podstawowego w bazie danych.

```
class MyContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Blog>()
            .HasKey(b => b.BlogId)
            .HasName("PrimaryKey_BlogId");
    }
}

public class Blog
{
    public int BlogId { get; set; }
    public string Url { get; set; }
}
```

Schemat domyślny

28.08.2018 • 2 minutes to read • [Edit Online](#)

NOTE

Ogólnie rzecz biorąc jest odpowiednie dla relacyjnych baz danych konfiguracji w tej sekcji. Metody rozszerzenia, pokazane tutaj staną się dostępne po zainstalowaniu dostawcy relacyjnej bazy danych (z powodu udostępnionej `Microsoft.EntityFrameworkCore.Relational` pakietu).

Schemat domyślny jest schemat bazy danych, które obiekty zostaną utworzone w, jeśli schemat nie jest jawnie skonfigurowany dla tego obiektu.

Konwencje

Zgodnie z Konwencją dostawcy bazy danych będzie wybrać najbardziej odpowiedni schemat domyślny. Na przykład Microsoft SQL Server będzie używać `dbo` bazy danych SQLite i schematu nie będzie używać schematu (ponieważ schematy nie są obsługiwane w SQLite).

Adnotacje danych

Nie można ustawić domyślny schemat przy użyciu adnotacji danych.

Interfejs Fluent API

Aby określić domyślny schemat, można użyć interfejsu API Fluent.

```
class MyContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.HasDefaultSchema("blogging");
    }
}
```

Kolumny obliczane

28.08.2018 • 2 minutes to read • [Edit Online](#)

NOTE

Ogólnie rzecz biorąc jest odpowiednie dla relacyjnych baz danych konfiguracji w tej sekcji. Metody rozszerzenia, pokazane tutaj staną się dostępne po zainstalowaniu dostawcy relacyjnej bazy danych (z powodu udostępnionej `Microsoft.EntityFrameworkCore.Relational` pakietu).

Kolumna obliczana jest kolumną, której wartość jest obliczana w bazie danych. Kolumna obliczana można użyć innych kolumn w tabeli można obliczyć jej wartość.

Konwencje

Zgodnie z Konwencją kolumnach obliczanych nie są tworzone w modelu.

Adnotacje danych

Nie można skonfigurować przy użyciu adnotacji danych kolumn obliczanych.

Interfejs Fluent API

Interfejs Fluent API służy do określenia, czy właściwość powinna być zamapowana z kolumną obliczaną.

```
class MyContext : DbContext
{
    public DbSet<Person> People { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Person>()
            .Property(p => p.DisplayName)
            .HasComputedColumnSql("[LastName] + ', ' + [FirstName]");
    }
}

public class Person
{
    public int PersonId { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public string DisplayName { get; set; }
}
```

Sekwencje

28.08.2018 • 2 minutes to read • [Edit Online](#)

NOTE

Ogólnie rzecz biorąc jest odpowiednie dla relacyjnych baz danych konfiguracji w tej sekcji. Metody rozszerzenia, pokazane tutaj staną się dostępne po zainstalowaniu dostawcy relacyjnej bazy danych (z powodu udostępnionej `Microsoft.EntityFrameworkCore.Relational` pakietu).

Sekwencja generuje kolejnych wartości liczbowe w bazie danych. Sekwencje nie są skojarzone z określonej tabeli.

Konwencje

Zgodnie z Konwencją sekwencji nie są wprowadzane w modelu.

Adnotacje danych

Nie można skonfigurować przy użyciu adnotacji danych sekwencji.

Interfejs Fluent API

Interfejs Fluent API umożliwia tworzenie sekwencji w modelu.

```
class MyContext : DbContext
{
    public DbSet<Order> Orders { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.HasSequence<int>("OrderNumbers");
    }
}

public class Order
{
    public int OrderId { get; set; }
    public int OrderNo { get; set; }
    public string Url { get; set; }
}
```

Można również skonfigurować dodatkowe aspekty sekwencji, takie jak jego schematu, wartość początkową i przyrostu.

```
class MyContext : DbContext
{
    public DbSet<Order> Orders { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.HasSequence<int>("OrderNumbers", schema: "shared")
            .StartsAt(1000)
            .IncrementsBy(5);
    }
}
```

Gdy wprowadzono sekwencji służy do generowania wartości właściwości w modelu. Na przykład, można użyć [wartości domyślne](#) do wstawienia następnej wartości z sekwencji.

```
class MyContext : DbContext
{
    public DbSet<Order> Orders { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.HasSequence<int>("OrderNumbers", schema: "shared")
            .StartsAt(1000)
            .IncrementsBy(5);

        modelBuilder.Entity<Order>()
            .Property(o => o.OrderNo)
            .HasDefaultValueSql("NEXT VALUE FOR shared.OrderNumbers");
    }
}

public class Order
{
    public int OrderId { get; set; }
    public int OrderNo { get; set; }
    public string Url { get; set; }
}
```

Wartości domyślne

28.08.2018 • 2 minutes to read • [Edit Online](#)

NOTE

Ogólnie rzecz biorąc jest odpowiednie dla relacyjnych baz danych konfiguracji w tej sekcji. Metody rozszerzenia, pokazane tutaj staną się dostępne po zainstalowaniu dostawcy relacyjnej bazy danych (z powodu udostępnionej *Microsoft.EntityFrameworkCore.Relational* pakietu).

Wartość domyślna w kolumnie jest wartość, która zostanie wstawiony, jeśli jest wstawiany nowego wiersza, ale nie określono wartości dla kolumny.

Konwencje

Zgodnie z Konwencją wartość domyślna jest nieskonfigurowany.

Adnotacje danych

Nie można ustawić wartość domyślną, przy użyciu adnotacji danych.

Interfejs Fluent API

Interfejs Fluent API umożliwia określenie wartości domyślnej dla właściwości.

```
class MyContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Blog>()
            .Property(b => b.Rating)
            .HasDefaultValue(3);
    }
}

public class Blog
{
    public int BlogId { get; set; }
    public string Url { get; set; }
    public int Rating { get; set; }
}
```

Można również określić fragment SQL, który jest używany do obliczania wartości domyślnej.

```
class MyContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Blog>()
            .Property(b => b.Created)
            .HasDefaultValueSql("getdate()");
    }
}

public class Blog
{
    public int BlogId { get; set; }
    public string Url { get; set; }
    public DateTime Created { get; set; }
}
```

Indeksy

28.08.2018 • 2 minutes to read • [Edit Online](#)

NOTE

Ogólnie rzeczą biorąc jest odpowiednie dla relacyjnych baz danych konfiguracji w tej sekcji. Metody rozszerzenia, pokazane tutaj staną się dostępne po zainstalowaniu dostawcy relacyjnej bazy danych (z powodu udostępnionej `Microsoft.EntityFrameworkCore.Relational` pakietu).

Indeks w relacyjnej bazie danych jest mapowany na tę samą koncepcję jako indeks w obszarach podstawowych programu Entity Framework.

Konwencje

Zgodnie z Konwencją indeksy są nazywane `IX_<type name>_<property name>`. Dla indeksów złożonych `<property name>` staje się listę nazw właściwości oddzielonych znakiem podkreślenia.

Adnotacje danych

Indeksy nie można skonfigurować przy użyciu adnotacji danych.

Interfejs Fluent API

Aby skonfigurować nazwę indeksu, można użyć interfejsu API Fluent.

```
class MyContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Blog>()
            .HasIndex(b => b.Url)
            .HasName("Index_URL");
    }
}

public class Blog
{
    public int BlogId { get; set; }
    public string Url { get; set; }
}
```

Można również określić filtr.

```
class MyContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Blog>()
            .HasIndex(b => b.Url)
            .HasFilter("[Url] IS NOT NULL");
    }
}

public class Blog
{
    public int BlogId { get; set; }
    public string Url { get; set; }
}
```

Po dodaniu przy użyciu dostawcy programu SQL Server EF 'IS NOT NULL' filtrować wszystkie kolumny dopuszczające wartość null, które są częścią unikatowego indeksu. Aby zastąpić tę Konwencję, możesz podać `null` wartość.

```
class MyContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Blog>()
            .HasIndex(b => b.Url)
            .IsUnique()
            .HasFilter(null);
    }
}

public class Blog
{
    public int BlogId { get; set; }
    public string Url { get; set; }
}
```

Ograniczenia klucza obcego

28.08.2018 • 2 minutes to read • [Edit Online](#)

NOTE

Ogólnie rzeczą biorąc jest odpowiednie dla relacyjnych baz danych konfiguracji w tej sekcji. Metody rozszerzenia, pokazane tutaj staną się dostępne po zainstalowaniu dostawcy relacyjnej bazy danych (z powodu udostępnionej `Microsoft.EntityFrameworkCore.Relational` pakietu).

Wprowadzono ograniczenie klucza obcego dla każdej relacji w modelu.

Konwencje

Zgodnie z Konwencją ograniczenia klucza obcego są nazywane

`FK_<dependent type name>_<principal type name>_<foreign key property name>`. Złożone klucze obcych
`<foreign key property name>` staje się oddzielone znakiem podkreślenia listę nazw właściwości klucza obcego.

Adnotacje danych

Nazwy ograniczenia klucza obcego nie można skonfigurować przy użyciu adnotacji danych.

Interfejs Fluent API

Interfejs Fluent API umożliwia skonfigurowanie nazwę ograniczenia klucza obcego relacji.

```
class MyContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }
    public DbSet<Post> Posts { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Post>()
            .HasOne(p => p.Blog)
            .WithMany(b => b.Posts)
            .HasForeignKey(p => p.BlogId)
            .HasConstraintName("ForeignKey_Post_Blog");
    }
}

public class Blog
{
    public int BlogId { get; set; }
    public string Url { get; set; }

    public List<Post> Posts { get; set; }
}

public class Post
{
    public int PostId { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }

    public int BlogId { get; set; }
    public Blog Blog { get; set; }
}
```

Klucze alternatywne (ograniczenia unikatowe)

28.08.2018 • 2 minutes to read • [Edit Online](#)

NOTE

Ogólnie rzeczą biorąc jest odpowiednie dla relacyjnych baz danych konfiguracji w tej sekcji. Metody rozszerzenia, pokazane tutaj staną się dostępne po zainstalowaniu dostawcy relacyjnej bazy danych (z powodu udostępnionej `Microsoft.EntityFrameworkCore.Relational` pakietu).

Ograniczenia unique zostało wprowadzone dla każdego klucza alternatywnego w modelu.

Konwencje

Zgodnie z Konwencją indeksu i ograniczenia, które zostały wprowadzone dla klucza alternatywnego będą miały nazwę nadaną `AK_<type name>_<property name>`. Klucze alternatywne złożonego `<property name>` staje się listę nazw właściwości oddzielonych znakiem podkreślenia.

Adnotacje danych

Nie można skonfigurować ograniczeń UNIQUE przy użyciu adnotacji danych.

Interfejs Fluent API

Interfejs Fluent API umożliwia skonfigurowanie nazwę indeksu i ograniczenie klucza alternatywnego.

```
class MyContext : DbContext
{
    public DbSet<Car> Cars { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Car>()
            .HasAlternateKey(c => c.LicensePlate)
            .HasName("AlternateKey_LicensePlate");
    }
}

class Car
{
    public int CarId { get; set; }
    public string LicensePlate { get; set; }
    public string Make { get; set; }
    public string Model { get; set; }
}
```

Dziedziczenie (relacyjna baza danych)

28.08.2018 • 3 minutes to read • [Edit Online](#)

NOTE

Ogólnie rzecz biorąc jest odpowiednie dla relacyjnych baz danych konfiguracji w tej sekcji. Metody rozszerzenia, pokazane tutaj staną się dostępne po zainstalowaniu dostawcy relacyjnej bazy danych (z powodu udostępnionej `Microsoft.EntityFrameworkCore.Relational` pakietu).

Dziedziczenie w modelu platformy EF jest używane do kontrolowania, jak dziedziczenia klas jednostek jest reprezentowana w bazie danych.

NOTE

Obecnie tylko tabela wg hierarchii (TPH) ze wzorcem jest zaimplementowana w wersji EF Core. Innych typowych wzorców, takich jak tabela wg typu (TPT), a tabela konkretny-wg typu (TPC) nie są jeszcze dostępne.

Konwencje

Zgodnie z Konwencją dziedziczenia zostanie zamapowane przy użyciu wzorca Tabela wg hierarchii (TPH). TPH używa jednej tabeli do przechowywania danych dla wszystkich typów w hierarchii. Kolumna dyskryminatora jest używany do identyfikowania typu każdy wiersz reprezentuje.

EF Core tylko skonfigurować dziedziczenie, jeśli co najmniej dwa dziedziczone typy są jawnie uwzględnione w modelu (zobacz [dziedziczenia](#) Aby uzyskać więcej informacji).

Poniżej przedstawiono przykład przedstawiający Scenariusz proste dziedziczenie i dane przechowywane w tabeli relacyjnej bazy danych za pomocą wzorca TPH. *Dyskryminatora* kolumna określa, jakiego typu *blogu* są przechowywane w każdym wierszu.

```
class MyContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }
    public DbSet<RssBlog> RssBlogs { get; set; }
}

public class Blog
{
    public int BlogId { get; set; }
    public string Url { get; set; }
}

public class RssBlog : Blog
{
    public string RssUrl { get; set; }
}
```

Results				
	BlogId	Discriminator	Url	RssUrl
1	1	Blog	http://blogs.msdn.com/dotnet	NULL
2	2	RssBlog	http://blogs.msdn.com/adonet	http://blogs.msdn.com/b/adonet/atom.aspx

Anotacje danych

Nie można użyć anotacji danych, aby skonfigurować dziedziczenie.

Interfejs Fluent API

Interfejs Fluent API umożliwia skonfigurowanie nazwy i typu kolumny dyskryminatora i wartości, które są używane do identyfikowania poszczególnych typów w hierarchii.

```
class MyContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Blog>()
            .HasDiscriminator<string>("blog_type")
            .HasValue<Blog>("blog_base")
            .HasValue<RssBlog>("blog_rss");
    }
}

public class Blog
{
    public int BlogId { get; set; }
    public string Url { get; set; }
}

public class RssBlog : Blog
{
    public string RssUrl { get; set; }
}
```

Konfigurowanie właściwość rozróżniacza

W powyższych przykładach dyskryminatora jest tworzona jako [w tle właściwość](#) na podstawowej jednostce w hierarchii. Ponieważ właściwości w modelu, można skonfigurować tak jak inne właściwości. Na przykład, aby ustawić maksymalną długość, gdy jest używany domyślnie przez Konwencję dyskryminatora:

```
modelBuilder.Entity<Blog>()
    .Property("Discriminator")
    .HasMaxLength(200);
```

Rozróżniacza również mogą być mapowane z właściwością rzeczywiste CLR w jednostce. Na przykład:

```
class MyContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Blog>()
            .HasDiscriminator<string>("BlogType");
    }
}

public class Blog
{
    public int BlogId { get; set; }
    public string Url { get; set; }
    public string BlogType { get; set; }
}

public class RssBlog : Blog
{
    public string RssUrl { get; set; }
}
```

Łącząc te dwie rzeczy ze sobą istnieje możliwość zarówno mapy rozróżniacza z właściwością rzeczywistym i skonfiguruj go tak:

```
modelBuilder.Entity<Blog>(b =>
{
    b.HasDiscriminator<string>("BlogType");

    b.Property(e => e.BlogType)
        .HasMaxLength(200)
        .HasColumnName("blog_type");
});
```

Zarządzanie schematami bazy danych

28.08.2018 • 2 minutes to read • [Edit Online](#)

EF Core zawiera dwa podstawowe sposoby synchronizacja schematu modelu i bazie danych EF Core. Aby wybrać między nimi, należy zdecydować, czy modelu platformy EF Core lub schemat bazy danych jest źródło prawdziwych danych.

Aby modelu platformy EF Core jako źródło prawdziwych danych, należy użyć [migracje](#). Po wprowadzeniu dowolnych zmian do modelu platformy EF Core, to podejście przyrostowo dotyczy odpowiedniej zmiany schematu bazy danych, aby pozostaje zgodny z modelu platformy EF Core.

Użyj [odwrotnego Engineering] [2](#) Jeśli schemat bazy danych jako źródło prawdziwych danych. Takie podejście umożliwia tworzenia szkieletu typu DbContext i klasami typu jednostki przez odtwarzanie schematu bazy danych do modelu platformy EF Core.

NOTE

[Tworzenie i upuszczanie interfejsów API] [3](#) można również utworzyć schemat bazy danych z modelu platformy EF Core. Jednak są one głównie do testowania, tworzenia prototypów i innych scenariuszy, w których porzucenie bazy danych jest do zaakceptowania.

Migracje

06.10.2018 • 9 minutes to read • [Edit Online](#)

Model danych zmienia się podczas tworzenia i jest niezsynchonizowana z bazą danych. Można porzucić bazy danych i umożliwić EF, Utwórz nową, który odpowiada modelu, ale ta procedura powoduje utratę danych. Funkcja migracji w programie EF Core umożliwia przyrostowe Aktualizowanie schematu bazy danych, aby zachować synchronizację z modelem danych aplikacji przy jednoczesnym zachowaniu istniejących danych w bazie danych.

Migracja obejmuje narzędzia wiersza polecenia i interfejsów API za pomocą następujących zadań:

- [Utwórz migracji](#). Generowanie kodu, który umożliwia zaktualizowanie bazy danych, aby zsynchronizować go z zestawem zmian modelu.
- [Aktualizacji bazy danych](#). Zastosuj oczekujące migracji do zaktualizowania schematu bazy danych.
- [Dostosowywanie kodu migracji](#). Czasami wygenerowany kod, musi zostać zmodyfikowane albo uzupełniane.
- [Usuń migrację](#). Usuwanie wygenerowanego kodu.
- [Przywrót migracji](#). Cofnąć zmiany w bazie danych.
- [Generuj skrypty SQL](#). Może być konieczne skryptu zaktualizuj produkcyjnej bazy danych lub rozwiązywania problemów z kodem migracji.
- [Zastosuj migracji w środowisku uruchomieniowym](#). Po aktualizacji w czasie projektowania i uruchamianie skryptów nie są najlepsze opcje, wywołaj `Migrate()` metody.

Instalowanie narzędzi

Zainstaluj [narzędzia wiersza polecenia](#):

- Dla programu Visual Studio, firma Microsoft zaleca [narzędzia Konsola Menedżera pakietów](#).
- W innych środowiskach rozwojowych, wybierz [narzędzi interfejsu wiersza polecenia platformy .NET Core](#).

Utwórz migracji

Po [zdefiniowany model początkowej](#), nadszedł czas, aby utworzyć bazę danych. Aby dodać początkowej migracji, uruchom następujące polecenie.

```
Add-Migration InitialCreate
```

```
dotnet ef migrations add InitialCreate
```

Trzy pliki są dodawane do projektu w obszarze **migracje** katalogu:

- **0000000000000000_InitialCreate.cs**— plik główny migracji. Zawiera operacje wymagane do zastosowania migracji (w `Up()`) i można go przywrócić (w `Down()`).
- **0000000000000000_InitialCreate.Designer.cs**— plik metadanych migracji. Zawiera informacje używane przez EF.
- **MyContextModelSnapshot.cs**— migawkę bieżącego modelu. Używane do ustalenia, co zmienione podczas dodawania dalej migracji.

Sygnatura czasowa w nazwie pliku pomaga im uporządkowane chronologicznie, tak aby był widoczny postęp

zmiany.

TIP

Mogą przenieść pliki migracji i zmieniać ich nazw. Nowe migracje są tworzone jako elementy równorzędne ostatniej migracji.

Aktualizowanie bazy danych

Następnie Zastosuj migrację do bazy danych w celu utworzenia schematu.

```
Update-Database
```

```
dotnet ef database update
```

Dostosowywanie kodu migracji

Po wprowadzeniu zmian do modelu platformy EF Core, schemat bazy danych może nie być zsynchonizowany. Aby przywrócić je na bieżąco, Dodaj inną migrację. Nazwa migracji mogą być używane jak wiadomość dotycząca zatwierdzenia, w systemie kontroli wersji. Na przykład, możesz wybrać nazwę, takich jak `AddProductReviews`. Jeśli ta zmiana jest nową klasą jednostki, aby.

```
Add-Migration AddProductReviews
```

```
dotnet ef migrations add AddProductReviews
```

Po migracji szkieletu (kod generowany dla niego), przejrzyj kod dokładności i dodać, usunąć lub zmodyfikować dowolne operacje wymagane do zastosowania je poprawnie.

Na przykład migracja może zawierać następujące operacje:

```
migrationBuilder.DropColumn(  
    name: "FirstName",  
    table: "Customer");  
  
migrationBuilder.DropColumn(  
    name: "LastName",  
    table: "Customer");  
  
migrationBuilder.AddColumn<string>(  
    name: "Name",  
    table: "Customer",  
    nullable: true);
```

Chociaż te operacje zapewnić zgodność schematu bazy danych, nie przechowują nazwy istniejących klientów. Ją ulepszyć, przepisać go tak, jak pokazano poniżej.

```
migrationBuilder.AddColumn<string>(
    name: "Name",
    table: "Customer",
    nullable: true);

migrationBuilder.Sql(
@"
    UPDATE Customer
    SET Name = FirstName + ' ' + LastName;
");

migrationBuilder.DropColumn(
    name: "FirstName",
    table: "Customer");

migrationBuilder.DropColumn(
    name: "LastName",
    table: "Customer");
```

TIP

Proces tworzenia szkieletów migracji ostrzega, gdy operacja może spowodować utratę danych (np. porzucenie kolumny). Jeśli widzisz tego ostrzeżenia, należy szczególnie przejrzeć kod migracji dokładności.

Zastosuj migrację do bazy danych za pomocą odpowiedniego polecenia.

```
Update-Database
```

```
dotnet ef database update
```

Pusty migracji

Czasami przydatne jest dodać migrację bez wprowadzania żadnych zmian w modelu. W przypadku dodawania nowej migracji tworzy pliki kodu z puste klasy. Można dostosować tej migracji, aby wykonywać operacje, które bezpośrednio nie odnoszą się do modelu platformy EF Core. Jest kilka rzeczy, które można zmienić, aby zarządzać w ten sposób:

- Wyszukiwanie pełnotekstowe
- Funkcje
- Procedury składowane
- Wyzwalacze
- Widoki

Usuń migrację

Czasami Dodaj migrację i należy pamiętać, że należy wprowadzić dodatkowe zmiany w modelu platformy EF Core, zanim zostaną one zastosowane. Aby usunąć ostatniego migracji, użyj tego polecenia.

```
Remove-Migration
```

```
dotnet ef migrations remove
```

Po usunięciu migracji, można wprowadzić zmiany modelu dodatkowe i dodaj go ponownie.

Przywrót migracji

Jeśli migracji (lub kilka migracji) już zastosowane do bazy danych, ale trzeba przywrócić ją, można użyć tego samego polecenia zastosowania migracji, ale określ nazwę migracji, które chcesz przywrócić.

```
Update-Database LastGoodMigration
```

```
dotnet ef database update LastGoodMigration
```

Generuj skrypty SQL

Podczas debugowania migracji lub ich wdrożeniem w produkcyjnej bazie danych, jest przydatne, można wygenerować skryptu SQL. Skrypt można być dodatkowo zweryfikowane pod kątem dokładności i dopasowane do potrzeb produkcyjnej bazy danych. Skrypt można również w połączeniu z technologią wdrożenia. Podstawowe polecenia jest następująca.

```
Script-Migration
```

```
dotnet ef migrations script
```

Dostępnych jest kilka opcji tego polecenia.

z migracji należy ostatniej migracji, które są stosowane do bazy danych przed uruchomieniem skryptu. Jeśli migracja nie zostały zastosowane, należy określić `0` (jest to wartość domyślna).

Do migracji jest ostatni migracji, która zostanie zastosowana do bazy danych po uruchomieniu skryptu. Domyślnie do ostatniego migracji w projekcie.

Idempotentne Opcjonalnie można wygenerować skryptu. Ten skrypt tylko w przypadku migracji nie zostały jeszcze zastosowane do bazy danych. Jest to przydatne, jeśli nie dokładnie wiesz ostatniej migracji zastosowana do bazy danych zostało lub jeśli są wdrażane do wielu baz danych, które mogą znajdować się w różnych migracji.

Zastosuj migracji w czasie wykonywania

Niektóre aplikacje mogą chcieć zastosować migracji w czasie wykonywania podczas uruchamiania lub pierwszego uruchomienia. To zrobić za pomocą `Migrate()` metody.

Ta metoda tworzy się w górnej części `IMigrator` usługa, która może służyć do bardziej zaawansowanych scenariuszy. Użyj `DbContext.GetService<IMigrator>()` do niego dostęp.

```
myDbContext.Database.Migrate();
```

WARNING

- Ta metoda nie jest dla wszystkich użytkowników. Chociaż jest to doskonałe rozwiązanie dla aplikacji przy użyciu lokalnej bazy danych, większość aplikacji będzie wymagać bardziej niezawodne strategię wdrażania, takie jak Generowanie skryptów SQL.
- Nie wywołuj `EnsureCreated()` przed `Migrate()`. `EnsureCreated()` Pomija migracji w celu utworzenia schematu, co powoduje, że `Migrate()` nie powiedzie się.

Następne kroki

Aby uzyskać więcej informacji, zobacz [Entity Framework Core odnoszą się narzędzia — EF Core](#).

Migracja w środowiskach zespołu

28.08.2018 • 2 minutes to read • [Edit Online](#)

Podczas pracy z migracją w środowiskach zespołu, należy zwracać szczególną uwagę na migawki pliku modelu. Ten plik może określić, czy migracja z partnerem nie pozostawia żadnych śladów scala z Twoimi czy trzeba rozwiązać konflikt, ponownie tworząc migracji przed ich udostępnieniem.

Scalanie

Podczas migracji scalania z członkami zespołu, może wystąpić konflikty w modelu migawki pliku. W przypadku niepowiązanych obie zmiany scalanie jest proste i dwie migracje mogą współistnieć. Na przykład może wystąpić konflikt scalania w konfiguracji typu jednostki Klient, który wygląda tak:

```
<<<<< Mine
b.Property<bool>("Deactivated");
=====
b.Property<int>("LoyaltyPoints");
>>>>> Theirs
```

Ponieważ obu tych właściwości muszą istnieć w ostatnim modelu Ukończ scalanie, dodając obie te właściwości. W wielu przypadkach system kontroli wersji może automatycznie scalić takich zmian.

```
b.Property<bool>("Deactivated");
b.Property<int>("LoyaltyPoints");
```

W takich przypadkach migracji i migrację z partnerem są niezależne od siebie nawzajem. Ponieważ ktorąś z tych funkcji można najpierw zastosować, nie potrzebujesz dodatkowych zmian przed ich udostępnieniem ze swoim zespołem migracji.

Rozwiązywanie konfliktów

Czasami wystąpią true konflikt podczas scalania modelu modelu migawki. Na przykład możesz i z partnerem każdego zmieniona tej samej właściwości.

```
<<<<< Mine
b.Property<string>("Username");
=====
b.Property<string>("Alias");
>>>>> Theirs
```

Jeśli wystąpi tego rodzaju konflikt rozwiązać ten problem, ponownie utworzyć plan migracji. Wykonaj następujące kroki:

1. Przerwij, scalania i wycofania do katalogu roboczego przed scaleniem
2. Usuń plan migracji (ale zachowaj zmiany modelu)
3. Scal zmiany z partnerem w katalogu roboczym
4. Ponowne dodanie migracji

Po wykonaniu tego, dwie migracje mogą być stosowane w odpowiedniej kolejności. Ich migracji zostanie zastosowana jako pierwsza, zmiana nazwy kolumny, która ma *Alias*, po tej dacie migracji zmienia jego nazwę, aby

Username.

Migracja może być bezpiecznie udostępniane reszta zespołu.

Operacje migracji niestandardowe

06.11.2018 • 3 minutes to read • [Edit Online](#)

Interfejs API MigrationBuilder umożliwia wykonywać wiele różnych operacji podczas migracji, ale go nie jest wyczerpująca. Interfejs API jest jednak również rozszerzalny, co pozwala zdefiniować własne operacje. Istnieją dwa sposoby, aby rozszerzyć interfejs API: przy użyciu `Sql()` metody lub przez zdefiniowanie niestandardowego `MigrationOperation` obiektów.

Aby zilustrować, Przyjrzyjmy się wykonania operacji, która tworzy użytkownika bazy danych za pomocą danej metody. W naszym migracji chcemy włączenia zapisywania następujący kod:

```
migrationBuilder.CreateUser("SQLUser1", "Password");
```

Za pomocą MigrationBuilder.Sql()

Najłatwiejszym sposobem realizowania operacji niestandardowej jest zdefiniowanie metodę rozszerzającą wywołującą `MigrationBuilder.Sql()`. Oto przykład, który generuje odpowiednie języka Transact-SQL.

```
static MigrationBuilder CreateUser(
    this MigrationBuilder migrationBuilder,
    string name,
    string password)
=> migrationBuilder.Sql($"CREATE USER {name} WITH PASSWORD '{password}';");
```

Jeśli migracji potrzebne do obsługi wielu dostawców bazy danych, można użyć `MigrationBuilder.ActiveProvider` właściwości. Oto przykład obsługi zarówno programu Microsoft SQL Server, jak i bazy danych PostgreSQL.

```
static MigrationBuilder CreateUser(
    this MigrationBuilder migrationBuilder,
    string name,
    string password)
{
    switch (migrationBuilder.ActiveProvider)
    {
        case "Npgsql.EntityFrameworkCore.PostgreSQL":
            return migrationBuilder
                .Sql($"CREATE USER {name} WITH PASSWORD '{password}';");

        case "Microsoft.EntityFrameworkCore.SqlServer":
            return migrationBuilder
                .Sql($"CREATE USER {name} WITH PASSWORD = '{password}';");
    }

    return migrationBuilder;
}
```

To podejście tylko wtedy, gdy wiesz, co dostawcy której zostaną zastosowane niestandardowych operacji.

Za pomocą MigrationOperation

Aby oddzielić niestandardowych operacji z bazy danych SQL, można zdefiniować własne `MigrationOperation` go reprezentuje. Operacja są następnie przekazywane do dostawcy, dzięki czemu można określić, odpowiednie SQL w

celu generowania.

```
class CreateUserOperation : MigrationOperation
{
    public string Name { get; set; }
    public string Password { get; set; }
}
```

W przypadku tej metody, metoda rozszerzenia musi jedynie jedną z tych operacji, aby dodać

```
MigrationBuilder.Operations .
```

```
static MigrationBuilder CreateUser(
    this MigrationBuilder migrationBuilder,
    string name,
    string password)
{
    migrationBuilder.Operations.Add(
        new CreateUserOperation
        {
            Name = name,
            Password = password
        });

    return migrationBuilder;
}
```

Takie podejście wymaga każdego dostawcy wiedzieć, jak wygenerować kodu SQL do wykonania tej operacji w ich **IMigrationsSqlGenerator** usługi. Oto przykład zastępowanie generator programu SQL Server do obsługi nowej operacji.

```

class MyMigrationsSqlGenerator : SqlServerMigrationsSqlGenerator
{
    public MyMigrationsSqlGenerator(
        MigrationsSqlGeneratorDependencies dependencies,
        IMigrationsAnnotationProvider migrationsAnnotations)
        : base(dependencies, migrationsAnnotations)
    {
    }

    protected override void Generate(
        MigrationOperation operation,
        IModel model,
        MigrationCommandListBuilder builder)
    {
        if (operation is CreateUserOperation createUserOperation)
        {
            Generate(createUserOperation, builder);
        }
        else
        {
            base.Generate(operation, model, builder);
        }
    }

    private void Generate(
        CreateUserOperation operation,
        MigrationCommandListBuilder builder)
    {
        var sqlHelper = Dependencies.SqlGenerationHelper;
        var stringMapping = Dependencies.TypeMappingSource.FindMapping(typeof(string));

        builder
            .Append("CREATE USER ")
            .Append(sqlHelper.DelimitIdentifier(operation.Name))
            .Append(" WITH PASSWORD = ")
            .Append(stringMapping.GenerateSqlLiteral(operation.Password))
            .AppendLine(sqlHelper.StatementTerminator)
            .EndCommand();
    }
}

```

Zaktualizowano zastąpić domyślnej migracji sql generator usługi.

```

protected override void OnConfiguring(DbContextOptionsBuilder options)
=> options
    .UseSqlServer(connectionString)
    .ReplaceService<IMigrationsSqlGenerator, MyMigrationsSqlGenerator>();

```

Korzystanie z osobnego projektu

29.09.2018 • 2 minutes to read • [Edit Online](#)

Możesz chcieć przechowywać migracji w innym zestawie niż jeden zawierający swoje `DbContext`. Do uaktualnienia wersji do wersji, można użyć tej strategii do obsługi wielu zestawów migracji, na przykład, jeden do tworzenia aplikacji i inny.

Aby to zrobić...

1. Utwórz nową bibliotekę klas.
2. Dodaj odwołanie do zestawu DbContext.
3. Przenieś migracje i pliki migawki modelu do biblioteki klas.

TIP

Jeśli nie masz żadnych istniejących migracji, wygeneruje w do projektu zawierającego kontekstu DbContext, a następnie przenieś go. Jest to ważne, ponieważ jeśli zestaw migracji nie zawiera istniejącego migracji, polecenia Add-migracji nie można odnaleźć kontekstu DbContext.

4. Skonfiguruj zestaw migracji:

```
options.UseSqlServer(  
    connectionString,  
    x => x.MigrationsAssembly("MyApp.Migrations"));
```

5. Dodaj odwołanie do zestawu migracji z zestawu startowego.

- Jeśli spowoduje to utworzenie zależności cyklicznej, zaktualizuj ścieżkę wyjściową biblioteki klas:

```
<PropertyGroup>  
  <OutputPath>..\\MyStartupProject\\bin\\$(Configuration)\\</OutputPath>  
</PropertyGroup>
```

Jeśli tak, nie wszystko prawidłowo, należy dodać nowe migracji do projektu.

```
Add-Migration NewMigration -Project MyApp.Migrations
```

```
dotnet ef migrations add NewMigration --project MyApp.Migrations
```

Migracje z wielu dostawców

13.09.2018 • 2 minutes to read • [Edit Online](#)

[EF Core Tools] 1 tylko tworzenia szkieletu migracji active dostawcy. Czasami jednak warto użyć więcej niż jednego dostawcę (na przykład Microsoft SQL Server i bazy danych SQLite) z Twojego DbContext. Istnieją dwa sposoby określają je za pomocą migracji. Dwa zestawy można zachować na potrzeby migracji — jeden dla każdego dostawcy — lub scalania ich do jednego zestawu, który działa na obu.

Dwa zestawy migracji

Pierwszym sposobem służy do generowania dwie migracje dla każdej zmiany modelu.

Jednym ze sposobów, aby zrobić to polega na umieszczeniu każdego zestawu migracji [w osobnym zestawie] 2 i ręcznie przełączać między dodawaniem dwie migracje active dostawcy (i zestawu migracji).

Jest innym podejściem, która ułatwia pracę z narzędziami do tworzenia nowego typu, który dziedziczy z typu DbContext i zastępuje active dostawcy. Ten typ jest używany w projekt czas podczas dodawania lub zastosowanie migracji.

```
class MySqliteDbContext : MyDbContext
{
    protected override void OnConfiguring(DbContextOptionsBuilder options)
        => options.UseSqlite("Data Source=my.db");
}
```

NOTE

Ponieważ każdy zestaw migracji wykorzystuje swoje własne typy DbContext, to podejście nie wymaga użycia zestawu oddzielne migracje.

Podczas dodawania nowej migracji, określić typy kontekstu.

```
Add-Migration InitialCreate -Context MyDbContext -OutputDir Migrations\SqlServerMigrations
Add-Migration InitialCreate -Context MySqliteDbContext -OutputDir Migrations\SqliteMigrations
```

```
dotnet ef migrations add InitialCreate --context MyDbContext --output-dir Migrations/SqlServerMigrations
dotnet ef migrations add InitialCreate --context MySqliteDbContext --output-dir Migrations/SqliteMigrations
```

TIP

Nie potrzebujesz określić katalog wyjściowy dla następnej migracji, ponieważ są one tworzone jako elementy równorzędne do ostatni z nich.

Zestaw jedną migrację

Jeśli nie masz dwa zestawy na potrzeby migracji, należy ręcznie połączyć je w jeden zestaw, który można zastosować do obu dostawców.

Adnotacje mogą współistnieć, ponieważ dostawca ignoruje wszelkie adnotacji, które go nie rozumie. Na przykład

kolumna klucza podstawowego, który działa zarówno z programu Microsoft SQL Server i bazy danych SQLite może wyglądać następująco.

```
Id = table.Column<int>(nullable: false)
    .Annotation("SqlServer:ValueGenerationStrategy",
        SqlServerValueGenerationStrategy.IdentityColumn)
    .Annotation("Sqlite:Autoincrement", true),
```

Jeśli operacje mogą być stosowane tylko dla jednego dostawcy (lub są one między dostawcami), użyj `ActiveProvider` właściwość, aby sprawdzić, który dostawca jest aktywny.

```
if (migrationBuilder.ActiveProvider == "Microsoft.EntityFrameworkCore.SqlServer")
{
    migrationBuilder.CreateSequence(
        name: "EntityFrameworkHiLoSequence");
}
```

Tabela historii migracje niestandardowe

13.09.2018 • 2 minutes to read • [Edit Online](#)

Domyślnie śledzi informacje o programu EF Core migracji, które zostały zastosowane do bazy danych, rejestrując je w tabeli o nazwie `__EFMigrationsHistory`. Z różnych powodów można dostosować do własnych potrzeb.

IMPORTANT

W przypadku dostosowania tabeli historii migracje po zastosowaniem migracje, jesteś odpowiedzialny za aktualizowanie istniejącej tabeli w bazie danych.

Nazwa schematu i tabeli

Można zmienić schematu i tabeli przy użyciu nazwy `MigrationsHistoryTable()` method in Class metoda `OnConfiguring()` (lub `ConfigureServices()`) programu ASP.NET Core). Oto przykład przy użyciu dostawcy programu SQL Server programu EF Core.

```
protected override void OnConfiguring(DbContextOptionsBuilder options)
    => options.UseSqlServer(
        connectionString,
        x => x.MigrationsHistoryTable("__MyMigrationsHistory", "mySchema"));
```

Inne zmiany

Aby skonfigurować dodatkowe aspekty tabeli, należy zastąpić i Zastąp właściwe dla dostawcy `IHistoryRepository` usługi. Oto przykład zmiany `MigrationId` nazwa kolumny, która ma *identyfikator* w programie SQL Server.

```
protected override void OnConfiguring(DbContextOptionsBuilder options)
    => options
        .UseSqlServer(connectionString)
        .ReplaceService<IHistoryRepository, MyHistoryRepository>();
```

WARNING

`SqlServerHistoryRepository` znajduje się wewnątrz wewnętrznego obszaru nazw i mogą ulec zmianie w przyszłych wersjach.

```
class MyHistoryRepository : SqlServerHistoryRepository
{
    public MyHistoryRepository(HistoryRepositoryDependencies dependencies)
        : base(dependencies)
    {
    }

    protected override void ConfigureTable(EntityTypeBuilder<HistoryRow> history)
    {
        base.ConfigureTable(history);

        history.Property(h => h.MigrationId).HasColumnName("Id");
    }
}
```

Tworzenie i upuszczanie interfejsów API

16.11.2018 • 2 minutes to read • [Edit Online](#)

Metody `EnsureCreated` i `EnsureDeleted` udostępniają uproszczone zamiast [migracje](#) zarządzania schemat bazy danych. Te metody są przydatne w scenariuszach danych jest przejściowy i można było porzucić po zmianie schematu. Na przykład podczas tworzenia prototypów w testach lub pamięciach podręcznych.

Niektórzy dostawcy (zwłaszcza tymi nierelacyjnymi) nie obsługuje migracji. W przypadku tych dostawców `EnsureCreated` często jest najprostszym sposobem zainicjować schemat bazy danych.

WARNING

`EnsureCreated` i migracje nie działają dobrze ze sobą. Jeśli używasz migracji, nie należy używać `EnsureCreated` zainicjować schematu.

Przechodzenie ze `EnsureCreated` do migracji nie jest bezproblemowe. Najprostszym sposobem, aby to zrobił jest porzucenia bazy danych i ponownego utworzenia go za pomocą migracji. Jeśli przewidujesz używanie migracji w przyszłości, najlepiej zamiast `EnsureCreated`, po prostu zacznij z migracji.

EnsureDeleted

Metoda `EnsureDeleted` spowoduje porzucenie bazy danych, jeśli taki istnieje. Jeśli nie masz odpowiednich uprawnień, jest zgłoszany wyjątek.

```
// Drop the database if it exists  
dbContext.Database.EnsureDeleted();
```

EnsureCreated

`EnsureCreated` spowoduje utworzenie bazy danych, jeśli nie istnieje i zainicjować schemat bazy danych. Jeśli istnieje żadnych tabel (w tym tabel dla innej klasy `DbContext`) schematu nie można zainicjować.

```
// Create the database if it doesn't exist  
dbContext.Database.EnsureCreated();
```

TIP

Ponadto dostępnych asynchronicznych wersji tych metod.

Skrypt SQL

Aby uzyskać SQL używane przez `EnsureCreated`, można użyć metody `GenerateCreateScript`.

```
var sql = dbContext.Database.GenerateCreateScript();
```

Wiele klas typu `DbContext`

EnsureCreated działa tylko w przypadku, gdy żadna tabela znajdują się w bazie danych. Jeśli to konieczne, można napisać własne sprawdzenie, czy schemat musi zostać zainicjowany i usługa bazowego IRelationalDatabaseCreator zainicjować schematu.

```
// TODO: Check whether the schema needs to be initialized

// Initialize the schema for this DbContext
var databaseCreator = dbContext.GetService<IRelationalDatabaseCreator>();
databaseCreator.CreateTables();
```

Odtwarzanie

16.11.2018 • 10 minutes to read • [Edit Online](#)

Odtwarzanie jest proces tworzenia szkieletów klasy typów jednostek i klasy DbContext, na podstawie schematu bazy danych. Mogą być wykonywane przy użyciu `Scaffold-DbContext` polecenia narzędzia konsoli Menedżera pakietów (PMC) EF Core lub `dotnet ef dbcontext scaffold` polecenia narzędzia .NET interfejsu wiersza poleceń (CLI).

Instalowanie programu

Przed odtwarzaniem, należy zainstallować jedną [narzędzia PMC](#) (tylko Visual Studio) lub [narzędzi interfejsu wiersza poleceń](#). Zobacz linki, aby uzyskać szczegółowe informacje.

Należy także zainstallować odpowiednią [dostawcy bazy danych](#) dla schematu bazy danych ma zostać odtworzona.

Parametry połączenia

Pierwszy argument polecenia jest ciąg połączenia z bazą danych. Narzędzia użye te parametry połączenia można odczytać schematu bazy danych.

Jak oferty i wprowadzić ciąg połączenia zależy od Shell, które używają można wykonać polecenia. Zajrzyj do dokumentacji powłoki, aby uzyskać szczegółowe informacje. Na przykład programu PowerShell wymaga jako znak ucieczki `$` znaków, ale nie `\`.

```
Scaffold-DbContext 'Data Source=(localdb)\MSSQLLocalDB;Initial Catalog=Chinook'  
Microsoft.EntityFrameworkCore.SqlServer
```

```
dotnet ef dbcontext scaffold "Data Source=(localdb)\MSSQLLocalDB;Initial Catalog=Chinook"  
Microsoft.EntityFrameworkCore.SqlServer
```

Konfiguracja i wpisami tajnymi użytkowników

Jeśli masz projekt platformy ASP.NET Core, możesz użyć `Name=<connection-string>` składni można odczytać parametrów połączenia z konfiguracji.

Ta funkcja działa dobrze z [narzędzie Menedżer klucz tajny](#) być oddzielone od kodu hasła bazy danych.

```
dotnet user-secrets set ConnectionStrings.Chinook "Data Source=(localdb)\MSSQLLocalDB;Initial  
Catalog=Chinook"  
dotnet ef dbcontext scaffold Name=Chinook Microsoft.EntityFrameworkCore.SqlServer
```

Nazwa dostawcy

Drugi argument jest nazwą dostawcy. Nazwa dostawcy jest zwykle taka sama jak nazwa pakietu NuGet dostawcy.

Określanie tabel

Wszystkie tabele w schemacie bazy danych są odtwarzane do typów jednostek, domyślnie. Można ograniczyć, które tabele są odtwarzane, określając schematy i tabele.

`-Schemas` Parametru w konsoli zarządzania Pakietami i `--schema` opcji interfejsu wiersza polecenia może służyć do uwzględnienia w każdej tabeli w schemacie.

`-Tables` (PMC) i `--table` (CLI) może służyć do uwzględnienia określonych tabel.

Aby uwzględnić wiele tabel w konsoli zarządzania Pakietami, skorzystaj z tablicy.

```
Scaffold-DbContext ... -Tables Artist, Album
```

Aby uwzględnić wiele tabel w interfejsu wiersza polecenia, należy określić opcję wiele razy.

```
dotnet ef dbcontext scaffold ... --table Artist --table Album
```

Zachowywanie nazwy

Nazwy tabel i kolumn są stałe, aby lepiej dopasować je do konwencji nazewnictwa platformy .NET, właściwości i typy domyślnie. Określanie `-UseDatabaseNames` przełącznika w konsoli zarządzania Pakietami lub `--use-database-names` opcji interfejsu wiersza polecenia spowoduje wyłączenie to zachowania, zachowując możliwe oryginalne nazwy bazy danych. Nieprawidłowe identyfikatory .NET nadal zostanie rozwiązany i syntetyzowany nazwy, takie jak właściwości nawigacji nadal będą zgodne z konwencjami nazewnictwa platformy .NET.

Interfejs API Fluent lub adnotacji danych

Typy jednostek są konfigurowane przy użyciu interfejsu API Fluent domyślnie. Określ `-DataAnnotations` (PMC) lub `--data-annotations` (CLI), aby zamiast tego użyć adnotacji danych, gdy jest to możliwe.

Na przykład za pomocą interfejsu API Fluent będzie tworzenia szkieletu tej.

```
entity.Property(e => e.Title)
    .IsRequired()
    .HasMaxLength(160);
```

Podczas korzystania z adnotacji danych będzie tworzenia szkieletu to.

```
[Required]
[StringLength(160)]
public string Title { get; set; }
```

Nazwa typu DbContext

Nazwa klasy DbContext szkieletu będzie nazwa bazy danych z sufiksem *kontekstu* domyślnie. Aby określić inne konto, użyj `-Context` w konsoli zarządzania Pakietami i `--context` w interfejsie wiersza polecenia.

Katalogi i przestrzenie nazw

Klas jednostek i klasy DbContext są szkieletu do katalogu głównego projektu i użyj projektu domyślny obszar nazw. Można określić katalog, w przypadku, gdy klasy są szkieletu za pomocą `-OutputDir` (PMC) lub `--output-dir` (CLI). Przestrzeń nazw będzie głównej przestrzeni nazw, a także nazw podkatalogów w katalogu głównym projektu.

Można również użyć `-ContextDir` (PMC) i `--context-dir` (CLI) do tworzenia szkieletu klasy DbContext w

oddzielnym katalogu z klasami typu jednostki.

```
Scaffold-DbContext ... -ContextDir Data -OutputDir Models
```

```
dotnet ef dbcontext scaffold ... --context-dir Data --output-dir Models
```

Jak to działa

Odtwarzanie rozpoczyna się od odczytu schematu bazy danych. Odczytuje informacje o tabeli, kolumny, ograniczenia i indeksy.

Następnie używa informacji o schemacie, aby utworzyć model programu EF Core. Tabele są używane do tworzenia typów jednostek. Kolumny są używane do tworzenia właściwości; i kluczy obcych są używane do tworzenia relacji.

Na koniec model jest używany do generowania kodu. Odpowiednie jednostki typu klasy, interfejsu API Fluent i danych adnotacje są szkieletem. Aby ponownie utworzyć ten sam model aplikacji.

Co nie działa

Nie wszystkie informacje o modelu może być reprezentowany za pomocą schematu bazy danych. Na przykład informacje o **hierarchii dziedziczenia, należące do typów, i tabeli podział** nie są obecne w schemacie bazy danych. W związku z tym te konstrukcje będzie nigdy nie być odtwarzane.

Ponadto **niektóre typy kolumn** nie mogą być obsługiwane przez dostawcę programu EF Core. Kolumny te nie zostaną uwzględnione w modelu.

EF Core wymaga klucza każdego typu jednostki. Tabele, jednak nie są wymagane do określenia klucza podstawowego. **Tabele bez klucza podstawowego** obecnie nie są odtwarzane.

Można zdefiniować **tokeny współbieżności** w modelu platformy EF Core uniemożliwić dwa użytkownikom aktualizowanie tej samej jednostki w tym samym czasie. Niektóre bazy danych ma specjalny typ do reprezentowania kolumna (na przykład rowversion w programie SQL Server) tego typu w takiej sytuacji firma Microsoft może wycofać inżynier te informacje; jednak inne tokeny współbieżności będzie nie być odtwarzane.

Dostosowywanie modelu

Kod wygenerowany przez platformę EF Core to Twój kod. Możesz go zmienić. Zostaną tylko wygenerowane, jeśli możesz odtworzyć tego samego modelu ponownie. Reprezentuje utworzony szkielet kodu *jeden* modelu, który może służyć do uzyskania dostępu do bazy danych, ale na pewno nie jest *tylko* modelu, który może służyć.

Dostosuj klasy typów jednostek i klasy DbContext odpowiednio do potrzeb. Na przykład można zmienić nazwy właściwości i typy, wprowadzają hierarchii dziedziczenia lub dzielenie tabeli do wielu jednostek. Indeksy unikatowe, nieużywane sekwencji i właściwości nawigacji, opcjonalne właściwości skalarne i nazwy ograniczenia mogą również usuwać z modelu.

Można również dodać dodatkowe konstruktory, metod, właściwości itd. w oddzielnym pliku, przy użyciu innej klasy częściowej. To podejście sprawdza się nawet wtedy, gdy zamierzasz odtwarzać modelu ponownie.

Aktualizowanie modelu

Po wprowadzeniu zmian w bazie danych, może być konieczne zaktualizowanie modelu platformy EF Core w celu odzwierciedlenia tych zmian. W przypadku prostych zmian w bazie danych może być łatwiej tylko ręcznie wprowadź zmiany w modelu platformy EF Core. Na przykład zmiana nazwy tabeli lub kolumny, usunięcie

kolumny lub aktualizowanie typ kolumny są proste zmiany w kodzie.

Najbardziej znaczące zmiany nie są jednak jako łatwe upewnić ręcznie. Jeden typowy przepływ pracy jest, aby odwrócić inżynier modelu z bazy danych ponownie, używając `-Force` (PMC) lub `--force` (CLI) w celu zastąpienia istniejącego modelu jedynie zaktualizowane.

Kolejną funkcją najczęściej żądanych jest możliwość aktualizacji modelu z bazy danych przy jednoczesnym zachowaniu dostosowywania, takich jak zmiana nazwy, typu hierarchie itp. Użyj problem [#831](#) do śledzenia postępu dla tej funkcji.

WARNING

Jeśli odtworzeniu modelu z bazy danych ponownie, wszelkie zmiany wprowadzone w plikach zostaną utracone.

Wykonanie zapytania o dane

28.08.2018 • 2 minutes to read • [Edit Online](#)

Entity Framework Core używa Language Integrated Query (LINQ) do zapytania o dane z bazy danych. LINQ umożliwia przy użyciu języka C# (lub wybranym języku .NET) do pisania zapytań silnie typizowaną oparte na klasach pochodnych kontekstu i jednostki. Reprezentacja zapytania LINQ są przekazywane do dostawcy bazy danych, aby być przetłumaczony na język zapytań specyficznych dla bazy danych (na przykład SQL dla relacyjnej bazy danych). Aby uzyskać bardziej szczegółowe informacje dotyczące sposobu przetwarzania zapytania, zobacz [jak działa zapytanie](#).

Podstawowe zapytania

28.08.2018 • 2 minutes to read • [Edit Online](#)

Dowiedz się, jak można załadować jednostek z bazy danych przy użyciu Language Integrated Query (LINQ).

TIP

Przykład użyty w tym artykule można zobaczyć w witrynie GitHub.

101 przykładów LINQ

Ta strona zawiera kilka przykładów, aby osiągnąć typowych zadań przy użyciu platformy Entity Framework Core. Aby uzyskać obszerny zestaw przykładów pokazujący, co jest możliwe za pomocą LINQ, zobacz [101 przykładów LINQ](#).

Podczas ładowania wszystkie dane

```
using (var context = new BloggingContext())
{
    var blogs = context.Blogs.ToList();
}
```

Trwa ładowanie pojedynczej jednostki

```
using (var context = new BloggingContext())
{
    var blog = context.Blogs
        .Single(b => b.BlogId == 1);
}
```

Filtrowanie

```
using (var context = new BloggingContext())
{
    var blogs = context.Blogs
        .Where(b => b.Url.Contains("dotnet"))
        .ToList();
}
```

Ładowanie powiązanych danych

28.08.2018 • 13 minutes to read • [Edit Online](#)

Entity Framework Core umożliwia ładowanie powiązanych jednostek przy użyciu właściwości nawigacji w modelu. Istnieją trzy powszechnie używane wzorce Obiektywnie używana do ładowania powiązane dane.

- **Wczesne ładowanie** oznacza, że powiązane dane są ładowane z bazy danych w ramach początkowego zapytania.
- **Jawne ładowanie** oznacza, że powiązanych danych jest jawnie załadowane z bazy danych w późniejszym czasie.
- **Powolne ładowanie** oznacza, że powiązane przezroczyste załadowaniu danych z bazy danych podczas uzyskiwania dostępu do właściwości nawigacji.

TIP

[Przykład](#) użyty w tym artykule można zobaczyć w witrynie GitHub.

Wczesne ładowanie

Możesz użyć `Include` metodę, aby określić powiązanych danych, które mają zostać uwzględnione w wynikach kwerendy. W poniższym przykładzie, blogów, które są zwracane w wynikach będzie mieć ich `Posts` właściwość wypełniony pokrewnych wpisów.

```
using (var context = new BloggingContext())
{
    var blogs = context.Blogs
        .Include(blog => blog.Posts)
        .ToList();
}
```

TIP

Entity Framework Core będą automatycznie konfigurowania właściwości nawigacji do innych jednostek, które wcześniej zostały załadowane do wystąpienia kontekstu. Dlatego nawet wtedy, gdy jawnie nie zawiera danych dla właściwości nawigacji, nadal można wypełnić właściwość, jeśli niektóre lub wszystkie powiązane jednostki zostały wcześniej załadowane.

Może zawierać dane związane z wieloma relacjami w ramach pojedynczego zapytania.

```
using (var context = new BloggingContext())
{
    var blogs = context.Blogs
        .Include(blog => blog.Posts)
        .Include(blog => blog.Owner)
        .ToList();
}
```

W tym wiele poziomów

Możesz przejść do szczegółów w relacji do uwzględnienia na wielu poziomach powiązanych danych przy użyciu `ThenInclude` metody. Poniższy przykład ładuje wszystkie blogi, ich pokrewnych wpisów i autor każdego wpisu.

```
using (var context = new BloggingContext())
{
    var blogs = context.Blogs
        .Include(blog => blog.Posts)
            .ThenInclude(post => post.Author)
        .ToList();
}
```

NOTE

Bieżące wersje programu Visual Studio oferuje opcje uzupełniania niepoprawny kod i może spowodować, że poprawne wyrażenia do oflagowania z błędami składni, korzystając z `ThenInclude` metody właściwości nawigacji kolekcji. Jest to objaw błędów funkcji IntelliSense w <https://github.com/dotnet/roslyn/issues/8237>. Jest bezpiecznie zignorować te błędy składniowe fałszywe, tak długo, jak kod jest poprawny i mogą być pomyślnie skompilowane.

Można połączyć w łańcuch wielu wywołań `ThenInclude` aby kontynuować, włączając dalsze poziomy powiązane dane.

```
using (var context = new BloggingContext())
{
    var blogs = context.Blogs
        .Include(blog => blog.Posts)
            .ThenInclude(post => post.Author)
                .ThenInclude(author => author.Photo)
        .ToList();
}
```

Możesz połączyć wszystkie te które mają zostać objęte powiązane dane z wielu poziomów i wielu katalogów głównych tego samego zapytania.

```
using (var context = new BloggingContext())
{
    var blogs = context.Blogs
        .Include(blog => blog.Posts)
            .ThenInclude(post => post.Author)
                .ThenInclude(author => author.Photo)
        .Include(blog => blog.Owner)
            .ThenInclude(owner => owner.Photo)
        .ToList();
}
```

Możesz uwzględnić wiele powiązanych jednostek dla jednej jednostki, które jest uwzględniane. Na przykład podczas wykonywania zapytań dotyczących `Blog`s, obejmują `Posts`, a następnie oba `Author` i `Tags` z `Posts`. Aby to zrobić, należy określić każdy obejmować ścieżkę, począwszy od głównego. Na przykład `Blog -> Posts -> Author` i `Blog -> Posts -> Tags`. Oznacza to, zostanie nadmiarowe sprzężenia, w większości przypadków będzie konsolidować EF sprzężeń podczas generowania SQL.

```
using (var context = new BloggingContext())
{
    var blogs = context.Blogs
        .Include(blog => blog.Posts)
            .ThenInclude(post => post.Author)
        .Include(blog => blog.Posts)
            .ThenInclude(post => post.Tags)
        .ToList();
}
```

Obejmuję na typy pochodne

Może zawierać dane związane z tego zdefiniowany tylko w typie pochodnym przy użyciu `Include` i `ThenInclude`.

Biorąc pod uwagę następujący wzór:

```
public class SchoolContext : DbContext
{
    public DbSet<Person> People { get; set; }
    public DbSet<School> Schools { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<School>().HasMany(s => s.Students).WithOne(s => s.School);
    }
}

public class Person
{
    public int Id { get; set; }
    public string Name { get; set; }
}

public class Student : Person
{
    public School School { get; set; }
}

public class School
{
    public int Id { get; set; }
    public string Name { get; set; }

    public List<Student> Students { get; set; }
}
```

Zawartość `School` nawigacji wszystkie osoby, których uczniowie mogą być eagerly ładowane, przy użyciu wielu wzorców:

- Używanie `cast`

```
context.People.Include(person => ((Student)person).School).ToList()
```

- za pomocą `as` — operator

```
context.People.Include(person => (person as Student).School).ToList()
```

- za pomocą przeciążenia `Include` przyjmującą parametr typu `string`

```
context.People.Include("Student").ToList()
```

Ignorowane obejmuje

Jeśli zmienisz zapytanie tak, aby nie zwraca wystąpienia typu jednostki, który rozpoczął się zapytania, operatory dołączania są ignorowane.

W poniższym przykładzie operatory dołączania są oparte na `Blog`, ale następnie `Select` operator jest używany do zmiany zwracanego typu anonimowego przez zapytanie. W tym przypadku operatory `include` nie mają wpływu.

```
using (var context = new BloggingContext())
{
    var blogs = context.Blogs
        .Include(blog => blog.Posts)
        .Select(blog => new
    {
        Id = blog.BlogId,
        Url = blog.Url
    })
    .ToList();
}
```

Domyślnie EF Core zarejestruje ostrzeżenie obejmującą gdy operatory są ignorowane. Zobacz [rejestrowania](#) więcej informacji na temat wyświetlania danych wyjściowych rejestrowania. Można zmienić zachowanie, gdy include operator jest ignorowana, throw lub nic nie rób. Odbywa się podczas konfigurowania opcji kontekstu — zwykle w `DbContext.OnConfiguring`, lub `Startup.cs` korzystania z platformy ASP.NET Core.

```
protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
{
    optionsBuilder
        .UseSqlServer(@"Server=
(localdb)\mssqllocaldb;Database=EFQuerying;Trusted_Connection=True;ConnectRetryCount=0")
        .ConfigureWarnings(warnings => warnings.Throw(CoreEventId.IncludeIgnoredWarning));
}
```

jawne ładowanie

NOTE

Ta funkcja została wprowadzona w programie EF Core 1.1.

Można jawnie załadować właściwości nawigacji za pomocą `DbContext.Entry(...)` interfejsu API.

```
using (var context = new BloggingContext())
{
    var blog = context.Blogs
        .Single(b => b.BlogId == 1);

    context.Entry(blog)
        .Collection(b => b.Posts)
        .Load();

    context.Entry(blog)
        .Reference(b => b.Owner)
        .Load();
}
```

Właściwość nawigacji można także jawnie załadować, wykonując oddzielne zapytania, które zwraca powiązanych jednostek. Jeśli jest włączone śledzenie zmian, a następnie podczas ładowania jednostki, EF Core będą automatycznie ustawić właściwości nawigacji między jednostkami nowo załadowane do odwoływania się do żadnych jednostek już załadowany i ustawić właściwości nawigacji jednostki już załadowane do odwoływania się do jednostka nowo załadowane.

Tworzenie zapytań powiązanych jednostek

Można również uzyskać kwerenda LINQ, która reprezentuje zawartość właściwości nawigacji.

Dzięki temu można korzystać z możliwości, takie jak uruchomienie aggregate-operator za pośrednictwem

powiązanych jednostek bez ładowania ich do pamięci.

```
using (var context = new BloggingContext())
{
    var blog = context.Blogs
        .Single(b => b.BlogId == 1);

    var postCount = context.Entry(blog)
        .Collection(b => b.Posts)
        .Query()
        .Count();
}
```

Można również filtrować, które powiązane jednostki są ładowane do pamięci.

```
using (var context = new BloggingContext())
{
    var blog = context.Blogs
        .Single(b => b.BlogId == 1);

    var goodPosts = context.Entry(blog)
        .Collection(b => b.Posts)
        .Query()
        .Where(p => p.Rating > 3)
        .ToList();
}
```

Ładowanie z opóźnieniem

NOTE

Ta funkcja została wprowadzona w programie EF Core 2.1.

Najprostszym sposobem użycia ładowanych z opóźnieniem jest po zainstalowaniu [Microsoft.EntityFrameworkCore.Proxies](#) pakietu i włączanie go z wywołaniem `UseLazyLoadingProxies`. Na przykład:

```
protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    => optionsBuilder
        .UseLazyLoadingProxies()
        .UseSqlServer(myConnectionString);
```

Lub w przypadku używania AddDbContext:

```
.AddDbContext<BloggingContext>(
    b => b.UseLazyLoadingProxies()
        .UseSqlServer(myConnectionString));
```

EF Core będzie z opóźnieniem ładowania dla dowolnej właściwości nawigacji, która może zostać przesłonięta — oznacza to, Włącz musi być `virtual` i klasy, które mogą być dziedziczone z. Na przykład w następujących elementach `Post.Blog` i `Blog.Posts` właściwości nawigacji będą ładowane z opóźnieniem.

```
public class Blog
{
    public int Id { get; set; }
    public string Name { get; set; }

    public virtual ICollection<Post> Posts { get; set; }
}

public class Post
{
    public int Id { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }

    public virtual Blog Blog { get; set; }
}
```

Powolne ładowanie bez serwerów proxy

Serwery proxy ładowanych z opóźnieniem pracy przez iniekcję `ILazyLoader` usługi do jednostki, zgodnie z opisem w [Konstruktorzy typów jednostek](#). Na przykład:

```

public class Blog
{
    private ICollection<Post> _posts;

    public Blog()
    {
    }

    private Blog(ILazyLoader lazyLoader)
    {
        LazyLoader = lazyLoader;
    }

    private ILazyLoader LazyLoader { get; set; }

    public int Id { get; set; }
    public string Name { get; set; }

    public ICollection<Post> Posts
    {
        get => LazyLoader.Load(this, ref _posts);
        set => _posts = value;
    }
}

public class Post
{
    private Blog _blog;

    public Post()
    {
    }

    private Post(ILazyLoader lazyLoader)
    {
        LazyLoader = lazyLoader;
    }

    private ILazyLoader LazyLoader { get; set; }

    public int Id { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }

    public Blog Blog
    {
        get => LazyLoader.Load(this, ref _blog);
        set => _blog = value;
    }
}

```

To nie wymaga typów jednostek, dziedziczenie lub właściwości nawigacji, aby być wirtualne oraz umożliwia wystąpień jednostek utworzonych za pomocą `new` z opóźnieniem obciążenia raz przyłączonych do kontekstu. Jednak wymaga to dodania odwołania do `ILazyLoader` usługa, która jest zdefiniowana w [Microsoft.EntityFrameworkCore.Abstractions](#) pakietu. Ten pakiet zawiera minimalny zestaw typów, dzięki czemu istnieje niewielki wpływ w zależności od jego. Jednak aby całkowicie uniknąć zależności od tego, wszystkie pakiety programu EF Core w typów jednostek, istnieje możliwość wstawić `ILazyLoader.Load` metody jako pełnomocnik. Na przykład:

```
public class Blog
{
    private ICollection<Post> _posts;

    public Blog()
    {
    }

    private Blog(Action<object, string> lazyLoader)
    {
        LazyLoader = lazyLoader;
    }

    private Action<object, string> LazyLoader { get; set; }

    public int Id { get; set; }
    public string Name { get; set; }

    public ICollection<Post> Posts
    {
        get => LazyLoader.Load(this, ref _posts);
        set => _posts = value;
    }
}

public class Post
{
    private Blog _blog;

    public Post()
    {
    }

    private Post(Action<object, string> lazyLoader)
    {
        LazyLoader = lazyLoader;
    }

    private Action<object, string> LazyLoader { get; set; }

    public int Id { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }

    public Blog Blog
    {
        get => LazyLoader.Load(this, ref _blog);
        set => _blog = value;
    }
}
```

Kod powyżej używa `Load` metodę rozszerzenia, aby wprowadzić nieco oczyszczarkę plików przy użyciu delegata:

```
public static class PocoLoadingExtensions
{
    public static TRelated Load<TRelated>(
        this Action<object, string> loader,
        object entity,
        ref TRelated navigationField,
        [CallerMemberName] string navigationName = null)
        where TRelated : class
    {
        loader?.Invoke(entity, navigationName);

        return navigationField;
    }
}
```

NOTE

Parametr konstruktora delegata ładowanych z opóźnieniem należy wywołać "lazyLoader". Konfigurację, aby użyć innej nazwy, niż jest to planowana w przyszłej wersji.

Powiązane dane i serializacja

Ponieważ właściwości nawigacji automatycznie poprawki będą programu EF Core, można na końcu cykli w grafie obiektu. Na przykład blog i jej powiązane wpisy ładowania spowoduje obiekt blogu, który odwołuje się zbiór wpisów. Każda z tych wpisów mają odwołaniem zwrotnym do blogu.

Niektóre środowiska serializacji nie zezwalają na tych cykli. Na przykład na składnik Json.NET zgłosi następujący wyjątek, jeśli okaże się cyklu.

```
Newtonsoft.Json.JsonSerializationException: Self odwołujące się do pętli wykryto dla właściwości "Blog" z typem "MyApplication.Models.Blog".
```

Jeśli używasz platformy ASP.NET Core, można skonfigurować program Json.NET, aby zignorować cykle, które znajdzie się na grafie obiektu. Jest to realizowane w `ConfigureServices(...)` method in `Startup.cs`.

```
public void ConfigureServices(IServiceCollection services)
{
    ...

    services.AddMvc()
        .AddJsonOptions(
            options => options.SerializerSettings.ReferenceLoopHandling =
Newtonsoft.Json.ReferenceLoopHandling.Ignore
        );

    ...
}
```

Klient programu vs. Ocena serwera

10.09.2018 • 2 minutes to read • [Edit Online](#)

Entity Framework Core obsługuje części zapytania Trwa obliczanie na urządzeniu klienckim i części wypychania do bazy danych. Jest dostawca bazy danych, aby określić części zapytania, które zostanie obliczone w bazie danych.

TIP

[Przykład](#) użyty w tym artykule można zobaczyć w witrynie GitHub.

Oceny klienta

W poniższym przykładzie metoda pomocnika służy do standaryzacji adresów URL dla blogów, z którego są zwarcane z bazy danych programu SQL Server. Ponieważ dostawca programu SQL Server nie wgląda w jaki sposób ta metoda jest implementowana, nie jest możliwe tłumaczenie SQL. Wszystkie inne aspekty zapytania są oceniane w bazie danych, ale przekazywanie zwarcanego `URL` za pośrednictwem tej metody jest wykonywane na komputerze klienckim.

```
var blogs = context.Blogs
    .OrderByDescending(blog => blog.Rating)
    .Select(blog => new
    {
        Id = blog.BlogId,
        Url = StandardizeUrl(blog.Url)
    })
    .ToList();
```

```
public static string StandardizeUrl(string url)
{
    url = url.ToLower();

    if (!url.StartsWith("http://"))
    {
        url = string.Concat("http://", url);
    }

    return url;
}
```

Problemy z wydajnością oceny klienta

Oceny klienta mogą być bardzo przydatne, w niektórych przypadkach go może spowodować obniżenie wydajności. Należy wziąć pod uwagę następujące zapytanie, której metody pomocnika teraz jest używane w filtrze. Ponieważ to nie może zostać wykonana w bazie danych, wszystkie dane są pobierane do pamięci, a następnie filtr jest stosowany na komputerze klienckim. W zależności od ilości danych, a ilość tych danych jest odfiltrowana może to spowodować spadek wydajności.

```
var blogs = context.Blogs
    .Where(blog => StandardizeUrl(blog.Url).Contains("dotnet"))
    .ToList();
```

Rejestrowanie oceny klienta

Domyślnie program EF Core będzie rejestrować ostrzeżenie podczas oceny klienta jest wykonywane. Zobacz [rejestrowania](#) więcej informacji na temat wyświetlania danych wyjściowych rejestrowania.

Zachowanie opcjonalne: zgłoszenie wyjątku dla oceny klienta

Można zmienić zachowanie w przypadku oceny klienta throw lub nic nie rób. Odbywa się podczas konfigurowania opcji kontekstu — zwykle w `DbContext.OnConfiguring`, lub `Startup.cs` korzystania z platformy ASP.NET Core.

```
protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
{
    optionsBuilder
        .UseSqlServer(@"Server=(localdb)\mssqllocaldb;Database=EFQuerying;Trusted_Connection=True;")
        .ConfigureWarnings(warnings => warnings.Throw(RelationalEventId.QueryClientEvaluationWarning));
}
```

Śledzenie programu vs. Bez śledzenia zapytań

28.08.2018 • 3 minutes to read • [Edit Online](#)

Czy platformy Entity Framework Core przechowuje informacje o wystąpienie jednostki w jego śledzenie zmian do śledzenia zachowania formantów. Jeśli jednostka jest śledzona, wszelkie zmiany wykryte w jednostce zostaną utrwalone w bazie danych podczas `SaveChanges()`. Entity Framework Core będzie również konfigurowania właściwości nawigacji między jednostkami, które są uzyskiwane z zapytania śledzenia i jednostek, które wcześniej zostały załadowane do wystąpienia typu `DbContext`.

TIP

Przykład użyty w tym artykule można zobaczyć w witrynie GitHub.

Śledzenia zapytań

Domyślnie są śledzenia zapytań, które zwracają typów jednostek. Oznacza to, można wprowadzić zmiany do tych wystąpień jednostki i utrwalonych te zmiany przez `SaveChanges()`.

W poniższym przykładzie zostanie wykryte i utrwalone w bazie danych podczas zmiany klasyfikacji blogi `SaveChanges()`.

```
using (var context = new BloggingContext())
{
    var blog = context.Blogs.SingleOrDefault(b => b.BlogId == 1);
    blog.Rating = 5;
    context.SaveChanges();
}
```

Bez śledzenia zapytań

Nie śledzenia zapytań są przydatne, gdy wyniki są używane w scenariuszu tylko do odczytu. Są one szybsze wykonywanie, ponieważ nie ma potrzeby informacje o monitorowaniu zmian konfiguracji.

Można wymienić określone zapytanie do śledzenia nie można:

```
using (var context = new BloggingContext())
{
    var blogs = context.Blogs
        .AsNoTracking()
        .ToList();
}
```

Możesz również zmienić domyślne zachowanie na poziomie wystąpienia kontekstu śledzenia:

```
using (var context = new BloggingContext())
{
    context.ChangeTracker.QueryTrackingBehavior = QueryTrackingBehavior.NoTracking;

    var blogs = context.Blogs.ToList();
}
```

NOTE

Nie śledzenia zapytań w dalszym ciągu wykonywać rozwiązywanie tożsamości w ramach zapytania przesyłać. Jeśli zestaw wyników zawiera tej samej jednostki wiele razy, to samo wystąpienie elementu Klasa jednostki zostanie zwrócony dla każdego wystąpienia w zestawie wyników. Jednak słabe odwołania są używane do śledzenia jednostek, które już zostały zwrócone. Jeśli poprzedni wynik z tą samą tożsamością wykracza poza zakres, a następnie uruchamia wyrzucanie elementów bezużytecznych, może pojawić się nowe wystąpienie jednostki. Aby uzyskać więcej informacji, zobacz [jak działa zapytanie](#).

Śledzenie i projekcji

Nawet jeśli typ wyniku zapytania nie jest typem jednostki, jeśli wynik zawiera typy jednostek nadal będą one śledzone domyślnie. W następującym zapytaniu, która zwraca typ anonimowy, wystąpienia `Blog` w wyniku będą śledzone zestawu.

```
using (var context = new BloggingContext())
{
    var blog = context.Blogs
        .Select(b =>
            new
            {
                Blog = b,
                Posts = b.Posts.Count()
            });
}
```

Jeśli zestaw wyników nie zawiera żadnych typów jednostek, Brak śledzenia jest wykonywana. W następującym zapytaniu, która zwraca typ anonimowy z niektórych wartości z jednostki (ale nie wystąpień typu rzeczywistego jednostek) nie jest Brak Śledzenia wykonywane.

```
using (var context = new BloggingContext())
{
    var blog = context.Blogs
        .Select(b =>
            new
            {
                Id = b.BlogId,
                Url = b.Url
            });
}
```

Pierwotne zapytania SQL

07.01.2019 • 6 minutes to read • [Edit Online](#)

Entity Framework Core umożliwia lista rozwijana umożliwiająca pierwotne zapytania SQL podczas pracy z usługą relacyjnej bazy danych. Może to być przydatne, jeśli zapytanie, które należy wykonać, nie można wyrazić za pomocą LINQ lub przy użyciu zapytania LINQ wynikiem jest nieefektywne zapytania SQL. Pierwotne zapytania SQL może zwrócić typów jednostek, lub, począwszy od programu EF Core 2.1, [typy zapytań](#) będących częścią modelu.

TIP

[Przykład](#) użyty w tym artykule można zobaczyć w witrynie GitHub.

Ograniczenia

Istnieją pewne ograniczenia, które należy zwrócić uwagę podczas korzystania z pierwotne zapytania SQL:

- Zapytanie SQL musi zwracać dane dotyczące wszystkich właściwości typu obiekt lub kwerendę.
- Nazwy kolumn w zestawie wyników muszą być zgodne nazwy kolumn, które są mapowane właściwości. Należy pamiętać, że to różni się od EF6 gdzie mapowania właściwości/kolumn została zignorowana dla pierwotne zapytania SQL i nazw musiały być zgodne z nazwami właściwości kolumny zestawu wyników.
- Zapytania SQL nie może zawierać powiązane dane. Jednak w wielu przypadkach można utworzyć na podstawie zapytania za pomocą `Include` operator, aby zwrócić dane dotyczące (zobacz [powiązanych danych w tym](#)).
- `SELECT` Konfigurowalna powinno być zazwyczaj instrukcji przekazany do tej metody: EF Core musi ocenić operatorów dodatkowych zapytań na serwerze (na przykład, aby przetłumaczyć operatorów LINQ zastosowane po `FromSql`), SQL podane są traktowane jak podzapytania. Oznacza to, SQL przekazywane nie powinna zawierać żadnych znaków lub opcje, które nie są prawidłowe w podzapytaniu, takich jak:
 - końcowe średnikami
 - W programie SQL Server, końcowe wskazówka poziomie zapytania (na przykład `OPTION (HASH JOIN)`)
 - W programie SQL Server `ORDER BY` klauzula, która nie jest powiązany z `TOP 100 PERCENT` w `SELECT` — klauzula
- Instrukcje SQL innych niż `SELECT` są rozpoznawane automatycznie jako innego niż konfigurowalna. W konsekwencji pełnych wyników procedury składowane zawsze są zwracane do klienta i dowolnych operatorów LINQ zastosowane po `FromSql` jest oceniana w pamięci.

Podstawowe pierwotne zapytania SQL

Możesz użyć `FromSql` metodę rozszerzenia, aby rozpoczęć zapytanie LINQ, na podstawie pierwotne zapytania SQL.

```
var blogs = context.Blogs
    .FromSql("SELECT * FROM dbo.Blogs")
    .ToList();
```

Pierwotne zapytania SQL może służyć do wykonywania procedury składowanej.

```
var blogs = context.Blogs
    .FromSql("EXECUTE dbo.GetMostPopularBlogs")
    .ToList();
```

Przekazywanie parametrów

Podobnie jak w przypadku dowolnego interfejsu API, który akceptuje SQL, jest ważne, aby zdefiniować parametry wszystkie dane wejściowe użytkownika mają ochronę przed ataku polegającego na iniekcji SQL. Można zawierać symbole zastępcze parametr ciągu zapytania SQL, a następnie podaj wartości parametrów jako dodatkowe argumenty. Możesz podać wartości parametrów zostaną automatycznie przekonwertowane na `SqlParameter`.

Poniższy przykład przekazuje pojedynczy parametr do procedury składowanej. Chociaż może to wyglądać jak `String.Format` składni, podana wartość jest opakowana w miejsce dodaje parametr i nazwę parametru wygenerowanego `{0}` określono symbolu zastępczego.

```
var user = "johndoe";

var blogs = context.Blogs
    .FromSql("EXECUTE dbo.GetMostPopularBlogsForUser {0}", user)
    .ToList();
```

To jest taka sama zapytania, ale przy użyciu składni interpolacji ciągu, który jest obsługiwany w programie EF Core 2.0 i nowszych:

```
var user = "johndoe";

var blogs = context.Blogs
    .FromSql($"EXECUTE dbo.GetMostPopularBlogsForUser {user}")
    .ToList();
```

Można także skonstruować `SqlParameter` i podać go jako wartość parametru. Dzięki temu można użyć nazwanych parametrów ciągu zapytania SQL.

```
var user = new SqlParameter("user", "johndoe");

var blogs = context.Blogs
    .FromSql("EXECUTE dbo.GetMostPopularBlogsForUser @user", user)
    .ToList();
```

Tworzenie za pomocą LINQ

Jeśli zapytanie SQL może być składana na w bazie danych, można utworzyć na podstawie początkowego pierwotne zapytania SQL przy użyciu operatorów LINQ. Zapytania SQL, które mogą być składane na zaczynają się od `SELECT` — słowo kluczowe.

W poniższym przykładzie użyto pierwotne zapytanie SQL, które wybiera z funkcji Table-Valued (TVF), a następnie komponuje się na nim za pomocą LINQ, aby wykonać filtrowanie i sortowanie.

```
var searchTerm = ".NET";

var blogs = context.Blogs
    .FromSql($"SELECT * FROM dbo.SearchBlogs({searchTerm})")
    .Where(b => b.Rating > 3)
    .OrderByDescending(b => b.Rating)
    .ToList();
```

W tym powiązane dane

Tworzenie przy użyciu operatorów LINQ, może służyć do uwzględnienia powiązanych danych w zapytaniu.

```
var searchTerm = ".NET";

var blogs = context.Blogs
    .FromSql($"SELECT * FROM dbo.SearchBlogs({searchTerm})")
    .Include(b => b.Posts)
    .ToList();
```

WARNING

Zawsze używaj parametryzacji pierwotne zapytania SQL: Interfejsy API, które akceptują pierwotne SQL string, takich jak `FromSql` i `ExecuteSqlCommand` Zezwalaj na wartości, które mają być łatwo przekazywane jako parametry. Oprócz sprawdzania poprawności danych wejściowych użytkownika, należy zawsze używać parametryzacji dla każdej wartości pierwotne zapytania SQL/polecenia. Jeśli używasz ciągów dynamicznie tworzenie dowolnej części ciągu zapytania, a następnie ponosisz odpowiedzialność za sprawdzanie poprawności wszystkie dane wejściowe, aby zapewnić ochronę przed atakami polegającymi na iniekcji SQL.

Zapytania asynchroniczne

28.08.2018 • 2 minutes to read • [Edit Online](#)

Zapytania asynchroniczne uniknąć blokowania wątku podczas wykonywania zapytania w bazie danych. Może to być przydatne w celu uniknięcia zawieszenia się interfejsu użytkownika aplikacji klienckiej grubości. Operacje asynchroniczne także może zwiększyć przepływność w aplikacji sieci web, w którym wątku można zwolnić, do obsługi innych żądań podczas końca operacji bazy danych. Aby uzyskać więcej informacji, zobacz [asynchronicznego programowania w języku C#](#).

WARNING

EF Core nie obsługuje wielu operacji równoległych, które są uruchamiane na tym samym wystąpieniu kontekstu. Należy zawsze poczekać na zakończenie operacji przed rozpoczęciem następnej operacji. Zazwyczaj jest to wykonywanie przy użyciu `await` słowo kluczowe w każdej operacji asynchronicznej.

Entity Framework Core udostępnia zestaw metod asynchronicznego rozszerzenia, które mogą być używane jako alternatywę dla metody LINQ, które powodują zapytanie do wykonania i zwracanie wyników. Przykłady obejmują `ToListAsync()`, `ToDictionaryAsync()`, `SingleAsync()` itp. Brak wersji asynchronicznej operatorów LINQ takich jak `Where(...)`, `OrderBy(...)`, itp., ponieważ te metody tylko zbudowania drzewa wyrażeń LINQ i nie powodują zapytanie do wykonania w bazie danych.

IMPORTANT

Metody rozszerzenia programu EF Core asynchroniczne są zdefiniowane w `Microsoft.EntityFrameworkCore` przestrzeni nazw. Ta przestrzeń nazw, należy zimportować dla metod, które mają być dostępne.

```
public async Task<List<Blog>> GetBlogsAsync()
{
    using (var context = new BloggingContext())
    {
        return await context.Blogs.ToListAsync();
    }
}
```

Jak działają zapytań

29.09.2018 • 4 minutes to read • [Edit Online](#)

Entity Framework Core używa Language Integrated Query (LINQ) do zapytania o dane z bazy danych. LINQ umożliwia przy użyciu języka C# (lub wybranym języku .NET) do pisania zapytań silnie typizowaną oparte na klasach pochodnych kontekstu i jednostki.

Czas życia zapytania

Poniżej znajduje się przegląd wysokiego poziomu procesu, który przechodzi każdego zapytania.

1. Zapytania LINQ są przetwarzane przez program Entity Framework Core do budowania reprezentacji, które jest gotowe do przetworzenia przez dostawcę bazy danych
 - a. Wynik jest buforowany, dzięki czemu to przetwarzanie nie musi odbywać się za każdym razem, gdy zapytanie jest wykonywane
2. Wynik jest przekazywany do dostawcy bazy danych
 - a. Dostawca bazy danych identyfikuje części zapytania, które mogą być obliczane w bazie danych
 - b. Te części zapytania są tłumaczone na język użytego zapytania bazy danych (na przykład SQL dla relacyjnej bazy danych)
 - c. Co najmniej jeden zapytania są wysyłane do bazy danych i zwróconym zestawie wyników (wyniki są wartości z bazy danych, a nie wystąpień jednostek)
3. Dla każdego elementu w zestawie wyników
 - a. Jeśli jest to zapytania śledzenia, EF sprawdza, czy dane reprezentuje jednostkę już w śledzeniu zmian dla wystąpienia kontekstu
 - Jeśli tak, zwracany jest istniejąca jednostka
 - W przeciwnym razie jest tworzona nowa jednostka skonfigurowano śledzenie zmian i zwracany jest nowa jednostka
 - b. Jeśli zapytania śledzenia nie EF sprawdza, czy dane reprezentuje jednostkę już w zestawie wyników dla tego zapytania
 - Jeśli tak, jest zwracana istniejącej jednostki ⁽¹⁾
 - Jeśli nie, nowy obiekt jest tworzony i zwracany

⁽¹⁾ Nie śledzenia zapytań za pomocą słabe odwołania do śledzenie jednostek, które już zostały zwrócone. Jeśli poprzedni wynik z tą samą tożsamością wykracza poza zakres, a następnie uruchamia wyrzucanie elementów bezużytecznych, może pojawić się nowe wystąpienie jednostki.

Gdy zapytania są wykonywane.

Po wywołaniu operatorów LINQ, są po prostu budowania reprezentacji w pamięci, zapytania. Zapytanie jest wysyłane do bazy danych tylko wtedy, gdy wyniki są używane.

Najbardziej typowe operacje, których wynikiem zapytania są wysyłane do bazy danych są:

- Iteracja wyniki w parametrze `for` pętli
- Przy użyciu operatora, takich jak `ToList`, `ToArray`, `Single`, `Count`
- Powiązanie danych z wynikami zapytania do interfejsu użytkownika

WARNING

Zawsze weryfikowały dane wejściowe użytkownika: podczas EF Core chroni przed atakami polegającymi na iniekcji SQL przy użyciu parametrów i anulowania zapewnianego elementu literałów w zapytaniach, go nie sprawdza poprawność danych wejściowych. Weryfikacji odpowiednią dla wymagań aplikacji należy wykonać przed wartości ze źródeł niezaufanych są używane w kwerendach LINQ, przypisany do właściwości jednostki lub przekazywane do innych interfejsów EF Core API. Dotyczy to danych podawanych przez użytkownika używane do dynamicznego utworzenia kwerendy. Nawet wtedy, gdy za pomocą LINQ, jeśli użytkownik akceptuje dane wejściowe użytkownika, aby zbudować wyrażenia, należy się upewnić, że można konstruować tylko wyrażenia zamierzone.

Filtры запросов глобальных

08.12.2018 • 3 minutes to read • [Edit Online](#)

Фильтры запросов глобальных являются предикатами запросов LINQ (выражения логического типа, которые передаются в программу LINQ *где* — оператор запросов) применяемые к типам единиц в моделью метаданных (обычно в *OnModelCreating*). Такие фильтры автоматически применяются ко всем запросам LINQ, исключающие эти типы единиц, включая ссылки на свойства навигации, опосредованное обращение, такие как при использовании `Include` или `bezpośredniego` тип единиц. Некоторые типовые приложения этой функции следующие:

- **Удаление** — Означает тип единицы `IsDeleted` свойства.
- **Многодоступность** — Означает тип единицы `TenantId` свойства.

Пример

Ниже приведен пример использования глобальных фильтров для реализации политики удаления и поддержки множественных владельцев в простом модельном представлении для управления блогами.

TIP

Пример используется в этом артикуле можно просмотреть в [GitHub](#).

Сначала нужно определить единицы:

```
public class Blog
{
    private string _tenantId;

    public int BlogId { get; set; }
    public string Name { get; set; }
    public string Url { get; set; }

    public List<Post> Posts { get; set; }
}

public class Post
{
    public int PostId { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }
    public bool IsDeleted { get; set; }

    public int BlogId { get; set; }
    public Blog Blog { get; set; }
}
```

Нужно помнить, что декларация `_tenantId` на *blogu* единицы. Это поможет связать каждое появление блога с определенной владельцем. Также определена `IsDeleted` свойство *wpis* типа единицы. Используется для отслеживания, что *wpis* появление было "все удалены". Это означает, что появление было удалено без физического удаления базовых данных.

Затем настройте фильтры запросов в *OnModelCreating* при использовании `HasQueryFilter` интерфейса API.

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Blog>().Property<string>("TenantId").HasField("_tenantId");

    // Configure entity filters
    modelBuilder.Entity<Blog>().HasQueryFilter(b => EF.Property<string>(b, "TenantId") == _tenantId);
    modelBuilder.Entity<Post>().HasQueryFilter(p => !p.IsDeleted);
}
```

Predykatu wyrażenie przekazany do *HasQueryFilter* wywołania teraz zostaną automatycznie zastosowane na żadne zapytania LINQ dla tych typów.

TIP

Zwróć uwagę na użycie pola poziomu wystąpienia typu DbContext: `_tenantId` używane do ustawiania bieżącej dzierżawy. Filtry na poziomie modelu będzie używać wartości z wystąpienia poprawny kontekst (oznacza to, że wystąpienie, które jest wykonywane zapytanie).

Wyłączanie filtrów

Filtryle mogą być wyłączone dla poszczególnych zapytań LINQ za pomocą `IgnoreQueryFilters()` operatora.

```
blogs = db.Blogs
    .Include(b => b.Posts)
    .IgnoreQueryFilters()
    .ToList();
```

Ograniczenia

Filtryle zapytań globalnych mają następujące ograniczenia:

- Filtry nie może zawierać odwołań do właściwości nawigacji.
- Filtry można zdefiniować tylko dla głównego typu jednostki z hierarchii dziedziczenia.

Tagi kwerendy

16.11.2018 • 2 minutes to read • [Edit Online](#)

NOTE

Ta funkcja jest nowa w programie EF Core 2.2.

Funkcja ta ułatwia korelowanie zapytań LINQ w kodzie za pomocą wygenerowanego zapytań SQL przechwycone w dziennikach. Dodawanie adnotacji do zapytań LINQ, za pomocą nowego `TagWith()` metody:

```
var nearestFriends =
    (from f in context.Friends.TagWith("This is my spatial query!")
     orderby f.Location.Distance(myLocation) descending
     select f).Take(5).ToList();
```

To zapytanie LINQ jest tłumaczona na następującą instrukcję SQL:

```
-- This is my spatial query!

SELECT TOP(@__p_1) [f].[Name], [f].[Location]
FROM [Friends] AS [f]
ORDER BY [f].[Location].STDistance(@__myLocation_0) DESC
```

Istnieje możliwość wywołania `TagWith()` wiele razy na tego samego zapytania. Tagi kwerendy kumulują się. Na przykład biorąc pod uwagę następujące metody:

```
IQueryable<Friend> GetNearestFriends(Point myLocation) =>
    from f in context.Friends.TagWith("GetNearestFriends")
    orderby f.Location.Distance(myLocation) descending
    select f;

IQueryable<T> Limit<T>(IQueryable<T> source, int limit) =>
    source.TagWith("Limit").Take(limit);
```

Następujące zapytanie:

```
var results = Limit(GetNearestFriends(myLocation), 25).ToList();
```

Przekłada się na:

```
-- GetNearestFriends

-- Limit

SELECT TOP(@__p_1) [f].[Name], [f].[Location]
FROM [Friends] AS [f]
ORDER BY [f].[Location].STDistance(@__myLocation_0) DESC
```

Użytkownik może również używać ciągów wielowierszowe jako tagi zapytania. Na przykład:

```
var results = Limit(GetNearestFriends(myLocation), 25).TagWith(  
    @"This is a multi-line  
    string").ToList();
```

Generuje następujące instrukcje SQL:

```
-- GetNearestFriends  
  
-- Limit  
  
-- This is a multi-line  
-- string  
  
SELECT TOP(@__p_1) [f].[Name], [f].[Location]  
FROM [Friends] AS [f]  
ORDER BY [f].[Location].STDistance(@__myLocation_0) DESC
```

Znane ograniczenia

Tagi kwerendy nie można sparametryzować: programu EF Core zawsze traktuje tagi kwerendy w zapytaniu LINQ jako literały ciągu, które znajdują się w wygenerowanej tabeli SQL. Zapytania skompilowane, które przyjmują tagi kwerendy, ponieważ parametry nie są dozwolone.

Zapisywanie danych

28.08.2018 • 2 minutes to read • [Edit Online](#)

Każde wystąpienie kontekstu ma `ChangeTracker` odpowiadające do śledzenia zmian, które muszą zostać zapisane w bazie danych. Po wprowadzeniu dowolnych zmian do wystąpień klas jednostek, zmiany te są rejestrowane w `ChangeTracker` i następnie zapisywane w bazie danych podczas wywoływanego `SaveChanges`. Dostawca bazy danych jest odpowiedzialny za tłumaczenie zmiany do określonej bazy danych operacji (na przykład `INSERT`, `UPDATE`, i `DELETE` polecenia do relacyjnej bazy danych).

Zapisywanie podstawowe

28.08.2018 • 2 minutes to read • [Edit Online](#)

Dowiedz się, jak dodawanie, modyfikowanie i usuwanie danych przy użyciu klas jednostek i kontekstu.

TIP

Przykład użyty w tym artykule można zobaczyć w witrynie GitHub.

Dodawanie danych

Użyj `DbSet.Add` metodę, aby dodać nowe wystąpienia klas jednostek. Dane zostanie wstawiony w bazie danych podczas wywoływania `SaveChanges`.

```
using (var context = new BloggingContext())
{
    var blog = new Blog { Url = "http://sample.com" };
    context.Blogs.Add(blog);
    context.SaveChanges();
}
```

TIP

Metody `Add`, `Dołącz` i aktualizacji, wszystkie robocze na pełny wykres jednostek jest przekazywany do nich, zgodnie z opisem w [powiązanych danych](#) sekcji. Alternatywnie właściwość `EntityEntry.State` może służyć do ustawiania stanu pojedynczej jednostki. Na przykład `context.Entry(blog).State = EntityState.Modified`.

Aktualizowanie danych

EF automatycznie wykrywa zmiany wprowadzone do istniejącej jednostki, która jest śledzona przez kontekst. Obejmuje to jednostki, które możesz obciążenia/zapytań z bazy danych i jednostek, które zostały wcześniej dodane i zapisane w bazie danych.

Po prostu zmodyfikuj wartości przypisane do właściwości, a następnie wywołać `SaveChanges`.

```
using (var context = new BloggingContext())
{
    var blog = context.Blogs.First();
    blog.Url = "http://sample.com/blog";
    context.SaveChanges();
}
```

Usuwanie danych

Użyj `DbSet.Remove` metody, można usunąć wystąpień klas jednostek.

Jeśli ta jednostka już istnieje w bazie danych, zostaną usunięte podczas `SaveChanges`. Jeśli jednostka nie został jeszcze zapisany w bazie danych (czyli jego śledzenia dodawania) zostaną usunięte z kontekstu, a nie będzie już wstawiany gdy `SaveChanges` jest wywoływana.

```
using (var context = new BloggingContext())
{
    var blog = context.Blogs.First();
    context.Blogs.Remove(blog);
    context.SaveChanges();
}
```

Wiele operacji w jednym SaveChanges

Można połączyć wiele operacji Dodawanie/aktualizowanie/usuwanie na jedno wywołanie *SaveChanges*.

NOTE

W przypadku większości dostawców bazy danych *SaveChanges* jest transakcyjna. Oznacza to wszystkie operacje będą sukcesem lub niepowodzeniem, a operacje nigdy nie lewej częściowo zastosowanych.

```
using (var context = new BloggingContext())
{
    // seeding database
    context.Blogs.Add(new Blog { Url = "http://sample.com/blog" });
    context.Blogs.Add(new Blog { Url = "http://sample.com/another_blog" });
    context.SaveChanges();
}

using (var context = new BloggingContext())
{
    // add
    context.Blogs.Add(new Blog { Url = "http://sample.com/blog_one" });
    context.Blogs.Add(new Blog { Url = "http://sample.com/blog_two" });

    // update
    var firstBlog = context.Blogs.First();
    firstBlog.Url = "";

    // remove
    var lastBlog = context.Blogs.Last();
    context.Blogs.Remove(lastBlog);

    context.SaveChanges();
}
```

Zapisywanie powiązanych danych

28.08.2018 • 4 minutes to read • [Edit Online](#)

Oprócz izolowanych jednostek można wprowadzić korzystanie z relacji zdefiniowanych w modelu.

TIP

Przykład użyty w tym artykule można zobaczyć w witrynie GitHub.

Dodawanie wykresu nowych jednostek

Jeśli utworzysz kilka nowych jednostek powiązanych, dodając jeden z nich do kontekstu spowoduje, że inne osoby mają zostać dodane za.

W poniższym przykładzie blogu i trzy powiązane wpisy są wszystkie wstawione do bazy danych. Znalezione wpisy i dodawane, ponieważ są one dostępne za pośrednictwem `Blog.Posts` właściwości nawigacji.

```
using (var context = new BloggingContext())
{
    var blog = new Blog
    {
        Url = "http://blogs.msdn.com/dotnet",
        Posts = new List<Post>
        {
            new Post { Title = "Intro to C#" },
            new Post { Title = "Intro to VB.NET" },
            new Post { Title = "Intro to F#" }
        }
    };
    context.Blogs.Add(blog);
    context.SaveChanges();
}
```

TIP

Właściwość `EntityEntry.State` służy do ustawiania stanu pojedynczej jednostki. Na przykład

```
context.Entry(blog).State = EntityState.Modified
```

Dodawanie powiązanej jednostki

Jeśli odwołujesz nową jednostkę z właściwością nawigacji jednostki, która jest już śledzony przez kontekst jednostki będą wykrywane i wstawione do bazy danych.

W poniższym przykładzie `post` dodaje się jednostki, ponieważ jest ona dodawana do `Posts` właściwość `blog` jednostki, która została pobrana z bazy danych.

```
using (var context = new BloggingContext())
{
    var blog = context.Blogs.Include(b => b.Posts).First();
    var post = new Post { Title = "Intro to EF Core" };

    blog.Posts.Add(post);
    context.SaveChanges();
}
```

Zmiana relacji

Jeśli zmienisz właściwości nawigacji jednostki, odpowiednie zmiany będą kolumna klucza obcego w bazie danych.

W poniższym przykładzie `post` jednostki zostanie zaktualizowany i będzie należeć do nowego `blog` jednostki ponieważ jej `Blog` właściwość nawigacji jest ustawiona, aby wskazywać `blog`. Należy pamiętać, że `blog` również zostaną wstawione do bazy danych, ponieważ jest nowa jednostka, która odwołuje się do właściwości nawigacji jednostki, która jest już śledzona przez kontekst (`post`).

```
using (var context = new BloggingContext())
{
    var blog = new Blog { Url = "http://blogs.msdn.com/visualstudio" };
    var post = context.Posts.First();

    post.Blog = blog;
    context.SaveChanges();
}
```

Usuwanie relacji

Możesz usunąć relację, ustawiając nawigacji odwołanie `null`, lub usuwanie obiektu pokrewnego nawigacji kolekcji.

Usunięcie relacji może mieć skutki uboczne jednostki zależne, zgodnie z każdym skonfigurowane w relacji zachowanie przy usuwaniu.

Domyślnie dla wymaganych relacji skonfigurowano zachowanie dotyczące usuwania cascade, a jednostki podrzędne/zależnych od ustawień lokalnych, które zostaną usunięte z bazy danych. W przypadku relacji opcjonalne usuwanie kaskadowe nie skonfigurowano domyślnie, ale zostanie ustawiona właściwość klucza obcego na wartość null.

Zobacz [relacje wymaganych i opcjonalnych](#) Aby dowiedzieć się, jak można skonfigurować requiredness relacji.

Zobacz [usuwanie kaskadowe](#) szczegółowe informacje na temat sposobu usuwanie kaskadowe zachowania działa jak może być jawnie skonfigurowane i jak są wybrane przez Konwencję.

W poniższym przykładzie skonfigurowano usuwanie kaskadowe relacji między `Blog` i `Post`, więc `post` została usunięta z bazy danych.

```
using (var context = new BloggingContext())
{
    var blog = context.Blogs.Include(b => b.Posts).First();
    var post = blog.Posts.First();

    blog.Posts.Remove(post);
    context.SaveChanges();
}
```

Usuwanie kaskadowe

12.09.2018 • 21 minutes to read • [Edit Online](#)

Usuwanie kaskadowe jest najczęściej używany w terminologii bazy danych do opisania cechę, która umożliwia usunięcie wiersza automatycznie wyzwalać usunięcie powiązane wiersze. Ścisłe powiązanych koncepcji, również pasuje do żadnego zachowań usuwania programu EF Core jest automatyczne usuwanie obiektu podzielnego, gdy relacji do elementu nadzielnego zostały oddzielone — jest to często nazywane "Usuwanie porzucone".

EF Core implementuje kilka różnych delete zachowań i umożliwia konfigurację zachowania Usuń poszczególne relacji. EF Core implementuje również konwencje, które automatycznie konfigurują zachowania delete przydatne domyślne dla każdej relacji na podstawie [requiredness relacji](#).

Usuń zachowania

Usuń zachowania są zdefiniowane w *DeleteBehavior* moduł wyliczający typu i mogą być przekazywane do *OnDelete* wygodnego interfejsu API do kontroli czy usunięcie jednostki jednostki/element nadzienny lub severing programu Relacja do podmiotów zależnych od ustawień lokalnych/podzienny musi mieć efekt uboczny w jednostek zależnych od ustawień lokalnych/podzienny.

Dostępne są trzy akcje, które EF można podjąć, gdy jednostka jednostki/nadzienna została usunięta lub jest oddzielone relacji do elementu nadziennego:

- Można je usunąć nadziennych/zależnych od ustawień lokalnych
- Wartości klucza obcego elementu nadziennego można ustawić na wartość null
- Element nadzienny nie jest zmieniany

NOTE

Zachowanie dotyczące usuwania konfigurowane w modelu platformy EF Core jest stosowane tylko jednostki głównej jest usuwana za pomocą programu EF Core i podmioty zależne są ładowane do pamięci (czyli pod kątem śledzonych zależności). Odpowiednie zachowania kaskadowe musi być Instalator w bazie danych, aby upewnić się, dane, które nie jest śledzony przez kontekst ma niezbędnych działań, które są stosowane. Jeśli użyjesz programu EF Core do utworzenia bazy danych tego zachowania kaskadowe będzie Instalatora dla Ciebie.

Dla drugiego z powyższej akcji ustawienie wartości klucza obcego o wartości null jest nieprawidłowa w przypadku klucza obcego nie dopuszcza wartości null. (Dopuszcza klucz obcy jest odpowiednikiem wymaganej relacji). W takich przypadkach programu EF Core śledzi, czy właściwość klucza obcego została oznaczona jako wartości null do momentu SaveChanges jest wywoływana, co jest zgłoszany wyjątek, ponieważ zmiana nie może zostać utrwalona w bazie danych. Jest to podobne do pobierania naruszenie ograniczenia z bazy danych.

Istnieją cztery Usuń zachowań, zgodnie z opisem w poniższych tabelach.

Opcjonalne relacji

W przypadku relacji opcjonalne (dopuszcza wartości null z kluczem obcym) on jest możliwa zapisać wartości null wartości klucza obcego, co powoduje następujące skutki:

NAZWA ZACHOWANIA	WPŁYW NA ZALEŻNYCH OD USTAWIEŃ LOKALNYCH/PODRZĘDNY W PAMIĘCI	WPŁYW NA ZALEŻNYCH OD USTAWIEŃ LOKALNYCH/PODRZĘDNEJ BAZY DANYCH
Kaskadowe	Jednostki są usuwane.	Jednostki są usuwane.

NAZWA ZACHOWANIA	WPŁYW NA ZALEŻNYCH OD USTAWIEŃ LOKALNYCH/PODRZĘDNY W PAMIĘCI	WPŁYW NA ZALEŻNYCH OD USTAWIEŃ LOKALNYCH/PODRZĘDNEJ BAZY DANYCH
ClientSetNull (opcja domyślna)	Właściwości klucza obcego są ustawione na wartość null	Brak
SetNull	Właściwości klucza obcego są ustawione na wartość null	Właściwości klucza obcego są ustawione na wartość null
ograniczenia	Brak	Brak

Wymagane relacje

W przypadku relacji (innych niż null z kluczem obcym), wymagane jest *nie* można zapisać wartości null wartości klucza obcego, co powoduje następujące skutki:

NAZWA ZACHOWANIA	WPŁYW NA ZALEŻNYCH OD USTAWIEŃ LOKALNYCH/PODRZĘDNY W PAMIĘCI	WPŁYW NA ZALEŻNYCH OD USTAWIEŃ LOKALNYCH/PODRZĘDNEJ BAZY DANYCH
Kaskadowe (opcja domyślna)	Jednostki są usuwane.	Jednostki są usuwane.
ClientSetNull	Zgłasza SaveChanges	Brak
SetNull	Zgłasza SaveChanges	Zgłasza SaveChanges
ograniczenia	Brak	Brak

W tabelach powyżej *Brak* może doprowadzić do naruszenia ograniczenia. Na przykład jeśli jednostka jednostki/podrzędny jest usuwana, lecz nie podjęto żadnej akcji można zmienić klucza obcego z zależnych od ustawień lokalnych/podrzędny, następnie bazie danych prawdopodobnie zgłosi na SaveChanges z powodu naruszenia ograniczenia obcego.

Na wysokim poziomie:

- Jeśli masz jednostek, które nie mogą istnieć bez klasy nadrzędnej, a EF automatyzującą automatycznego usuwania elementy podrzędne, a następnie użyć *Cascade*.
 - Jednostki, które nie mogą znajdować się bez nadrzędnej zwykle należy na użytku wymaganych relacji, które *Cascade* jest ustawieniem domyślnym.
- Jeśli masz jednostek, które mogą lub nie może mieć elementu nadrzędnego i EF, aby zadbać o nulling się klucz obcy dla Ciebie, a następnie użyć *ClientSetNull*
 - Jednostek, które może istnieć bez nadrzędnej zwykle należy użyć opcjonalne relacji, dla którego *ClientSetNull* jest ustawieniem domyślnym.
 - Jeśli mają również próbować nawet propagację wartości null do podrzędnych klucze obce w bazie danych po jednostce podrzędnej nie został załadowany, następnie za pomocą *SetNull*. Jednak pamiętać, że baza danych musi obsługiwać to Konfigurowanie bazy danych, np. to może spowodować inne ograniczenia, co w praktyce często sprawia, że ta opcja niepraktyczne. Dlatego *SetNull* nie jest ustawieniem domyślnym.
- Jeśli nie chcesz programu EF Core kiedykolwiek automatycznie Usuń jednostkę lub wartość null out klucza obcego, automatycznie, a następnie użyć *Ogranicz*. Należy pamiętać, że wymaga to, że Twój kod Synchronizuj podrzędnych jednostek i ich wartości klucza obcego ręcznie w przeciwnym razie ograniczenie wyjątki zostanie wygenerowany.

NOTE

W programie EF Core, w odróżnieniu od EF6 efekty kaskadowych nie nawiązały natychmiast, ale zamiast tego tylko wtedy, gdy jest wywoływana SaveChanges.

NOTE

Zmiany w programie EF Core 2.0: w poprzednich wersjach *Ogranicz* spowodowałoby opcjonalne właściwości klucza obcego w śledzonych jednostki zależne, należy ustawić wartości null i zostało domyślne zachowanie dla relacji opcjonalnej przy usuwaniu. W programie EF Core 2.0 *ClientSetNull* została wprowadzona do reprezentowania tego zachowania i stało się domyślnie w przypadku relacji opcjonalnej. Zachowanie *Ogranicz* została dostosowana do nigdy nie ma żadnych efektów ubocznych na jednostki zależne.

Przykłady usuwania jednostki

Poniższy kod jest częścią [przykładowe](#), można pobrać i uruchomić. Przykład pokazuje, co się stanie dla każdego zachowania dotyczące usuwania dla relacji wymagane i opcjonalne, po usunięciu obiektu nadziednego.

```
var blog = context.Blogs.Include(b => b.Posts).First();
var posts = blog.Posts.ToList();

DumpEntities(" After loading entities:", context, blog, posts);

context.Remove(blog);

DumpEntities($" After deleting blog '{blog.BlogId}':", context, blog, posts);

try
{
    Console.WriteLine();
    Console.WriteLine(" Saving changes:");

    context.SaveChanges();

    DumpSql();

    DumpEntities(" After SaveChanges:", context, blog, posts);
}
catch (Exception e)
{
    DumpSql();

    Console.WriteLine();
    Console.WriteLine($" SaveChanges threw {e.GetType().Name}: {(e is DbUpdateException ? e.InnerException.Message : e.Message)}");
}
```

Przejdzmy teraz przez poszczególnych odmian, aby zrozumieć, co się dzieje.

DeleteBehavior.Cascade z relacją wymagane lub opcjonalne

```
After loading entities:  
Blog '1' is in state Unchanged with 2 posts referenced.  
Post '1' is in state Unchanged with FK '1' and reference to blog '1'.  
Post '2' is in state Unchanged with FK '1' and reference to blog '1'.
```

```
After deleting blog '1':  
Blog '1' is in state Deleted with 2 posts referenced.  
Post '1' is in state Unchanged with FK '1' and reference to blog '1'.  
Post '2' is in state Unchanged with FK '1' and reference to blog '1'.
```

```
Saving changes:  
DELETE FROM [Posts] WHERE [PostId] = 1  
DELETE FROM [Posts] WHERE [PostId] = 2  
DELETE FROM [Blogs] WHERE [BlogId] = 1
```

```
After SaveChanges:  
Blog '1' is in state Detached with 2 posts referenced.  
Post '1' is in state Detached with FK '1' and no reference to a blog.  
Post '2' is in state Detached with FK '1' and no reference to a blog.
```

- Blog jest oznaczony jako usunięty
- Wpisy początkowo pozostają Unchanged, ponieważ kaskady tak się nie stanie do momentu SaveChanges
- SaveChanges wysyła usuwa zarówno zależności/podrębnych (wpisy), a następnie jednostkę/element nadzędny (blog)
- Po zapisaniu wszystkich jednostek są odłączone od teraz zostały usunięte z bazy danych

DeleteBehavior.ClientSetNull lub DeleteBehavior.SetNull z wymaganą relacją

```
After loading entities:  
Blog '1' is in state Unchanged with 2 posts referenced.  
Post '1' is in state Unchanged with FK '1' and reference to blog '1'.  
Post '2' is in state Unchanged with FK '1' and reference to blog '1'.
```

```
After deleting blog '1':  
Blog '1' is in state Deleted with 2 posts referenced.  
Post '1' is in state Unchanged with FK '1' and reference to blog '1'.  
Post '2' is in state Unchanged with FK '1' and reference to blog '1'.
```

```
Saving changes:  
UPDATE [Posts] SET [BlogId] = NULL WHERE [PostId] = 1
```

```
SaveChanges threw DbUpdateException: Cannot insert the value NULL into column 'BlogId', table  
'EFSaving.CascadeDelete.dbo.Posts'; column does not allow nulls. UPDATE fails. The statement has been  
terminated.
```

- Blog jest oznaczony jako usunięty
- Wpisy początkowo pozostają Unchanged, ponieważ kaskady tak się nie stanie do momentu SaveChanges
- SaveChanges podejmuje próbę ustawienia wpisu klucza Obcego na wartość null, ale to nie powiodło się ponieważ klucza Obcego nie dopuszcza wartości null

DeleteBehavior.ClientSetNull lub DeleteBehavior.SetNull z opcjonalną relacją

```

After loading entities:
Blog '1' is in state Unchanged with 2 posts referenced.
Post '1' is in state Unchanged with FK '1' and reference to blog '1'.
Post '2' is in state Unchanged with FK '1' and reference to blog '1'.

After deleting blog '1':
Blog '1' is in state Deleted with 2 posts referenced.
Post '1' is in state Unchanged with FK '1' and reference to blog '1'.
Post '2' is in state Unchanged with FK '1' and reference to blog '1'.

Saving changes:
UPDATE [Posts] SET [BlogId] = NULL WHERE [PostId] = 1
UPDATE [Posts] SET [BlogId] = NULL WHERE [PostId] = 2
DELETE FROM [Blogs] WHERE [BlogId] = 1

After SaveChanges:
Blog '1' is in state Detached with 2 posts referenced.
Post '1' is in state Unchanged with FK 'null' and no reference to a blog.
Post '2' is in state Unchanged with FK 'null' and no reference to a blog.

```

- Blog jest oznaczony jako usunięty
- Wpisy początkowo pozostać Unchanged, ponieważ kaskady tak się nie stanie do momentu SaveChanges
- Próby SaveChanges ustawia klucza Obcego z obu zależności/elementów podrzędnych (ogłoszeń) o wartości null przed usunięciem jednostki/element nadrzędny (blog)
- Po zapisaniu jednostki/element nadrzędny (blog) zostanie usunięty, ale zależności/elementy podrzędne (ogłoszeń) nadal są śledzone.
- Śledzone zależności/elementy podrzędne (ogłoszeń) ma wartości null klucza Obcego, a ich odwołania do jednostki/nadrzędnych usunięte (blog) została usunięta.

DeleteBehavior.Restrict z relacją wymagane lub opcjonalne

```

After loading entities:
Blog '1' is in state Unchanged with 2 posts referenced.
Post '1' is in state Unchanged with FK '1' and reference to blog '1'.
Post '2' is in state Unchanged with FK '1' and reference to blog '1'.

After deleting blog '1':
Blog '1' is in state Deleted with 2 posts referenced.
Post '1' is in state Unchanged with FK '1' and reference to blog '1'.
Post '2' is in state Unchanged with FK '1' and reference to blog '1'.

Saving changes:
SaveChanges threw InvalidOperationException: The association between entity types 'Blog' and 'Post' has
been severed but the foreign key for this relationship cannot be set to null. If the dependent entity should
be deleted, then setup the relationship to use cascade deletes.

```

- Blog jest oznaczony jako usunięty
- Wpisy początkowo pozostać Unchanged, ponieważ kaskady tak się nie stanie do momentu SaveChanges
- Ponieważ Ogranicz informuje EF, aby automatycznie ustawić klucza Obcego o wartości null, pozostaje inną niż null, i SaveChanges zgłasza bez zapisywania

Usuń porzucone przykłady

Poniższy kod jest częścią [przykładowe](#), można pobrać i uruchomić. Przykład pokazuje, co się stanie dla każdego zachowanie dotyczące usuwania dla relacji wymagane i opcjonalne po relacji między nadrzędnym/jednostki i jego elementy podrzędne/dependents jest oddzielone. W tym przykładzie relacji jest oddzielone przez usunięcie zależności/elementy podrzędne (ogłoszeń) z właściwości nawigacji kolekcji jednostki/nadrzędnej (blog). Jednak to zachowanie jest taka sama Jeśli odwołanie z zależnych od ustawień lokalnych/podrzędny do podmiotu

zabezpieczeń/element nadzędny jest zamiast tego pustych.

```
var blog = context.Blogs.Include(b => b.Posts).First();
var posts = blog.Posts.ToList();

DumpEntities(" After loading entities:", context, blog, posts);

blog.Posts.Clear();

DumpEntities(" After making posts orphans:", context, blog, posts);

try
{
    Console.WriteLine();
    Console.WriteLine(" Saving changes:");

    context.SaveChanges();

    DumpSql();

    DumpEntities(" After SaveChanges:", context, blog, posts);
}
catch (Exception e)
{
    DumpSql();

    Console.WriteLine();
    Console.WriteLine($" SaveChanges threw {e.GetType().Name}: {(e is DbUpdateException ? e.InnerException.Message : e.Message)}");
}
```

Przejdźmy teraz przez poszczególnych odmian, aby zrozumieć, co się dzieje.

DeleteBehavior.Cascade z relacją wymagane lub opcjonalne

```
After loading entities:
Blog '1' is in state Unchanged with 2 posts referenced.
Post '1' is in state Unchanged with FK '1' and reference to blog '1'.
Post '2' is in state Unchanged with FK '1' and reference to blog '1'.

After making posts orphans:
Blog '1' is in state Unchanged with 2 posts referenced.
Post '1' is in state Modified with FK '1' and no reference to a blog.
Post '2' is in state Modified with FK '1' and no reference to a blog.

Saving changes:
DELETE FROM [Posts] WHERE [PostId] = 1
DELETE FROM [Posts] WHERE [PostId] = 2

After SaveChanges:
Blog '1' is in state Unchanged with 2 posts referenced.
Post '1' is in state Detached with FK '1' and no reference to a blog.
Post '2' is in state Detached with FK '1' and no reference to a blog.
```

- Wpisy są oznaczane jako zmodyfikowane, ponieważ severing relacji klucza Obcego być oznaczony jako wartość null
 - W przypadku klucza Obcego nie dopuszcza wartości null, następnie wartość rzeczywista nie zmieni, nawet jeśli nie jest oznaczona jako wartości null
- SaveChanges wysyła usuwa dla zależności/dzieci (wpisów)
- Po zapisaniu zależności/elementy podzielone (ogłoszeń) są odłączone od teraz zostały usunięte z bazy danych

DeleteBehavior.ClientSetNull lub DeleteBehavior.SetNull z wymaganą relacją

After loading entities:

```
Blog '1' is in state Unchanged with 2 posts referenced.  
Post '1' is in state Unchanged with FK '1' and reference to blog '1'.  
Post '2' is in state Unchanged with FK '1' and reference to blog '1'.
```

After making posts orphans:

```
Blog '1' is in state Unchanged with 2 posts referenced.  
Post '1' is in state Modified with FK 'null' and no reference to a blog.  
Post '2' is in state Modified with FK 'null' and no reference to a blog.
```

Saving changes:

```
UPDATE [Posts] SET [BlogId] = NULL WHERE [PostId] = 1
```

```
SaveChanges threw DbUpdateException: Cannot insert the value NULL into column 'BlogId', table  
'EFSaving.CascadeDelete.dbo.Posts'; column does not allow nulls. UPDATE fails. The statement has been  
terminated.
```

- Wpisy są oznaczane jako zmodyfikowane, ponieważ severing relacji klucza Obcego być oznaczony jako wartość null
 - W przypadku klucza Obcego nie dopuszcza wartości null, następnie wartość rzeczywista nie zmieni, nawet jeśli nie jest oznaczona jako wartości null
- SaveChanges podejmuje próbę ustawienia wpisu klucza Obcego na wartość null, ale to nie powiodło się ponieważ klucz Obcego nie dopuszcza wartości null

DeleteBehavior.ClientSetNull lub DeleteBehavior.SetNull z opcjonalną relacją

After loading entities:

```
Blog '1' is in state Unchanged with 2 posts referenced.  
Post '1' is in state Unchanged with FK '1' and reference to blog '1'.  
Post '2' is in state Unchanged with FK '1' and reference to blog '1'.
```

After making posts orphans:

```
Blog '1' is in state Unchanged with 2 posts referenced.  
Post '1' is in state Modified with FK 'null' and no reference to a blog.  
Post '2' is in state Modified with FK 'null' and no reference to a blog.
```

Saving changes:

```
UPDATE [Posts] SET [BlogId] = NULL WHERE [PostId] = 1  
UPDATE [Posts] SET [BlogId] = NULL WHERE [PostId] = 2
```

After SaveChanges:

```
Blog '1' is in state Unchanged with 2 posts referenced.  
Post '1' is in state Unchanged with FK 'null' and no reference to a blog.  
Post '2' is in state Unchanged with FK 'null' and no reference to a blog.
```

- Wpisy są oznaczane jako zmodyfikowane, ponieważ severing relacji klucza Obcego być oznaczony jako wartość null
 - W przypadku klucza Obcego nie dopuszcza wartości null, następnie wartość rzeczywista nie zmieni, nawet jeśli nie jest oznaczona jako wartości null
- SaveChanges ustawia klucz Obcego z obu zależności/elementów podrzędnych (ogłoszeń) o wartości null
- Po zapisaniu zależności/elementy podrzędne (ogłoszeń) ma wartości null klucza Obcego, a ich odwołania do jednostki/nadrzędnych usunięte (blog) została usunięta.

DeleteBehavior.Restrict z relacją wymagane lub opcjonalne

```
After loading entities:
```

```
Blog '1' is in state Unchanged with 2 posts referenced.  
Post '1' is in state Unchanged with FK '1' and reference to blog '1'.  
Post '2' is in state Unchanged with FK '1' and reference to blog '1'.
```

```
After making posts orphans:
```

```
Blog '1' is in state Unchanged with 2 posts referenced.  
Post '1' is in state Modified with FK '1' and no reference to a blog.  
Post '2' is in state Modified with FK '1' and no reference to a blog.
```

```
Saving changes:
```

```
SaveChanges threw InvalidOperationException: The association between entity types 'Blog' and 'Post' has  
been severed but the foreign key for this relationship cannot be set to null. If the dependent entity should  
be deleted, then setup the relationship to use cascade deletes.
```

- Wpisy są oznaczane jako zmodyfikowane, ponieważ severing relacji klucza Obcego być oznaczony jako wartość null
 - W przypadku klucza Obcego nie dopuszcza wartości null, następnie wartość rzeczywista nie zmieni, nawet jeśli nie jest oznaczona jako wartości null
- Ponieważ *Ogranicz* informuje EF, aby automatycznie ustawić klucza Obcego o wartości null, pozostaje inną niż null, i SaveChanges zgłasza bez zapisywania

Kaskadowe nieśledzone jednostek

Gdy wywołujesz *SaveChanges*, reguły usuwanie kaskadowe zostaną zastosowane do dowolnej jednostki, które są śledzone przez kontekst. Jest to sytuacja we wszystkich przykładach pokazano powyżej, co jest Dlaczego SQL został wygenerowany można usunąć jednostki/element nadzędny (blog) i wszystkie zależności/elementy podrzędne (ogłoszeń):

```
DELETE FROM [Posts] WHERE [PostId] = 1  
DELETE FROM [Posts] WHERE [PostId] = 2  
DELETE FROM [Blogs] WHERE [BlogId] = 1
```

Jeśli tylko podmiot zabezpieczeń jest ładowany — na przykład, gdy zapytania są tworzone dla bloga bez `Include(b => b.Posts)` też dołączyć wpisy — następnie SaveChanges będą generowane tylko SQL, aby usunąć jednostkę/element nadzędny:

```
DELETE FROM [Blogs] WHERE [BlogId] = 1
```

Zależności/elementy podrzędne (ogłoszeń) tylko zostaną usunięte, jeśli baza danych ma odpowiedni zachowania kaskadowe skonfigurowane. Jeśli używasz programu EF utworzyć bazę danych, zachowania kaskadowe będzie instalatora dla Ciebie.

Obsługa konfliktów współbieżności

28.08.2018 • 6 minutes to read • [Edit Online](#)

NOTE

Ta strona dokumentów, jak współbieżność działa w programie EF Core i sposób obsługi konfliktów współbieżności w aplikacji. Zobacz [tokeny współbieżności](#) Aby uzyskać szczegółowe informacje na temat konfigurowania tokeny współbieżności w modelu.

TIP

[Przykład](#) użyty w tym artykule można zobaczyć w witrynie GitHub.

Baza danych współbieżności odnosi się do sytuacji, w których wiele procesów lub użytkownikom dostęp lub zmienić te same dane w bazie danych, w tym samym czasie. *Kontrola współbieżności* odwołuje się do określonych mechanizmów używane w celu zapewnienia spójności danych w obecności równoczesnych zmian.

Implementuje programu EF Core *mechanizmu kontroli optymistycznej współbieżności*, co oznacza, że umożliwia ona wiele procesów lub użytkownicy wprowadzać zmiany, niezależnie od siebie bez konieczności synchronizacji lub blokowania. W sytuacji idealnej te zmiany nie kolidują ze sobą i w związku z tym będzie możliwa pomyślnie. W najgorszym przypadku wielkości liter dwa lub więcej procesów podejmie próbę zmian powodując konflikt, i tylko jeden z nich ma być pomyślnie wykonane.

Jak działa Kontrola współbieżności w programie EF Core

Właściwości skonfigurowane tokeny współbieżności są używane do wdrożenia mechanizmu kontroli optymistycznej współbieżności: zawsze, gdy operacji update lub delete jest wykonywane podczas `SaveChanges`, wartość tokenu współbieżności w bazie danych jest porównywana oryginał wartości odczytu przez platformę EF Core.

- Jeśli wartości są zgodne, można wykonać operacji.
- Jeśli wartości nie są zgodne, EF Core założono, że inny użytkownik wykonał operacji powodującej konflikt i przerwia bieżącej transakcji.

Sytuacja, gdy inny użytkownik wykonał operację, która powoduje konflikt z bieżącej operacji jest znany jako *konfliktów współbieżności*.

Dostawcy baz danych są odpowiedzialni za wdrażanie porównanie wartości tokenu współbieżności.

W relacyjnych baz danych programu EF Core obejmuje sprawdzenia wartości tokenu współbieżności w `WHERE` klauzuli dowolnego `UPDATE` lub `DELETE` instrukcji. Po wykonaniu instrukcji, programem EF Core odczytuje liczbę wierszy, które miały wpływ.

Jeśli wpływają żadne wiersze nie został wykryty konflikt współbieżności i zgłasza wyjątek programu EF Core `DbUpdateConcurrencyException`.

Na przykład firma Microsoft może być konieczne skonfigurowanie `LastName` na `Person` za tokenem współbieżności. Następnie żadnych operacji aktualizacji w osoba będzie zawierać wybór współbieżność w `WHERE` klauzuli:

```
UPDATE [Person] SET [FirstName] = @p1  
WHERE [PersonId] = @p0 AND [LastName] = @p2;
```

Rozwiązywanie konfliktów współbieżności

Kontynuując poprzedni przykład gdy jeden użytkownik próbuje zapisać pewne zmiany `Person`, ale już inny użytkownik zmienił `LastName`, a następnie zostanie zgłoszony wyjątek.

W tym momencie aplikacja może po prostu informować użytkownika, że aktualizacja zakończyła się niepowodzeniem z powodu zmian powodujących konflikt i przejść. Jednak może być pożądane, aby monitować użytkownika, upewnić się, że ten rekord reprezentuje nadal tę samą osobę rzeczywiste i spróbuj ponownie wykonać operację.

Ten proces jest przykładem *Rozwiązywanie konfliktów współbieżności*.

Rozwiązywanie konfliktów współbieżności obejmuje scalanie oczekujące zmiany z bieżącą `DbContext` z wartościami w bazie danych. Scalony jakie wartości będą się różnić w zależności od aplikacji i zaleceniami dane wejściowymi użytkownika.

Istnieją trzy rodzaje wartości które mogą pomóc rozwiązać konflikt współbieżności:

- **Bieżące wartości** są wartościami, które aplikacja próbuje zapisać w bazie danych.
- **Oryginalne wartości** są wartościami, które zostały pierwotnie pobrane z bazy danych, zanim zmiany zostały wprowadzone.
- **Bazy danych wartości** wartości przechowywane w bazie danych.

Obsługa konfliktów współbieżności ogólne podejście jest:

1. CATCH `DbUpdateConcurrencyException` podczas `SaveChanges`.
2. Użyj `DbUpdateConcurrencyException.Entries` Aby przygotować nowy zestaw zmian dla obiektów, których to dotyczy.
3. Odśwież oryginalne wartości parametru tokenu współbieżności do bieżącej wartości w bazie danych.
4. Ponów próbę wykonania procesu, dopóki nie występują żadne konflikty.

W poniższym przykładzie `Person.FirstName` i `Person.LastName` są skonfigurowane jako tokeny współbieżności. Brak `// TODO:` komentarz w lokalizacji, gdzie obejmują określonej logiki aplikacji, aby wybrać wartość do zapisania.

```

using (var context = new PersonContext())
{
    // Fetch a person from database and change phone number
    var person = context.People.Single(p => p.PersonId == 1);
    person.PhoneNumber = "555-555-5555";

    // Change the person's name in the database to simulate a concurrency conflict
    context.Database.ExecuteSqlCommand(
        "UPDATE dbo.People SET FirstName = 'Jane' WHERE PersonId = 1");

    var saved = false;
    while (!saved)
    {
        try
        {
            // Attempt to save changes to the database
            context.SaveChanges();
            saved = true;
        }
        catch (DbUpdateConcurrencyException ex)
        {
            foreach (var entry in ex.Entries)
            {
                if (entry.Entity is Person)
                {
                    var proposedValues = entry.CurrentValues;
                    var databaseValues = entry.GetDatabaseValues();

                    foreach (var property in proposedValues.Properties)
                    {
                        var proposedValue = proposedValues[property];
                        var databaseValue = databaseValues[property];

                        // TODO: decide which value should be written to database
                        // proposedValues[property] = <value to be saved>;
                    }

                    // Refresh original values to bypass next concurrency check
                    entry.OriginalValues.SetValues(databaseValues);
                }
                else
                {
                    throw new NotSupportedException(
                        "Don't know how to handle concurrency conflicts for "
                        + entry.Metadata.Name);
                }
            }
        }
    }
}

```

Za pomocą transakcji

12.01.2019 • 8 minutes to read • [Edit Online](#)

Transakcje pozwalają na wykonanie kilku operacji na bazie danych w sposób niepodzielny. Jeżeli transakcja zostanie zatwierdzona, wszystkie operacje zostaną pomyślnie wykonane na bazie danych. Jeżeli transakcja zostanie wycofana, żadna z operacji nie zostanie wykonana na bazie danych.

TIP

Przykład użyty w tym artykule można zobaczyć w witrynie GitHub.

Domyślne zachowanie transakcji

Domyślnie, jeśli dostawca bazy danych obsługuje transakcje, wszystkie zmiany w pojedynczym wywołaniu operacji `SaveChanges()` są stosowane w transakcji. Jeżeli jakakolwiek z tych zmian nie powiedzie się, transakcja zostanie wycofana i zmiany nie zostaną zapisane. Gwarantuje to, że operacja `SaveChanges()` albo powiedzie się całkowicie, albo spowoduje pozostawienie niezmodyfikowanej bazy danych w przypadku błędu.

W przypadku większości zastosowań to domyślne zachowanie jest wystarczające. Transakcje powinny być sterowane ręcznie, tylko jeżeli taka konieczność wynika z wymagań danego zastosowania.

Kontrolowanie transakcji

Transakcje można rozpoczynać, zatwierdzać i wycofywać w interfejsie API `DbContext.Database`. Poniższy przykład przedstawia dwie operacje `SaveChanges()` i zapytanie LINQ wykonane w ramach jednej transakcji.

Nie wszyscy dostawcy bazy danych obsługują transakcje. Niektórzy dostawcy mogą zgłaszać wyjątki lub nie wykonywać żadnych operacji po wywołaniu interfejsu API transakcji.

```

using (var context = new BloggingContext())
{
    using (var transaction = context.Database.BeginTransaction())
    {
        try
        {
            context.Blogs.Add(new Blog { Url = "http://blogs.msdn.com/dotnet" });
            context.SaveChanges();

            context.Blogs.Add(new Blog { Url = "http://blogs.msdn.com/visualstudio" });
            context.SaveChanges();

            var blogs = context.Blogs
                .OrderBy(b => b.Url)
                .ToList();

            // Commit transaction if all commands succeed, transaction will auto-rollback
            // when disposed if either commands fails
            transaction.Commit();
        }
        catch (Exception)
        {
            // TODO: Handle failure
        }
    }
}

```

Transakcja międzykontekstowa (tylko relacyjne bazy danych)

Transakcje mogą być również współdzielone przez wiele wystąpień kontekstów. Ta funkcja jest dostępna tylko w przypadku korzystania z dostawcy relacyjnej bazy danych, ponieważ wymaga użycia parametrów `DbTransaction` i `DbConnection`, które są specyficzne dla relacyjnych baz danych.

Aby współdzielić transakcję, konteksty muszą współdzielić zarówno parametr `DbConnection`, jak i `DbTransaction`.

Zezwalanie na dostarczanie połączenia z zewnątrz

Współdzielenie parametru `DbConnection` wymaga, aby była możliwość przekazania połączenia do kontekstu podczas jego tworzenia.

Najprostszym sposobem na to, aby dostarczanie parametru `DbConnection` było możliwe z zewnątrz, jest zrezygnowanie z używania metody `DbContext.OnConfiguring` do konfigurowania kontekstu i utworzenie opcji `DbContextOptions` na zewnątrz, a następnie przekazanie ich do konstruktora kontekstu.

TIP

Parametr `DbContextOptionsBuilder` jest interfejsem API używanym w parametrze `DbContext.OnConfiguring` do konfigurowania kontekstu. Zostanie on teraz użyty zewnętrznie do utworzenia parametru `DbContextOptions`.

```

public class BloggingContext : DbContext
{
    public BloggingContext(DbContextOptions<BloggingContext> options)
        : base(options)
    { }

    public DbSet<Blog> Blogs { get; set; }
}

```

Alternatywą jest dalsze używanie parametru `DbContext.OnConfiguring`, ale akceptowanie parametru `DbConnection`,

który jest zapisywany i następnie używany w parametrze `DbContext.OnConfiguring`.

```
public class BloggingContext : DbContext
{
    private DbConnection _connection;

    public BloggingContext(DbConnection connection)
    {
        _connection = connection;
    }

    public DbSet<Blog> Blogs { get; set; }

    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    {
        optionsBuilder.UseSqlServer(_connection);
    }
}
```

Współdzielenie połączenia i transakcji

Teraz możesz utworzyć wiele wystąpień kontekstu współużytkujących to samo połączenie. Następnie użyj interfejsu API `DbContext.Database.UseTransaction(DbTransaction)` do zarejestrowania wielu kontekstów do jednej transakcji.

```
var options = new DbContextOptionsBuilder<BloggingContext>()
    .UseSqlServer(new SqlConnection(connectionString))
    .Options;

using (var context1 = new BloggingContext(options))
{
    using (var transaction = context1.Database.BeginTransaction())
    {
        try
        {
            context1.Blogs.Add(new Blog { Url = "http://blogs.msdn.com/dotnet" });
            context1.SaveChanges();

            using (var context2 = new BloggingContext(options))
            {
                context2.Database.UseTransaction(transaction.GetDbTransaction());

                var blogs = context2.Blogs
                    .OrderBy(b => b.Url)
                    .ToList();
            }
        }

        // Commit transaction if all commands succeed, transaction will auto-rollback
        // when disposed if either commands fails
        transaction.Commit();
    }
    catch (Exception)
    {
        // TODO: Handle failure
    }
}
```

Korzystanie z zewnętrznych transakcji `DbTransaction` (tylko relacyjne bazy danych)

Korzystając z wielu metod dostępu do danych w relacyjnej bazie danych, można współdzielić transakcje z operacjami wykonywanymi przez te różne metody.

Poniższy przykład przedstawia sposób wykonywania operacji ADO.NET SqlClient i Entity Framework Core w tej samej transakcji.

```
using (var connection = new SqlConnection(connectionString))
{
    connection.Open();

    using (var transaction = connection.BeginTransaction())
    {
        try
        {
            // Run raw ADO.NET command in the transaction
            var command = connection.CreateCommand();
            command.Transaction = transaction;
            command.CommandText = "DELETE FROM dbo.Blogs";
            command.ExecuteNonQuery();

            // Run an EF Core command in the transaction
            var options = new DbContextOptionsBuilder<BloggingContext>()
                .UseSqlServer(connection)
                .Options;

            using (var context = new BloggingContext(options))
            {
                context.Database.UseTransaction(transaction);
                context.Blogs.Add(new Blog { Url = "http://blogs.msdn.com/dotnet" });
                context.SaveChanges();
            }

            // Commit transaction if all commands succeed, transaction will auto-rollback
            // when disposed if either commands fails
            transaction.Commit();
        }
        catch (System.Exception)
        {
            // TODO: Handle failure
        }
    }
}
```

Używanie System.Transactions

NOTE

Ta funkcja jest nowa na platformie EF Core 2.1.

Istnieje możliwość użycia transakcji otoczenia, aby umożliwić koordynację na większą skalę.

```
using (var scope = new TransactionScope(
    TransactionScopeOption.Required,
    new TransactionOptions { IsolationLevel = IsolationLevel.ReadCommitted }))
{
    using (var connection = new SqlConnection(connectionString))
    {
        connection.Open();

        try
        {
            // Run raw ADO.NET command in the transaction
            var command = connection.CreateCommand();
            command.CommandText = "DELETE FROM dbo.Blogs";
            command.ExecuteNonQuery();

            // Run an EF Core command in the transaction
            var options = new DbContextOptionsBuilder<BloggingContext>()
                .UseSqlServer(connection)
                .Options;

            using (var context = new BloggingContext(options))
            {
                context.Blogs.Add(new Blog { Url = "http://blogs.msdn.com/dotnet" });
                context.SaveChanges();
            }

            // Commit transaction if all commands succeed, transaction will auto-rollback
            // when disposed if either commands fails
            scope.Complete();
        }
        catch (System.Exception)
        {
            // TODO: Handle failure
        }
    }
}
```

Istnieje również możliwość zarejestrowania w transakcji jawnej.

```

using (var transaction = new CommittableTransaction(
    new TransactionOptions { IsolationLevel = IsolationLevel.ReadCommitted }))
{
    var connection = new SqlConnection(connectionString);

    try
    {
        var options = new DbContextOptionsBuilder<BloggingContext>()
            .UseSqlServer(connection)
            .Options;

        using (var context = new BloggingContext(options))
        {
            context.Database.OpenConnection();
            context.Database.EnlistTransaction(transaction);

            // Run raw ADO.NET command in the transaction
            var command = connection.CreateCommand();
            command.CommandText = "DELETE FROM dbo.Blogs";
            command.ExecuteNonQuery();

            // Run an EF Core command in the transaction
            context.Blogs.Add(new Blog { Url = "http://blogs.msdn.com/dotnet" });
            context.SaveChanges();
            context.Database.CloseConnection();
        }

        // Commit transaction if all commands succeed, transaction will auto-rollback
        // when disposed if either commands fails
        transaction.Commit();
    }
    catch (System.Exception)
    {
        // TODO: Handle failure
    }
}

```

Ograniczenia przestrzeni nazw System.Transactions

1. Platforma EF Core korzysta z dostawców baz danych do implementowania obsługi przestrzeni nazw `System.Transactions`. Mimo że dostawcy ADO.NET często obsługują platformę .NET Framework, ten interfejs API został dopiero niedawno dodany do platformy EF Core. Jeżeli dostawca nie implementuje obsługi przestrzeni nazw `System.Transactions`, możliwe jest, że wywołania do tego interfejsu API zostaną zignorowane. `SqlClient` dla platformy .NET Core obsługuje je od wersji 2.1 w góre. Klient SQL dla platformy .NET Core 2.0 spowoduje zgłoszenie wyjątku. Jeśli spróbujesz użyć funkcji.

IMPORTANT

Zaleca się przetestowanie, czy ten interfejs API działa poprawnie z Twoim dostawcą, zanim skorzystasz z niego do zarządzania transakcjami. W przypadku problemów zachęcamy do kontaktu z osobami obsługującymi danego dostawcę bazy danych.

2. Począwszy od wersji 2.1 implementacja przestrzeni nazw `System.Transactions` na platformie .NET Core nie obsługuje transakcji rozproszonych, dlatego nie można używać parametrów `TransactionScope` lub `CommittableTransaction` do koordynowania transakcji w wielu menedżerach zasobów.

Zapisywanie asynchroniczne

28.08.2018 • 2 minutes to read • [Edit Online](#)

Zapisywanie asynchroniczne unika blokowania wątku, a zmiany są zapisywane w bazie danych. Może to być przydatne w celu uniknięcia zawieszenia się interfejsu użytkownika aplikacji klienckiej grubości. Operacje asynchroniczne także może zwiększyć przepływność w aplikacji sieci web, w którym wątku można zwolnić, do obsługi innych żądań podczas końca operacji bazy danych. Aby uzyskać więcej informacji, zobacz [asynchronicznego programowania w języku C#](#).

WARNING

EF Core nie obsługuje wielu operacji równoległych, które są uruchamiane na tym samym wystąpieniu kontekstu. Należy zawsze poczekać na zakończenie operacji przed rozpoczęciem następnej operacji. Zazwyczaj jest to wykonywane przy użyciu `await` słowo kluczowe w każdej operacji asynchronicznej.

Udostępnia platformy Entity Framework Core `DbContext.SaveChangesAsync()` asynchronicznego zamiast

`DbContext.SaveChanges()`.

```
public static async Task AddBlogAsync(string url)
{
    using (var context = new BloggingContext())
    {
        var blog = new Blog { Url = url };
        context.Blogs.Add(blog);
        await context.SaveChangesAsync();
    }
}
```

Odłączone jednostki

28.08.2018 • 13 minutes to read • [Edit Online](#)

Wystąpienie typu DbContext będzie automatycznie śledzić obiektów zwracanych z bazy danych. Zmiany wprowadzone do tych jednostek zostanie wykryty, gdy SaveChanges nosi nazwę bazy danych zostaną zaktualizowane zgodnie z potrzebami. Zobacz [podstawowe Zapisz i powiązanych danych](#) Aby uzyskać szczegółowe informacje.

Jednak czasami jednostki są wysyłane zapytanie przy użyciu jednego wystąpienia kontekstu i następnie zapisywane przy użyciu innego wystąpienia. To często zdarza się w scenariuszach "odłączonego", takich jak aplikacja sieci web, gdzie jednostki są badane, wysłane do klienta, modyfikowana, wysyłanych z powrotem do serwera w żądaniu i następnie zapisany. W tym przypadku kontekstu drugiego wystąpienia musi wiedzieć, czy jednostki są nowe (powinien zostać wstawiony) lub istniejące (powinien zostać zaktualizowany).

TIP

[Przykład](#) użyty w tym artykule można zobaczyć w witrynie GitHub.

TIP

EF Core można śledzić tylko jedno wystąpienie jednostki z danej wartości klucza podstawowego. Najlepszym sposobem, aby uniknąć tego problemu jest do użycia krótkotrwałe kontekstu dla każdej jednostki pracy w taki sposób, że kontekst zaczyna się puste, zawiera jednostki podłączone do niego, zapisuje te jednostki oraz kontekst jest usunięty i odrzucone.

Identyfikowanie nowych jednostek

Klient identyfikuje nowe jednostki

Najprostszym przypadku radzenia sobie z jest, gdy klient informuje serwer, czy jednostka jest nowym lub istniejącym. Na przykład często żądania, aby wstawić nowy obiekt różni się od żądanie zaktualizowania istniejącej jednostki.

Dalszej części tej sekcji omówiono przypadki gdzie go, które są niezbędne określić w inny sposób, czy należy wstawić lub zaktualizować.

Za pomocą automatycznego generowania kluczy

Wartość automatycznie generowanego klucza często może służyć do określenia, czy jednostka musi być wstawiane lub aktualizowane. Jeśli nie ustalono klucza (oznacza to, że nadal ma wartość domyślną CLR o wartości null, wartość zero, itp.), a jednostka musi być nowe musi wstawiana. Z drugiej strony Jeśli ustalono wartość klucza, następnie go musi już zostały wcześniej zapisane i teraz wymaga aktualizacji. Innymi słowy Jeśli klucz ma wartość, a następnie jednostki badano wysłane do klienta i ma teraz wróć do zaktualizowania.

Jest łatwy do sprawdzenia nie ustalono klucza, gdy znana jest typem jednostki:

```
public static bool IsItNew(Blog blog)
=> blog.BlogId == 0;
```

EF ma również wbudowane sposobem wykonania tego typu jednostki i typ klucza:

```
public static bool IsItNew(DbContext context, object entity)
=> !context.Entry(entity).IsKeySet;
```

TIP

Klucze są ustawione, tak szybko, jak jednostki są śledzone od kontekstu, nawet jeśli jednostka jest w stanie dodany. Dzięki temu podczas przechodzenia między wykres jednostek i podejmowania decyzji o postępowaniu z każdego, na przykład, jak za pomocą interfejsu API TrackGraph. Wartość klucza powinna służyć wyłącznie w taki sposób, w tym miejscu pokazano *przed wszelkie rozmowy do śledzenia jednostki.*

Przy użyciu innych kluczy

Niektóre inny mechanizm jest potrzebny do identyfikowania nowych jednostek, gdy wartości klucza nie są generowane automatycznie. Istnieją dwa ogólne podejścia do tego:

- Zapytanie dla jednostki
- Przekazać flagę od klienta

Aby wysłać zapytanie do jednostki, po prostu użyj metody Find:

```
public static bool IsItNew(BloggingContext context, Blog blog)
=> context.Blogs.Find(blog.BlogId) == null;
```

Jest poza zakres tego dokumentu, aby wyświetlić pełny kod do przekazywania flagę od klienta. W aplikacji sieci web zwykle oznacza to, co innych żądań dla rozmaitych akcji lub przekazywanie pewnego stanu w żądaniu, a następnie wyodrębniania go w kontrolerze.

Zapisywanie pojedynczych jednostek

Jeśli wiadomo, czy jest potrzebny insert nebo update, a następnie dodaj lub zaktualizuj można odpowiednio:

```
public static void Insert(DbContext context, object entity)
{
    context.Add(entity);
    context.SaveChanges();
}

public static void Update(DbContext context, object entity)
{
    context.Update(entity);
    context.SaveChanges();
}
```

Jednak jeśli jednostki używa automatycznego generowania wartości klucza, następnie metoda aktualizacji może służyć w obu przypadkach:

```
public static void InsertOrUpdate(DbContext context, object entity)
{
    context.Update(entity);
    context.SaveChanges();
}
```

Metoda aktualizacji zazwyczaj oznacza jednostkę do aktualizacji, wstawiania nie. Jednakże jeśli jednostka ma klucz wygenerowany automatycznie, a nie wartość klucza została ustawiona, a następnie jednostki zamiast tego jest automatycznie oznaczony do wstawienia.

TIP

To zachowanie została wprowadzona w programie EF Core 2.0. Dla wcześniejszych wersji należy zawsze jawnie wybrać Dodawanie lub aktualizowanie.

Jeśli jednostka nie korzysta z automatycznego generowania kluczy, a następnie aplikacja musi zdecydować, czy wstawione lub zaktualizowane jednostki: na przykład:

```
public static void InsertOrUpdate(BloggingContext context, Blog blog)
{
    var existingBlog = context.Blogs.Find(blog.BlogId);
    if (existingBlog == null)
    {
        context.Add(blog);
    }
    else
    {
        context.Entry(existingBlog).CurrentValues.SetValues(blog);
    }

    context.SaveChanges();
}
```

Dostępne są następujące kroki:

- Jeśli dodasz Znajdź zwraca wartość null, a następnie baza danych nie zawiera jeszcze blog o tym identyfikatorze, dlatego nazywamy Oznacz ją do wstawienia.
- Jeśli wyszukiwanie zwraca jednostkę, następnie istnieje w bazie danych i kontekstu jest teraz śledzenie istniejącej jednostki
 - Następnie używamy SetValues można ustawić wartości dla wszystkich właściwości dla tej jednostki do tych, które pochodzą od klienta.
 - Wywołanie SetValues spowoduje oznaczenie jednostkę którą chcesz zaktualizować, zgodnie z potrzebami.

TIP

SetValues oznaczy tylko zmienione właściwości, które mają różne wartości do tych w jednostce śledzone. Oznacza to, że gdy aktualizacja jest wysyłana, tylko te kolumny, które rzeczywiście zostały zmienione zostaną zaktualizowane. (I jeśli nic się nie zmieniło, aktualizacja nie zostanie wysłana w ogóle)

Praca z wykresami

Rozwiążanie tożsamości

Jak wspomniano powyżej, programem EF Core tylko śledzić jednego wystąpienia z danej wartości klucza podstawowego jednostki. Pracując z wykresami wykres najlepiej powinny być tworzone, tak, aby ta niezmienną jest utrzymywany i kontekst powinien być używany dla tylko jednej jednostki pracy. Jeśli wykres zawiera duplikaty, następnie będzie konieczne do przetworzenia na wykresie przed wysłaniem ich do programu EF w celu skonsolidowania wielu wystąpień w jednym. To może nie być prosta których wystąpienia mają wartościami będącymi w konflikcie i relacje, dlatego konsolidację duplikaty powinno się odbywać szybko, jak to możliwe w potoku aplikacji, aby uniknąć konfliktów.

Wszystkie nowe/wszystkich istniejących jednostek.

Przykładem korzystania z wykresów jest wstawianie lub aktualizowania blogu wraz z jego kolekcja skojarzone wpisów. Powinien zostać wstawiony wszystkich jednostek w wykresie lub wszystkie powinny być aktualizowane,

następnie proces jest taka sama, jak opisano powyżej dla jednej jednostki. Na przykład wykres blogów i wpisów umieszczonych w następujący sposób:

```
var blog = new Blog
{
    Url = "http://sample.com",
    Posts = new List<Post>
    {
        new Post {Title = "Post 1"},
        new Post {Title = "Post 2"},
    }
};
```

mogą być wstawiane w następujący sposób:

```
public static void InsertGraph(DbContext context, object rootEntity)
{
    context.Add(rootEntity);
    context.SaveChanges();
}
```

Spowoduje to oznaczenie blogu i wszystkie wpisy, które ma zostać wstawiony wywołania do Add.

Podobnie jeśli wszystkie jednostki w grafie muszą zostać zaktualizowane, wówczas aktualizacji mogą być używane:

```
public static void UpdateGraph(DbContext context, object rootEntity)
{
    context.Update(rootEntity);
    context.SaveChanges();
}
```

Blog i wszystkie jego wpisy zostaną oznaczone do zaktualizowania.

Kombinacja nowych i istniejących jednostek

Za pomocą automatycznego generowania kluczy aktualizacja ponownie można zarówno dla operacji wstawienia i aktualizacji, nawet jeśli wykres zawiera różne jednostki, które wymagają, wstawianie i tych, które wymagają zaktualizowania:

```
public static void InsertOrUpdateGraph(DbContext context, object rootEntity)
{
    context.Update(rootEntity);
    context.SaveChanges();
}
```

Aktualizacja spowoduje oznaczenie dowolnej jednostki w wykresu, blogu lub wpis do wstawienia nie zainstalowane zestawu wartości klucza, podczas gdy inne jednostki są oznaczone do aktualizacji.

Jak wcześniej, bez korzystania z automatycznego generowania kluczy, zapytanie i jakieś operacje przetwarzania może być użyty:

```
public static void InsertOrUpdateGraph(BloggingContext context, Blog blog)
{
    var existingBlog = context.Blogs
        .Include(b => b.Posts)
        .FirstOrDefault(b => b.BlogId == blog.BlogId);

    if (existingBlog == null)
    {
        context.Add(blog);
    }
    else
    {
        context.Entry(existingBlog).CurrentValues.SetValues(blog);
        foreach (var post in blog.Posts)
        {
            var existingPost = existingBlog.Posts
                .FirstOrDefault(p => p.PostId == post.PostId);

            if (existingPost == null)
            {
                existingBlog.Posts.Add(post);
            }
            else
            {
                context.Entry(existingPost).CurrentValues.SetValues(post);
            }
        }
    }

    context.SaveChanges();
}
```

Obsługa usuwa

Delete mogą być trudne do obsługi, ponieważ często Brak jednostki oznacza, że go usunąć. Jednym ze sposobów, aby poradzić sobie z tym jest używać "usuwania nietrwałego" w taki sposób, że jednostka jest oznaczony jako usunięty zamiast rzeczywistości usuwane. Usuwa, a następnie staje się taka sama jak aktualizacje. Usuwanie nietrwałego może być implementowany w przy użyciu [zapytania filtry](#).

Usuwa wartość true, aby uzyskać wspólny wzorzec jest użycie rozszerzenia wzorca zapytania do wykonania, co to jest zasadniczo różnica wykresu Na przykład:

```

public static void InsertUpdateOrDeleteGraph(BloggingContext context, Blog blog)
{
    var existingBlog = context.Blogs
        .Include(b => b.Posts)
        .FirstOrDefault(b => b.BlogId == blog.BlogId);

    if (existingBlog == null)
    {
        context.Add(blog);
    }
    else
    {
        context.Entry(existingBlog).CurrentValues.SetValues(blog);
        foreach (var post in blog.Posts)
        {
            var existingPost = existingBlog.Posts
                .FirstOrDefault(p => p.PostId == post.PostId);

            if (existingPost == null)
            {
                existingBlog.Posts.Add(post);
            }
            else
            {
                context.Entry(existingPost).CurrentValues.SetValues(post);
            }
        }

        foreach (var post in existingBlog.Posts)
        {
            if (!blog.Posts.Any(p => p.PostId == post.PostId))
            {
                context.Remove(post);
            }
        }
    }

    context.SaveChanges();
}

```

TrackGraph

Wewnętrznie, Dodaj, Dołącz i aktualizacji za pomocą przechodzenie grafu przy podejmowaniu wprowadzone dla każdej jednostki do tego, czy jego powinien być oznaczony jako dodano (Aby wstawić), zmodyfikowany (w celu aktualizacji), Unchanged (NIC), lub usunięte (Aby usunąć). Ten mechanizm jest uwidaczniany za pomocą interfejsu API TrackGraph. Na przykład założymy, że klient wysyła z powrotem wykres jednostek ustawia niektóre flagi dla każdej jednostki wskazujący sposób obsługi. TrackGraph następnie może służyć do przetwarzania tej flagi:

```
public static void SaveAnnotatedGraph(DbContext context, object rootEntity)
{
    context.ChangeTracker.TrackGraph(
        rootEntity,
        n =>
    {
        var entity = (EntityBase)n.Entry.Entity;
        n.Entry.State = entity isNew
            ? EntityState.Added
            : entity.IsChanged
                ? EntityState.Modified
                : entity.IsDeleted
                    ? EntityState.Deleted
                    : EntityState.Unchanged;
    });
    context.SaveChanges();
}
```

Flagi są wyświetlane tylko w ramach jednostki dla uproszczenia w przykładzie. Zazwyczaj flagi powinien być częścią obiekt DTO lub inny stan, zawarty w żądaniu.

Ustawienie jawnie wartości dla wygenerowanych właściwości

28.08.2018 • 5 minutes to read • [Edit Online](#)

Wygenerowana właściwość jest właściwością, którego wartość jest generowana (albo przez EF lub bazy danych) podczas dodane lub zaktualizowane jednostki. Zobacz [wygenerowanych właściwości](#) Aby uzyskać więcej informacji.

Mogą wystąpić sytuacje, w której chcesz ustawić jawną wartość dla wygenerowanej właściwości zamiast plan wygenerowany.

TIP

[Przykład](#) użyty w tym artykule można zobaczyć w witrynie GitHub.

Model

Model używany w tym artykule zawiera pojedynczy `Employee` jednostki.

```
public class Employee
{
    public int EmployeeId { get; set; }
    public string Name { get; set; }
    public DateTime EmploymentStarted { get; set; }
    public int Salary { get; set; }
    public DateTime? LastPayRaise { get; set; }
}
```

Zapisywanie jawną wartość podczas Dodaj

`Employee.EmploymentStarted` Skonfigurowano właściwości do wartości generowane przez bazę danych do nowych jednostek (przy użyciu wartości domyślnej).

```
modelBuilder.Entity<Employee>()
    .Property(b => b.EmploymentStarted)
    .HasDefaultValueSql("CONVERT(date, GETDATE())");
```

Poniższy kod powoduje wstawienie dwóch pracowników do bazy danych.

- W pierwszym, zostanie przypisana żadna wartość, aby `Employee.EmploymentStarted`, dzięki czemu pozostałe ustawioną na wartość domyślną CLR dla `DateTime`.
- Z drugiej ustawimy mają jawnie wartości `1-Jan-2000`.

```

using (var context = new EmployeeContext())
{
    context.Employees.Add(new Employee { Name = "John Doe" });
    context.Employees.Add(new Employee { Name = "Jane Doe", EmploymentStarted = new DateTime(2000, 1, 1) });
    context.SaveChanges();

    foreach (var employee in context.Employees)
    {
        Console.WriteLine(employee.EmployeeId + ": " + employee.Name + ", " + employee.EmploymentStarted);
    }
}

```

Dane wyjściowe pokazują, że bazy danych wygenerowaną wartość dla pierwszego pracownika i użyto naszych jawną wartość dla drugiego.

```

1: John Doe, 1/26/2017 12:00:00 AM
2: Jane Doe, 1/1/2000 12:00:00 AM

```

Jawne wartości w kolumnach tożsamości serwera SQL

Zgodnie z Konwencją `Employee.EmployeeId` właściwość jest magazynem generowane `IDENTITY` kolumny.

W większości sytuacji to podejście pokazano powyżej będzie działać w przypadku właściwości klucza. Jednakże można wstawić jawną wartości do programu SQL Server `IDENTITY` kolumnę, musisz ręcznie włączyć `IDENTITY_INSERT` przed wywołaniem `SaveChanges()`.

NOTE

Mamy [zgłoszenie dotyczące funkcji](#) na naszej liście prac, aby to zrobić automatycznie przy użyciu dostawcy programu SQL Server.

```

using (var context = new EmployeeContext())
{
    context.Employees.Add(new Employee { EmployeeId = 100, Name = "John Doe" });
    context.Employees.Add(new Employee { EmployeeId = 101, Name = "Jane Doe" });

    context.Database.OpenConnection();
    try
    {
        context.Database.ExecuteSqlCommand("SET IDENTITY_INSERT dbo.Employees ON");
        context.SaveChanges();
        context.Database.ExecuteSqlCommand("SET IDENTITY_INSERT dbo.Employees OFF");
    }
    finally
    {
        context.Database.CloseConnection();
    }

    foreach (var employee in context.Employees)
    {
        Console.WriteLine(employee.EmployeeId + ": " + employee.Name);
    }
}

```

Dane wyjściowe pokazują, że podane identyfikatory zostały zapisane w bazie danych.

```
100: John Doe  
101: Jane Doe
```

Ustawianie jawną wartość podczas aktualizacji

`Employee.LastPayRaise` Skonfigurowano właściwości do wartości generowane przez bazę danych podczas aktualizacji.

```
modelBuilder.Entity<Employee>()  
    .Property(b => b.LastPayRaise)  
    .ValueGeneratedOnAddOrUpdate();  
  
modelBuilder.Entity<Employee>()  
    .Property(b => b.LastPayRaise)  
    .Metadata.AfterSaveBehavior = PropertySaveBehavior.Ignore;
```

NOTE

Domyślnie program EF Core spowoduje zgłoszenie wyjątku, jeśli zostanie podjęta próba zapisania jawną wartość dla właściwości, który jest skonfigurowany do wygenerowania podczas aktualizacji. Aby tego uniknąć, należy do listy rozwijanej na niższym poziomie metadanych interfejsu API i ustawić `AfterSaveBehavior` (jak pokazano powyżej).

NOTE

Zmiany w programie EF Core 2.0: w poprzednich wersjach zachowanie wtórnym Zapisz zostało kontrolowane za pośrednictwem `IsReadOnlyAfterSave` flagi. Ta flaga została zamieniona przestarzały parametr i zastąpione przez `AfterSaveBehavior`.

Istnieje również wyzwalacza w bazie danych do generowania wartości dla `LastPayRaise` kolumny podczas `UPDATE` operacji.

```

CREATE TRIGGER [dbo].[Employees_UPDATE] ON [dbo].[Employees]
AFTER UPDATE
AS
BEGIN
    SET NOCOUNT ON;

    IF ((SELECT TRIGGER_NESTLEVEL()) > 1) RETURN;

    IF UPDATE(Salary) AND NOT Update(LastPayRaise)
    BEGIN
        DECLARE @Id INT
        DECLARE @OldSalary INT
        DECLARE @NewSalary INT

        SELECT @Id = INSERTED.EmployeeId, @NewSalary = Salary
        FROM INSERTED

        SELECT @OldSalary = Salary
        FROM deleted

        IF @NewSalary > @OldSalary
        BEGIN
            UPDATE dbo.Employees
            SET LastPayRaise = CONVERT(date, GETDATE())
            WHERE EmployeeId = @Id
        END
    END
END

```

Poniższy kod powoduje zwiększenie wynagrodzenia dwóch pracowników w bazie danych.

- W pierwszym, zostanie przypisana żadna wartość, aby `Employee.LastPayRaise` właściwości, dzięki czemu pozostanie ustawiony na wartość null.
- Dla drugiego firma Microsoft ustawiono jawną wartość jeden tydzień temu (podniesienie płatność sięga wstecz).

```

using (var context = new EmployeeContext())
{
    var john = context.Employees.Single(e => e.Name == "John Doe");
    john.Salary = 200;

    var jane = context.Employees.Single(e => e.Name == "Jane Doe");
    jane.Salary = 200;
    jane.LastPayRaise = DateTime.Today.AddDays(-7);

    context.SaveChanges();

    foreach (var employee in context.Employees)
    {
        Console.WriteLine(employee.EmployeeId + ": " + employee.Name + ", " + employee.LastPayRaise);
    }
}

```

Dane wyjściowe pokazują, że bazy danych wygenerowaną wartość dla pierwszego pracownika i użyto naszych jawną wartość dla drugiego.

```

1: John Doe, 1/26/2017 12:00:00 AM
2: Jane Doe, 1/19/2017 12:00:00 AM

```

Implementacji platformy .NET obsługiwanych przez platformę EF Core

25.10.2018 • 3 minutes to read • [Edit Online](#)

Chcemy, aby programu EF Core była dostępna wszędzie można napisać kod .NET i nadal pracujemy do tego celu. Podczas obsługi programu EF Core w .NET Core i .NET Framework jest objęta zautomatyzowanego testowania i wiele aplikacji znane go używać pomyślnie, platformy Mono, Xamarin i platformy uniwersalnej systemu Windows mają pewne problemy.

Omówienie

W poniższej tabeli przedstawiono wskazówki dotyczące każdej implementacji .NET:

IMPLEMENTACJA PROGRAMU .NET	STAN	WYMAGANIA DOTYCZĄCE PROGRAMU EF CORE 1.X	WYMAGANIA DOTYCZĄCE PROGRAMU EF CORE 2.X ⁽¹⁾
.NET core (platformy ASP.NET Core, konsoli ^{itp.})	W pełni obsługiwane i zalecane	Zestaw SDK dla platformy .NET core 1.x	.NET Core SDK 2.x
.NET framework (WinForms, WPF, ASP.NET, konsoli ^{itp.})	W pełni obsługiwane i zalecane. Dostępne są też EF6 ⁽²⁾	.NET Framework 4.5.1	.NET Framework 4.6.1
Narzędzie mono i Xamarin	Trwająca ⁽³⁾	Narzędzie mono 4.6 Xamarin.iOS 10 Xamarin.Mac 3 Xamarin.Android 7	5.4 platformy mono Xamarin.iOS 10.14 Xamarin.Mac 3.8 Xamarin.Android 7.5
Universal Windows Platform	EF Core 2.0.1 zalecane ⁽⁴⁾	Pakiet 5.x .NET core, platformy UWP	.NET core, platformy UWP w wersji 6.x pakietu

⁽¹⁾ EF Core 2.0 jest przeznaczony dla i dlatego wymaga implementacji platformy .NET, które obsługują .NET Standard 2.0.

⁽²⁾ Zobacz [Porównanie programów EF Core i EF6](#) do wyboru odpowiedniej technologii.

⁽³⁾ Brak problemów i znane ograniczenia za pomocą platformy Xamarin, które mogą uniemożliwić niektóre aplikacje opracowane przy użyciu programu EF Core 2.0 z działa prawidłowo. Sprawdź listę [aktywne problemy](#) do rozwiązania problemu.

⁽⁴⁾ Zobacz [Universal Windows Platform](#) dalszej części tego artykułu.

Platforma uniwersalna systemu Windows

Wcześniejszych wersjach programu EF Core i .NET platformy uniwersalnej systemu Windows ma wiele problemów ze zgodnością, szczególnie w przypadku aplikacje skompilowane przy użyciu łańcucha narzędzi .NET Native. Nowa wersja .NET platformy UWP dodaje obsługę platformy .NET Standard 2.0 i zawiera natywne 2.0 platformy .NET, która naprawia większość problemów ze zgodnością wcześniej zgłoszone. EF Core 2.0.1 został przetestowany dokładniej za pomocą platformy uniwersalnej systemu Windows, ale nie zautomatyzowane testy.

Podczas korzystania z programu EF Core w platformy uniwersalnej systemu Windows:

- Aby zoptymalizować wydajność zapytań, należy unikać typów anonimowych w kwerendach LINQ. Wdrażanie aplikacji do sklepu z aplikacjami platformy uniwersalnej systemu Windows wymaga od aplikacji można skompilować przy użyciu platformy .NET Native. Zapytania z typami anonimowymi ma zmniejszenie wydajności platformy .NET Native.
- Aby zoptymalizować `SaveChanges()` wydajności i użycia `ChangeTrackingStrategy.ChangingAndChangedNotifications` i zaimplementować `INotifyPropertyChanged`, `INotifyPropertyChanging`, i interfejsu `INotifyCollectionChanged` w typy jednostek.

Zgłaszanie problemów

Dla dowolnej kombinacji nie działa zgodnie z oczekiwaniemi, zaleca się utworzenie nowych problemów w [narzędzie do śledzenia problemów programu EF Core](#). Specyficzne dla platformy Xamarin problemów Użyj narzędzia do śledzenia błędów dla [Xamarin.Android](#) lub [Xamarin.iOS](#).

Dostawcy baz danych

05.12.2018 • 7 minutes to read • [Edit Online](#)

Entity Framework Core mogą uzyskiwać dostęp w wielu różnych bazach danych za pomocą wtyczek bibliotek nazywanych dostawcami bazy danych.

Bieżącego dostawcy

IMPORTANT

EF Core dostawcy są tworzone za pomocą różnych źródeł. Nie wszyscy dostawcy są przechowywane jako część projektu programu Entity Framework Core. Podczas wybierania dostawcy, pamiętaj ocenić jakość, licencjonowanie, pomocy technicznej, itp., aby upewnić się, że spełniają one wymagania. Ponadto upewnij się, że przejrzenie dokumentacji poszczególnych dostawców wersji szczegółowych informacji dotyczących zgodności.

PAKET NUGET	OBSŁUGIWANYCH APARATÓW BAZY DANYCH	ELEMENT UTRZYMUJĄCY / DOSTAWCY	INFORMACJE O / WYMAGAŃ	PRZYDATNE LINKI
Microsoft.EntityFrameworkCore.SqlServer	SQL Server 2008 lub nowszy	Projekt programu EF Core (Microsoft)		Dokumentacja
Microsoft.EntityFrameworkCore.SQLite	Bazy danych SQLite 3.7 lub nowszy	Projekt programu EF Core (Microsoft)		Dokumentacja
Microsoft.EntityFrameworkCore.InMemory	Baza danych programu EF Core w pamięci	Projekt programu EF Core (Microsoft)	Tylko do celów testowych	Dokumentacja
Microsoft.EntityFrameworkCore.Cosmos	Interfejs API SQL usługi Azure Cosmos DB	Projekt programu EF Core (Microsoft)	Tylko wersja zapoznawcza	blog
Npgsql.EntityFrameworkCore.PostgreSQL	PostgreSQL	Zespół programistyczny Npgsql		Dokumentacja
Pomelo.EntityFrameworkCore.MySql	MySQL, MariaDB	Pomelo Foundation projektu		readme
Pomelo.EntityFrameworkCore.MyCat	MyCAT serwera	Pomelo Foundation projektu	Tylko wersję wstępna	readme
EntityFrameworkCore.SqlServerCompact40	SQL Server Compact 4.0	Erik Ejlskov Jensen	.NET Framework	wiki
EntityFrameworkCore.SqlServerCompact35	SQL Server Compact 3.5	Erik Ejlskov Jensen	.NET Framework	wiki

PAKET NUGET	OBSŁUGIWANYCH APARATÓW BAZY DANYCH	ELEMENT UTRZYMUJĄCY / DOSTAWCY	INFORMACJE O / WYMAGAŃ	PRZYDATNE LINKI
EntityFrameworkCore.Jet	Pliki programu Microsoft Access	Bubi	.NET Framework	readme
MySQL.Data.EntityFrameworkCore	MySQL	Projekt MySQL (Oracle)		Dokumentacja
FirebirdSql.EntityFrameworkCore.Firebird	Firebird 2.5 i 3.x	Jiří Činčura		Dokumentacja
EntityFrameworkCore.FirebirdSql	Firebird 2.5 i 3.x	Rafael Almeida		wiki
IBM.EntityFrameworkCore	Db2, Informix	IBM	Wersja Windows	blog
IBM.EntityFrameworkCore-Inx	Db2, Informix	IBM	Wersji systemu Linux	blog
IBM.EntityFrameworkCore-osx	Db2, Informix	IBM	wersja systemu macOS	blog
EntityFrameworkCore.OpenEdge	OpenEdge postępu	Alex Wiese		readme
Devart.Data.Oracle.EntityFrameworkCore	Oracle 9.2.0.4 lub nowszy	DevArt	Płatne	Dokumentacja
Devart.Data.PostgreSql.EntityFrameworkCore	PostgreSQL 8.0 lub nowszy	DevArt	Płatne	Dokumentacja
Devart.Data.SQLite.EntityFrameworkCore	3 bazy danych SQLite	DevArt	Płatne	Dokumentacja
Devart.Data.MySql.EntityFrameworkCore	MySQL 5 lub nowszy	DevArt	Płatne	Dokumentacja

Przyszłe dostawców

Usługa cosmos DB

Mają firma Microsoft opracowuje dostawcę programu EF Core dla interfejsu API SQL w usłudze Cosmos DB. Jest to pierwszy dostawca pełnej korzystający z dokumentów bazy danych, który możemy zostały utworzone, a informacje w tym ćwiczeniu będzie informować ulepszenia w projekcie przyszłych wersjach programu EF Core i prawdopodobnie innych dostawców nierelacyjnych. Podgląd jest dostępny w [galerii pakietów NuGet](#).

Oracle

Zespołu Oracle .NET ogłosiła, czy jest planowane wersji dostawcy firmy Microsoft dla platformy EF Core w trzecim kwartale 2018 r. Zobacz ich [instrukcję kierunek dla platformy .NET Core i Entity Framework Core](#) Aby uzyskać więcej informacji. Pytania dotyczące tego dostawcy, w tym do osi czasu w wersji należy kierować [witryny społeczności Oracle](#).

W międzyczasie tworzył zespół EF [dostawcy programu EF Core próbki dla baz danych Oracle](#). Celem projektu nie jest utworzyć dostawcę usługi programu EF Core należące do firmy Microsoft. Rozpoczęliśmy projekt, aby zidentyfikować luki w działaniu relacyjnych i podstawowej programu EF Core, którym Musieliszy adreś w celu lepszej obsługi bazy danych Oracle. Ponadto powinna pomagać w Oracle lub firmom szybkie rozpoczęcie pracy rozwoju dostawcy Oracle dla platformy EF Core.

Firma Microsoft będzie należał wziąć pod uwagę wkładów, które zwiększa w przykładowej implementacji. Firma Microsoft będzie również Witaj i Zachęć pracach społeczności, aby utworzyć dostawcę programu Oracle typu open source dla platformy EF Core, korzystając z przykładu jako punktu wyjścia.

Dodawanie dostawcy bazy danych do aplikacji

Większość dostawców bazy danych dla platformy EF Core są dystrybuowane jako pakiety NuGet. Oznacza to, mogą być instalowane za pomocą `dotnet` narzędzia w wierszu polecenia:

```
dotnet add package provider_package_name
```

Lub w programie Visual Studio przy użyciu konsoli Menedżera pakietów NuGet:

```
install-package provider_package_name
```

Po zainstalowaniu skonfiguruje dostawcy w Twojej `DbContext`, albo w `OnConfiguring` metody lub `AddDbContext` metody korzystania z kontenera iniekcji zależności. Na przykład następujący wiersz konfiguruje dostawcę programu SQL Server przy użyciu parametrów połączenia przekazany:

```
optionsBuilder.UseSqlServer(  
    "Server=(localdb)\mssqllocaldb;Database=MyDatabase;Trusted_Connection=True;");
```

Dostawcy baz danych można rozszerzyć programu EF Core, aby włączyć funkcje unikatowe dla konkretnych baz danych. Niektóre pojęcia są wspólne dla większości baz danych i są objęte podstawowe składniki programu EF Core. Takie pojęcia zawierają wyrażenia zapytania w LINQ, transakcji i śledzenie zmian obiektów, gdy są one załadowane z bazy danych. Niektóre pojęcia są specyficzne dla określonego dostawcy. Na przykład dostawca programu SQL Server umożliwia [skonfigurować tabele zoptymalizowane pod kątem pamięci](#) (funkcja specyficzna dla programu SQL Server). Inne pojęcia są specyficzne dla klasy dostawców. Na przykład utworzyć dostawcy programu EF Core dla relacyjnych baz danych na wspólnej `Microsoft.EntityFrameworkCore.Relational` bibliotece, która udostępnia interfejsy API do skonfigurowania mapowania tabel i kolumn, ograniczeń klucza obcego, itp. Dostawcy są zazwyczaj dystrybuowane jako pakiety NuGet.

IMPORTANT

Po wydaniu nowej wersji poprawki programu EF Core często obejmuje aktualizacje `Microsoft.EntityFrameworkCore.Relational` pakietu. Po dodaniu dostawcy relacyjnej bazy danych, ten pakiet uzależnienie przechodnie aplikacji. Jednak wielu dostawców są wydawane niezależnie od programu EF Core i nie mogą być aktualizowane są zależne od nowszej wersji tego pakietu poprawek. Aby upewnić się, otrzymasz wszystkie poprawki błędów, zaleca się dodawania wersję poprawki `Microsoft.EntityFrameworkCore.Relational` jako bezpośrednie zależności aplikacji.

Dostawca bazy danych programu Microsoft SQL Server EF Core

28.08.2018 • 2 minutes to read • [Edit Online](#)

Ten dostawca bazy danych umożliwia platformy Entity Framework Core do użycia z programem Microsoft SQL Server (w tym usługi SQL Azure). Dostawca jest przechowywane jako część [projektu programu Entity Framework Core](#).

Zainstaluj

Zainstaluj [pakietu Microsoft.EntityFrameworkCore.SqlServer NuGet](#).

```
Install-Package Microsoft.EntityFrameworkCore.SqlServer
```

Rozpocznij

Następujące zasoby pomoże Ci rozpocząć pracę z tym dostawcą.

- [Rozpoczynanie pracy w programie .NET Framework \(Konsola, WinForms, WPF, itd.\)](#)
- [Wprowadzenie do programu ASP.NET Core](#)
- [UnicornStore przykładowej aplikacji](#)

Obsługiwane baz danych

- Microsoft SQL Server (2008 lub nowszy)

Obsługiwane platformy

- .NET framework (4.5.1 lub nowszy)
- .NET Core
- Narzędzie mono (4.2.0 lub nowszy)

Caution: Using this provider on Mono will make use of the Mono SQL Client implementation, which has a number of known issues. For example, it does not support secure connections (SSL).

Tabele zoptymalizowane pod kątem pamięci, obsługa w dostawcy bazy danych programu SQL Server programu EF Core

28.08.2018 • 2 minutes to read • [Edit Online](#)

NOTE

Ta funkcja została wprowadzona w programie EF Core 1.1.

Zoptymalizowane pod kątem pamięci tabeli są funkcją programu SQL Server, w którym cała tabela znajduje się w pamięci. Drugą kopię danych tabeli jest zachowywana na dysku, ale tylko na potrzeby trwałości. Dane w tabelach zoptymalizowanych pod kątem pamięci jest tylko do odczytu z dysku podczas odzyskiwania bazy danych. Na przykład po serwera należy uruchomić ponownie.

Konfigurowanie tabeli zoptymalizowanej pod kątem pamięci

Można określić, czy jednostka jest zamapowana do tabel jest zoptymalizowane pod kątem pamięci. Podczas tworzenia i bazę danych przy użyciu programu EF Core na podstawie modelu (przy użyciu migracji lub `Database.EnsureCreated()`), zoptymalizowana pod kątem pamięci tabeli zostanie utworzony dla tych jednostek.

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Blog>()
        .ForSqlServerIsMemoryOptimized();
}
```

Dostawca bazy danych systemu SQLite EF Core

28.08.2018 • 2 minutes to read • [Edit Online](#)

Ten dostawca bazy danych umożliwia platformy Entity Framework Core do użycia z bazą danych SQLite. Dostawca jest przechowywane jako część [projektu platformy Entity Framework Core](#).

Zainstaluj

Zainstaluj [pakietu Microsoft.EntityFrameworkCore.Sqlite NuGet](#).

```
Install-Package Microsoft.EntityFrameworkCore.Sqlite
```

Rozpocznij

Następujące zasoby pomoże Ci rozpocząć pracę z tym dostawcą.

- [Lokalne bazy danych SQLite na platformy uniwersalnej systemu Windows](#)
- [Aplikacji .NET core do nowej bazy danych SQLite](#)
- [Unicorn Clicker przykładowej aplikacji](#)
- [Unicorn Packer przykładowej aplikacji](#)

Obsługiwane baz danych

- Bazy danych SQLite (3.7 lub nowszy)

Obsługiwane platformy

- .NET framework (4.5.1 lub nowszy)
- .NET Core
- Narzędzie mono (4.2.0 lub nowszy)
- Platforma uniwersalna systemu Windows

Ograniczenia

Zobacz [ograniczenia SQLite](#) dla pewne ważne ograniczenia dotyczące dostawcy bazy danych SQLite.

Ograniczenia dotyczące dostawcy bazy danych SQLite EF Core

28.08.2018 • 2 minutes to read • [Edit Online](#)

Dostawca bazy danych SQLite ma kilka ograniczeń migracji. Większość z tych ograniczeń są wynikiem ograniczenia podstawowego aparatu bazy danych SQLite i nie są specyficzne dla platformy EF.

Ograniczenia modelowania

Wspólne biblioteki relacyjnych (udostępnione przez dostawców relacyjnej bazy danych programu Entity Framework) definiuje interfejsów API do modelowania pojęcia, które są wspólne dla większości relacyjnych baz danych. Kilka z tych koncepcji nie są obsługiwane przez dostawcę bazy danych SQLite.

- Schematy
- Sekwencje

Ograniczenia migracji

Aparat bazy danych SQLite nie obsługuje szereg operacji schematu, które są obsługiwane przez większość innych relacyjnych baz danych. Jeśli spróbujesz zastosować jedną nieobsługiwane operacje do bazy danych SQLite, a następnie `NotSupportedException` zostanie zgłoszony.

OPERACJA	OBSŁUGIWANE?	WYMAGA WERSJI
AddColumn	✓	1.0
AddForeignKey	✗	
AddPrimaryKey	✗	
AddUniqueConstraint	✗	
AlterColumn	✗	
CreateIndex	✓	1.0
CreateTable	✓	1.0
DropColumn	✗	
DropForeignKey	✗	
DropIndex	✓	1.0
DropPrimaryKey	✗	
DropTable	✓	1.0

OPERACJA	OBSŁUGIWANE?	WYMAGA WERSJI
DropUniqueConstraint	✗	
RenameColumn	✗	
RenameIndex	✓	2.1
RenameTable	✓	1.0
EnsureSchema	✓ (pusta)	2.0
DropSchema	✓ (pusta)	2.0
Insert	✓	2.0
Aktualizacja	✓	2.0
Usuwanie	✓	2.0

Obejście ograniczenia dotyczące migracji

Można obejść niektóre z tych ograniczeń, ręcznie wpisując kod w migracji do wykonania tabeli ponownie skompilować. Odbuduj tabelę obejmuje zmianę nazwy istniejącej tabeli, tworzenie nowej tabeli, kopowanie danych do nowej tabeli i usunięcie starych tabeli. Należy użyć `Sql(string)` metodę, aby wykonać niektóre z tych kroków.

Zobacz [co inne rodzaje z zmiany schematu tabeli](#) w dokumentacji oprogramowania SQLite, aby uzyskać więcej informacji.

W przyszłości EF mogą obsługiwać kilku z tych operacji przy użyciu podejścia Odbuduj tabelę w sposób niewidoczny. Możesz [śledzenie tej funkcji w projekcie usługi GitHub](#).

Dostawca programu EF Core bazy danych w pamięci

28.08.2018 • 2 minutes to read • [Edit Online](#)

Ten dostawca bazy danych umożliwia platformy Entity Framework Core ma być używany z bazą danych w pamięci. Może to być przydatne w przypadku testowania, mimo że dostawcy bazy danych SQLite w trybie w pamięci może być bardziej odpowiednie zastąpienia testu dla relacyjnych baz danych. Dostawca jest przechowywane jako część projektu programu Entity Framework Core.

Zainstaluj

Zainstaluj [pakietu Microsoft.EntityFrameworkCore.InMemory NuGet](#).

```
Install-Package Microsoft.EntityFrameworkCore.InMemory
```

Rozpocznij

Następujące zasoby pomoże Ci rozpocząć pracę z tym dostawcą.

- [Testowanie za pomocą InMemory](#)
- [Przykładowe UnicornStore testów aplikacji](#)

Obsługiwane baz danych

- Wbudowane bazy danych w pamięci (przeznaczony tylko do celów testowych)

Obsługiwane platformy

- .NET framework (4.5.1 lub nowszy)
- .NET Core
- Narzędzie mono (4.2.0 lub nowszy)
- Platforma uniwersalna systemu Windows

Pisanie dostawcy bazy danych

28.08.2018 • 2 minutes to read • [Edit Online](#)

Informacje na temat pisania dostawcy bazy danych programu Entity Framework Core, zobacz [więc chcesz napisać dostawcy programu EF Core](#) przez [Arthur Vickers](#).

NOTE

Te wpisy nie zostały zaktualizowane od czasu programu EF Core 1.1 i od tego czasu miały miejsce znaczące zmiany [681 problem](#) śledzenie aktualizacji tej dokumentacji.

Codebase programu EF Core jest "open source" i zawiera kilka dostawcy baz danych, które mogą służyć jako odwołanie. Można znaleźć kodu źródłowego w <https://github.com/aspnet/EntityFrameworkCore>. Może to być również przydatne Spójrz na kod dla często używanych dostawców innych firm, takich jak [Npgsql](#), [Pomelo MySQL](#), i [programu SQL Server Compact](#). W szczególności te projekty są Instalatora, aby rozszerzyć i uruchomić testy funkcjonalne, które możemy opublikować dla narzędzia NuGet. Zdecydowanie zaleca się tego rodzaju instalacji.

Aktualizowanie zmian w dostawcy

Po wydaniu wersji 2.1, zaczynając od pracy, firma Microsoft opracowała [dziennika zmian](#), może być konieczne odpowiednie zmiany w kodzie dostawcy. Ta wartość jest przeznaczona do pomocy podczas aktualizowania istniejącego dostawcy do pracy z nową wersją programu EF Core.

Przed 2.1, użyliśmy `providers-beware` i `providers-fyi` etykiety na naszych problemach usługi GitHub i żądań ściągnięcia dla podobnych celów. Firma Microsoft będzie kontynuować potrzeby tych labels na problemy stanowiące wskazówkę, które elementy robocze w danej wersji mogą również wymagać pracy w dostawcy. A `providers-beware` etykiety zazwyczaj oznacza, że implementacja elementu roboczego może spowodować przerwanie dostawców, podczas gdy `providers-fyi` etykiety zazwyczaj oznacza, że dostawcy nie będą działać, ale kod może być konieczny, należy zmienić mimo to, na przykład, aby włączyć nowe funkcje.

Sugerowane nazewnictwa dostawców innych firm

Zaleca się przy użyciu następujących nazewnictwa dla pakietów NuGet. Jest to zgodne z nazwami pakietów dostarczone przez zespół programu EF Core.

```
<Optional project/company name>.EntityFrameworkCore.<Database engine name>
```

Na przykład:

- `Microsoft.EntityFrameworkCore.SqlServer`
- `Npgsql.EntityFrameworkCore.PostgreSQL`
- `EntityFrameworkCore.SqlServerCompact40`

Zmiany wpływające na dostawcy

01.11.2018 • 5 minutes to read • [Edit Online](#)

Ta strona zawiera linki do przeprowadzanych na repozytorium programu EF Core, które mogą wymagać autorzy innych dostawców bazy danych, aby reagować żądań ściągnięcia. Jest do udostępniania punktu wyjściowego dla autorów istniejącego dostawcy baz danych innych firm, podczas aktualizowania ich dostawcy do nowej wersji.

Rozpoczynamy ten dziennik zmian z 2.1 do wersji 2.2. Przed 2.1 użyliśmy [providers-beware](#) i [providers-fyi](#) etykiety na nasze problemy i żądania ściągnięcia.

2.1---> 2.2

Zmiany tylko do testów

- <https://github.com/aspnet/EntityFrameworkCore/pull/12057> -Zezwalaj na możliwe do dostosowania ograniczników SQL w testach
 - Testowanie zmiany, które umożliwiają nieścisłym porównania punktu zmienoprzecinkowego w `BuiltInDataTypesTestBase`
 - Zmiany testów, które umożliwiają zapytania testy, aby być ponownie używane, za pomocą różnych ograniczników SQL
- <https://github.com/aspnet/EntityFrameworkCore/pull/12072> -Dodaj testy `DbFunction` testów specyfikacji relacyjnych
 - Taki sposób, że te testy mogą być uruchamiane względem wszystkich dostawców bazy danych
- <https://github.com/aspnet/EntityFrameworkCore/pull/12362> -Czyszczenie testowego przełączania do `Async`
 - Usuń `Wait` wywołań, niepotrzebne `async` i zmień nazwy niektórych metod testowych
- <https://github.com/aspnet/EntityFrameworkCore/pull/12666> -Ujednolicenie infrastrukturę testowania rejestrowania
 - Dodano `CreateListLoggerFactory` i usunąć niektóre poprzedniego infrastruktury rejestrowania, który będzie wymagać dostawców przy użyciu tych testów, reakcji
- <https://github.com/aspnet/EntityFrameworkCore/pull/12500> — Uruchamianie więcej testów zapytania synchronicznie i asynchronicznie
 - Nazwy testów i uwzględniając uległ zmianie, co będzie wymagać dostawców przy użyciu tych testów, reakcji
- <https://github.com/aspnet/EntityFrameworkCore/pull/12766> -Zmiana nazwy tego modelu `ComplexNavigations`
 - Za pomocą tych testów dostawców może być konieczne reagowanie
- <https://github.com/aspnet/EntityFrameworkCore/pull/12141> -Zwracania kontekstu danej puli zamiast usuwania w testy funkcjonalne
 - Ta zmiana obejmuje niektóre refaktoryzacji testów, które mogą wymagać dostawców reagować

Test i produktu zmian w kodzie

- <https://github.com/aspnet/EntityFrameworkCore/pull/12109> -Skonsolidować `RelationalTypeMapping.Clone` metody
 - Zmiany w 2.1 `RelationalTypeMapping` dozwolone dla uproszczenia w klasach pochodnych. Firma Microsoft uważa, nie zostało to istotne do dostawców, ale dostawców korzystać z zalet tej zmiany w ich typ pochodny mapowania klas.
- <https://github.com/aspnet/EntityFrameworkCore/pull/12069> -Oznakowane lub nazwanego zapytania
 - Dodaje infrastrukturę na potrzeby znakowania zapytań LINQ i posiadanie tych tagów, są wyświetlane

jako komentarze w SQL. Może to wymagać dostawców reagowania generowanie kodu SQL.

- <https://github.com/aspnet/EntityFrameworkCore/pull/13115> — Obsługuje dane przestrzenne za pośrednictwem nktę przerwania
 - Umożliwia mapowania typów i elementów członkowskich tłumaczy do zarejestrowania poza dostawcy
 - Dostawcy musi wywołać podstawowej. FindMapping() w ich implementacji ITypeMappingSource, aby działał
 - Postępuj zgodnie z tego wzorca, aby dodać obsługę przestrzenne do dostawcą, który jest spójny w ramach dostawcy.
- <https://github.com/aspnet/EntityFrameworkCore/pull/13199> — Dodawanie Ulepszone debugowanie Tworzenie dostawcy usługi
 - Umożliwia DbContextOptionsExtensions wdrożyć nowy interfejs, który pomaga zrozumieć, dlaczego dostawcy usługi wewnętrznej jest przebudowany osób
- <https://github.com/aspnet/EntityFrameworkCore/pull/13289> -Dodaje CanConnect interfejsu API do użytku przez kontrole kondycji
 - To żądanie Ściagnięcia dodaje koncepcji `CanConnect` który będzie używany przez platformy ASP.NET Core kondycji sprawdza, czy baza danych jest dostępna. Domyślnie, implementacja relacyjnych po prostu wywołuje funkcję `Exist`, ale dostawców można zaimplementować inny, jeśli to konieczne. Nierelacyjne dostawców należy zaimplementować nowy interfejs API w kolejności dla kontroli kondycji może być używany.
- <https://github.com/aspnet/EntityFrameworkCore/pull/13306> — Aktualizowanie podstawowy RelationalTypeMapping nie ustawiać DbParameter rozmiar
 - Zatrzymaj, ustawianie rozmiaru domyślnie, ponieważ może to spowodować obcięcie. Dostawców może być konieczne dodanie własnych logiki, jeśli rozmiar musi być ustawniona.
- <https://github.com/aspnet/EntityFrameworkCore/pull/13372> -RevEng: Zawsze określać typ kolumny dla kolumny dziesiętna
 - Zawsze należy skonfigurować typ kolumny dla kolumny dziesiętną w utworzony szkielet kodu, zamiast konfigurować zgodnie z Konwencją.
 - Dostawców nie powinna wymagać zmiany po ich stronie.
- <https://github.com/aspnet/EntityFrameworkCore/pull/13469> -Dodaje CaseExpression generowania wyrażenia SQL CASE
- <https://github.com/aspnet/EntityFrameworkCore/pull/13648> -Dodaje możliwość określenia mapowania typów na SqlFunctionExpression w celu zwiększenia magazynu wnioskowanie argumentów i wyników.

EF Core Tools i rozszerzenia

08.01.2019 • 7 minutes to read • [Edit Online](#)

Te narzędzia i rozszerzenia zapewniają dodatkowe funkcje dla programu Entity Framework Core 2.0 i nowszych.

IMPORTANT

Rozszerzenia są tworzone za pomocą różnych źródeł i nie są przechowywane jako część projektu platformy Entity Framework Core. Rozważając rozszerzenia innych firm, należy ocenić jego jakość licencjonowania, zgodności i pomocy technicznej, itp., aby upewnić się, że spełnia on wymagania.

Narzędzia

LLBLGen Pro

LLBLGen Pro jest jednostką modelowania rozwiązania z obsługą platformy Entity Framework i Entity Framework Core. Dzięki temu można łatwo zdefiniować model jednostki i dokonaj mapowania go do bazy danych, najpierw przy użyciu bazy danych lub model, dzięki czemu możesz rozpocząć pracę już teraz Pisanie zapytań.

[Witryny sieci Web](#)

Devart jednostki dla deweloperów

Deweloper jednostki jest zaawansowane Projektant ORM dla ADO.NET Entity Framework, NHibernate, LinqConnect, Telerik Data Access i LINQ to SQL. Go obsługuje projektowanie wizualnie modeli programu EF Core, najpierw przy użyciu modelu lub bazy danych najpierw zbliża się, a C# lub generowanie kodu w języku Visual Basic.

[Witryny sieci Web](#)

EF Core Power Tools

EF Core Power Tools to rozszerzenie programu Visual Studio 2017, które udostępnia różne zadania projektowania programu EF Core w prosty interfejs użytkownika. Obejmuje odtwarzanie klasy DbContext i jednostki z istniejących baz danych i [programu SQL Server DACPACs](#), zarządzanie migracjami baz danych i wizualizacje modelu.

[Witryny typu wiki w witrynie GitHub](#)

Edytor programu Entity Framework Visual

Edytor programu Entity Framework Visual to rozszerzenie programu Visual Studio 2017, które dodaje Projektant ORM dla projektowania wizualnego programów EF 6 i klas programu EF Core. Kod jest generowany przy użyciu szablonów T4, dzięki czemu można dostosować do dowolnego potrzeb. Obsługuje dziedziczenie, jednokierunkowe i powiązania dwukierunkowego, wyliczenia i możliwość kolorować klas i Dodaj bloki tekstu, aby wyjaśnić, potencjalnie specjalne części projektu.

[Portal Marketplace](#)

CatFactory

CatFactory to aparat tworzenia szkieletów dla platformy .NET Core, który można zautomatyzować Generowanie klasy DbContext, jednostki, mapowania konfiguracji i repozytorium klasy z bazy danych programu SQL Server.

[Repozytorium GitHub](#)

Firmy LoreSoft Entity Framework Core Generator

Entity Framework Core Generator (efg) jest narzędziem wiersza polecenia platformy .NET Core z można wygenerować modeli programu EF Core istniejącą bazę danych, podobnie jak `dotnet ef dbcontext scaffold`, lecz również obsługuje bezpieczne kodu [ponownego wygenerowania](#) za pomocą zastąpienia regionu lub przez analizowanie Plików mapowania. To narzędzie obsługuje generowania modeli widoków, weryfikacji i obiektu mapowania kodu.

[Samouczek dokumentacji](#)

Rozszerzenia

Microsoft.EntityFrameworkCore.AutoHistory

Biblioteka wtyczki, która umożliwia automatyczne rejestrowanie zmian danych, wykonywane przez platformę EF Core w tabeli historii.

[Repozytorium GitHub](#)

Microsoft.EntityFrameworkCore.DynamicLinq

.NET Core / .NET Standard port z System.Linq.Dynamic z obsługą async z programem EF Core.

System.Linq.Dynamic pochodzi jako przykład firmy Microsoft, który pokazuje, jak tworzyć zapytania LINQ dynamicznie z wyrażenia ciągu zamiast kodu.

[Repozytorium GitHub](#)

EFSecondLevelCache.Core

Rozszerzenie, które umożliwia przechowywanie wyników zapytania programu EF Core do pamięci podręcznej drugiego poziomu, tak aby kolejne wykonania tego samego zapytania można uniknąć, uzyskiwanie dostępu do bazy danych i pobierać dane bezpośrednio z pamięci podręcznej.

[Repozytorium GitHub](#)

EntityFrameworkCore.PrimaryKey

Ta biblioteka umożliwia pobieranie wartości klucza podstawowego (w tym kluczy złożonych) z dowolnej jednostki jako słownik.

[Repozytorium GitHub](#)

EntityFrameworkCore.TypedOriginalValues

Ta biblioteka umożliwia silnie typizowany dostęp do oryginalnych wartości właściwości jednostki.

[Repozytorium GitHub](#)

Geco

Geco (Generator konsoli) jest generator prostego kodu na podstawie projektu konsoli, działa na platformie .NET Core, która używa C# interpolowane ciągów w celu generowania kodu. Geco zawiera generator modelu odwróconego dla platformy EF Core dzięki obsłudze pluralizacji, singularizacji i szablonów do edycji. Umożliwia także skrypt generator danych inicjatora, runner skryptu i bazę danych czyszcząca.

[Repozytorium GitHub](#)

LinqKit.Microsoft.EntityFrameworkCore

LinqKit.Microsoft.EntityFrameworkCore jest zgodnych z programem EF Core wersji biblioteki LINQKit. LINQKit to bezpłatny zestaw rozszerzeń dla programu LINQ do SQL i Entity Framework użytkownicy zaawansowani.

Umożliwia ona zaawansowane funkcje, takie jak dynamiczne tworzenie predykatu wyrażeń i używanie zmiennych wyrażenia w podzapytania.

[Repozytorium GitHub](#)

NeinLinq.EntityFrameworkCore

NeinLinq rozszerza dostawców LINQ takim jak Entity Framework, aby umożliwić ponowne używanie funkcji ponowne napisanie zapytania i komplikowania zapytań dynamicznych przy użyciu można przetłumaczyć predykatach i selektory.

[Repozytorium GitHub](#)

Microsoft.EntityFrameworkCore.UnitOfWork

Wtyczki dla Microsoft.EntityFrameworkCore umożliwiają obsługę repozytorium, jednostkę pracy wzorców i wielu baz danych w transakcji rozproszonej, obsługiwane.

[Repozytorium GitHub](#)

EFCore.BulkExtensions

EF Core rozszerzenia potrzeby zbiorczych operacji (Insert, Update, Delete).

[Repozytorium GitHub](#)

Bricelam.EntityFrameworkCore.Pluralizer

Dodaje pluralizacja czasu projektowania do programu EF Core.

[Repozytorium GitHub](#)

PomeloFoundation/Pomelo.EntityFrameworkCore.Extensions.ToSql

Metody rozszerzenia prosty, która uzyskuje instrukcji SQL programu EF Core wygeneruje dla danego zapytania LINQ w prostych scenariuszy. Metoda ToSql jest ograniczone do prostych scenariuszy, ponieważ programu EF Core można wygenerować więcej niż jedną instrukcję SQL dla pojedynczego zapytania LINQ i różne instrukcje SQL, w zależności od wartości parametrów.

[Repozytorium GitHub](#)

Toolbelt.EntityFrameworkCore.IndexAttribute

Revival atrybutu [Index] dla platformy EF Core (z rozszerzeniem do konstruowania modelu).

[Repozytorium GitHub](#)

EfCore.InMemoryHelpers

Udostępnia otokę dostawcy bazy danych programu EF Core w pamięci. Sprawia, że więcej zachowywać się jak relacyjne dostawcy.

[Repozytorium GitHub](#)

EFCore.TemporalSupport

Implementacja danych czasowych pomocy technicznej dla platformy EF Core.

[Repozytorium GitHub](#)

EntityFrameworkCore.Cacheable

Pamięć podręczną zapytań drugiego poziomu o wysokiej wydajności dla platformy EF Core.

[Repozytorium GitHub](#)

Dotyczące narzędzi programu Entity Framework Core

29.09.2018 • 2 minutes to read • [Edit Online](#)

Pomoc narzędzia Entity Framework Core przy użyciu zadania podczas projektowania. Są głównie używane do zarządzania, migracji i do tworzenia szkieletu `DbContext` i typów jednostki przez odtwarzanie schemat bazy danych.

- [Narzędzia Konsola Menedżera pakietów programu EF Core](#)) uruchom w [Konsola Menedżera pakietów](#) w programie Visual Studio. Te narzędzia działają w projektach .NET Core i .NET Framework.
- [Narzędzi interfejsu wiersza polecenia \(CLI\) platformy EF Core .NET](#) to rozszerzenie dla wielu platform [narzędzi interfejsu wiersza polecenia platformy .NET Core](#). Narzędzia te wymagają projektu .NET Core SDK (z `Sdk="Microsoft.NET.Sdk"` lub podobne w pliku projektu).

Oba narzędzia uwidaczniają taką samą funkcjonalność. Jeśli tworzysz w programie Visual Studio, zalecamy użycie **Konsola Menedżera pakietów** narzędzi, ponieważ zapewniają one bardziej zintegrowanego środowiska pracy.

Następne kroki

- [Konsola Menedżera pakietów programu EF Core odnoszą się narzędzia](#)
- [Odwołują się do narzędzia .NET Core interfejsu wiersza polecenia platformy EF](#)

Entity Framework Core odnoszą się narzędzi — Konsola Menedżera pakietów w programie Visual Studio

16.11.2018 • 15 minutes to read • [Edit Online](#)

Narzędzia konsoli Menedżera pakietów (PMC) dla platformy Entity Framework Core wykonywać zadania podczas projektowania. Na przykład można utworzyć [migracje](#), zastosuj migracje i wygenerować kod dla modelu, w oparciu o istniejącą bazę danych. Polecenia uruchamiane wewnątrz programu Visual Studio przy użyciu [Konsola Menedżera pakietów](#). Te narzędzia działają w projektach .NET Core i .NET Framework.

Jeśli nie używasz programu Visual Studio, firma Microsoft zaleca [narzędzi wiersza polecenia programu EF Core](#) zamiast tego. Narzędzia interfejsu wiersza polecenia są wieloplatformowe i wykonywania w wierszu polecenia.

Instalowanie narzędzi

Procedury instalowania i aktualizowania narzędzia różnią się między platformą ASP.NET Core 2.1 + i wcześniejszych wersji lub innych typów projektów.

Platforma ASP.NET Core w wersji 2.1 i nowsze

Narzędzia są automatycznie uwzględniane w projektach programu ASP.NET Core 2.1 +, ponieważ

`Microsoft.EntityFrameworkCore.Tools` pakietu znajduje się w [meta Microsoft.aspnetcore.all](#)

`Microsoft.AspNetCore.App`.

W związku z tym nie trzeba nic robić, aby zainstalować narzędzia, ale trzeba:

- Przywróć pakiety przed użyciem narzędzia w nowym projekcie.
- Zainstaluj pakiet, aby zaktualizować narzędzia do nowszej wersji.

Aby upewnić się, że otrzymujesz najnowszej wersji narzędzia, firma Microsoft zaleca również wykonać poniższe czynności:

- Edytuj swoje `.csproj` pliku i Dodaj wiersz, określając najnowszą wersję `Microsoft.EntityFrameworkCore.Tools` pakietu. Na przykład `.csproj` plik może zawierać `<ItemGroup>`, wygląda następująco:

```
<ItemGroup>
  <PackageReference Include="Microsoft.AspNetCore.App" />
  <PackageReference Include="Microsoft.EntityFrameworkCore.Tools" Version="2.1.3" />
  <PackageReference Include="Microsoft.VisualStudio.Web.CodeGeneration.Design" Version="2.1.1" />
</ItemGroup>
```

Aktualizacja narzędzi, gdy zostanie wyświetlony komunikat jak w poniższym przykładzie:

Wersja narzędzi programu EF Core "2.1.1-rtm-30846" jest starsza niż w przypadku środowiska uruchomieniowego "2.1.3-rtm-32065". Aktualizacja narzędzi do najnowszych funkcji i poprawek błędów.

Aby zaktualizować narzędzia:

- Zainstaluj najnowszy zestaw .NET Core SDK.
- Aktualizacja programu Visual Studio do najnowszej wersji.

- Edytuj `.csproj` plik zawiera odwołania do pakietu do najnowszego pakietu narzędzi, jak pokazano wcześniej.

Inne wersje i typy projektów

Instalowanie narzędzi Konsola Menedżera pakietów, uruchamiając następujące polecenie w **Konsola Menedżera pakietów**:

```
Install-Package Microsoft.EntityFrameworkCore.Tools
```

Aktualizacja narzędzi, uruchamiając następujące polecenie w **Konsola Menedżera pakietów**.

```
Update-Package Microsoft.EntityFrameworkCore.Tools
```

Weryfikowanie instalacji

Sprawdź, czy narzędzia są zainstalowane, uruchamiając następujące polecenie:

```
Get-Help about_EntityFrameworkCore
```

Dane wyjściowe wyglądają następująco (program nie nakazuje możesz wersję narzędzia używasz):

```
          _/\_
  ____/ \ \ \
  | .   \| \
  | | | | | )  \ \
  | | | | | \ / | //| \\
  | | | | | /   \ \ \ \ \\

TOPIC
  about_EntityFrameworkCore

SHORT DESCRIPTION
  Provides information about the Entity Framework Core Package Manager Console Tools.

<A list of available commands follows, omitted here.>
```

Przy użyciu narzędzi

Przed rozpoczęciem korzystania z narzędzi:

- Należy zrozumieć różnicę między docelowej i uruchamianie projektu.
- Dowiedz się, jak używać narzędzi z biblioteki klas .NET Standard.
- Dla projektów ASP.NET Core należy ustawić środowisko.

Element docelowy i uruchamianie projektu

Polecenia można znaleźć *projektu* i *projekt startowy*.

- *Projektu* jest także znana jako *projekt docelowy* ponieważ jest za której polecenia Dodawanie lub usuwanie plików. Domyslnie **projekt domyślny** wybranego w **Konsola Menedżera pakietów** jest projekt docelowy. Należy określić inny projekt jako projekt docelowy za pomocą `--project` opcji.
- *Projekt startowy* jest tą, która narzędzi skompilować i uruchomić. Narzędzia konieczne wykonanie kodu aplikacji w czasie projektowania, aby uzyskać informacje na temat projektu, takie jak parametry połączenia bazy danych i Konfiguracja modelu. Domyslnie **projekt startowy** w **Eksploratora rozwiązań** jest projektem startowym. Należy określić inny projekt jako projekt startowy, za pomocą

--startup-project opcji.

Projekt startowy i projekt docelowy często są tym samym projektem. Jest to typowy scenariusz, w którym są one oddzielnych projektów, gdy:

- EF Core klasy kontekstu i jednostki są w bibliotece klas .NET Core.
- Aplikacją sieci web lub aplikacji konsolowej .NET Core odwołuje się do biblioteki klas.

Istnieje również możliwość [umieść kod migracji w bibliotece klas, niezależnie od kontekstu programu EF Core](#).

Innych platform docelowych

Narzędzia Konsola Menedżera pakietów pracować z projektami .NET Core lub .NET Framework. Aplikacje, które mają model platformy EF Core w bibliotece klas programu .NET Standard może nie mieć platformy .NET Core lub .NET Framework projektu. Na przykład dotyczy to aplikacji platformy Xamarin i platformy uniwersalnej Windows. W takich przypadkach można utworzyć platformy .NET Core lub .NET Framework projekt aplikacji konsoli którego jedynym celem jest do działania jako projekt startowy dla narzędzi. Projekt może być fikcyjnego projektu bez rzeczywistego kodu — jest wymagana tylko do zapewnienia celu narzędzi.

Dlaczego jest projektem fikcyjnym wymagane? Jak wspomniano wcześniej, narzędzia konieczne wykonanie kodu aplikacji w czasie projektowania. Aby to zrobić, muszą one korzystać ze środowiska uruchomieniowego .NET Core lub .NET Framework. Jeśli model programu EF Core jest w projekcie, który jest przeznaczony dla platformy .NET Core lub .NET Framework, narzędzia programu EF Core "pożyczają" środowisko uruchomieniowe z projektu. Nie można wykonać, jeśli model programu EF Core jest w bibliotece klas programu .NET Standard. .NET Standard nie jest rzeczywistą implementacją .NET; jest określenie zestawu interfejsów API, który musi obsługiwać implementacje platformy .NET. W związku z tym .NET Standard nie jest wystarczająca dla narzędzi programu EF Core do wykonywania kodu aplikacji. Projekt zastępczy, utworzone jako projekt startowy zawiera platformy konkretnego celu, w którym narzędzia można załadować biblioteki klas .NET Standard.

Środowiska ASP.NET Core

Aby określić środowisko dla projektów ASP.NET Core, ustaw **env:ASPNETCORE_ENVIRONMENT** przed uruchomieniem polecenia.

Wspólne parametry

W poniższej tabeli przedstawiono parametry, które są wspólne dla wszystkich poleceń programu EF Core:

PARAMETR	OPIS
-Kontekstu <ciąg>	<code>DbContext</code> Klasy. Nazwa klasy tylko lub w pełni kwalifikowana za pomocą przestrzeni nazw. Jeśli ten parametr zostanie pominięty, programem EF Core umożliwia znalezienie klasy kontekstu. W przypadku wielu klas kontekstu, ten parametr jest wymagany.
-Projekt <ciąg>	Projekt docelowy. Jeśli ten parametr zostanie pominięty, projekt domyślny dla Konsola Menedżera pakietów służy jako projekt docelowy.
Projekt startowy - <ciąg>	Projekt startowy. Jeśli ten parametr zostanie pominięty, projekt startowy w właściwości rozwiązań służy jako projekt docelowy.
-Verbose	Wyświetlić pełne dane wyjściowe.

Aby wyświetlić Pomoc dotyczącą polecenia, użyj programu PowerShell `Get-Help` polecenia.

TIP

Parametry kontekstu, projekt i projekt startowy obsługują rozszerzenia karty.

Dodaj migracji

Dodaje nową migrację.

Parametry:

PARAMETR	OPIS
— Nazwa <ciąg>	Nazwa migracji. To jest parametr pozycyjne i jest wymagana.
-OutputDir <ciąg>	Katalog (i podzielonej przestrzeni nazw) do użycia. Ścieżki są względne wobec katalogu docelowego projektu. Wartość domyślna to "Migracja".

Baza danych listy

Porzuca bazy danych.

Parametry:

PARAMETR	OPIS
-WhatIf	Pokaż bazę danych, która będzie można usunąć, ale nie należy usuwać jej.

Get-DbContext

Wyświetla dostępne `DbContext` typów.

Usuń migrację

Usuwa ostatniej migracji (wycofanie zmian w kodzie, które zostały przygotowane do migracji).

Parametry:

PARAMETR	OPIS
-Force	Przywróć migracji (wycofać zmiany, które zostały zastosowane do bazy danych).

Tworzenie szkieletu DbContext

Generuje kod dla `DbContext` i typy jednostek bazy danych. Aby `Scaffold-DbContext` można wygenerować typu jednostki, w tabeli bazy danych musi mieć klucz podstawowy.

Parametry:

PARAMETR	OPIS
-Connection <ciąg >	Parametry połączenia z bazą danych. W przypadku projektów ASP.NET Core 2.x wartość może być <i>nazwa</i> = < <i>nazwa parametrów połączenia</i> >. W takim przypadku nazwa pochodzi od źródła konfiguracji, które są skonfigurowane dla projektu. To jest parametr pozycyjne i jest wymagana.
-Provider <ciąg >	Dostawca do użycia. Zazwyczaj jest to nazwa pakietu NuGet, na przykład: <code>Microsoft.EntityFrameworkCore.SqlServer</code> . To jest parametr pozycyjne i jest wymagana.
-OutputDir <ciąg >	Katalog, który można umieścić pliki w. Ścieżki są względne wobec katalogu projektu.
-ContextDir <ciąg >	Katalog, aby umieścić <code>DbContext</code> w pliku. Ścieżki są względne wobec katalogu projektu.
-Kontekstu <ciąg >	Nazwa <code>DbContext</code> klasy do wygenerowania.
-Schematów <String [] >	Schematów tabel w celu wygenerowania typów jednostek dla. Jeżeli ten parametr zostanie pominięty, uwzględniono wszystkich schematów.
-Tabele <String [] >	Tabele, aby wygenerować typy jednostek dla. Jeżeli ten parametr zostanie pominięty, uwzględniane są wszystkie tabele.
-DataAnnotations	Aby skonfigurować model (tam, gdzie jest to możliwe), należy użyć atrybutów. Jeżeli pominięto ten parametr jest używany tylko interfejsu API fluent.
-UseDatabaseNames	Użyj nazwy tabel i kolumn dokładnie tak, jak pojawiają się w bazie danych. Jeżeli ten parametr zostanie pominięty, nazwy baz danych zostały zmienione, aby ściślej odpowiadają Konwencji stylistycznych nazwy języka C#.
-Force	Nadpisz istniejące pliki.

Przykład:

```
Scaffold-DbContext "Server=(localdb)\mssqllocaldb;Database=Blogging;Trusted_Connection=True;"  
Microsoft.EntityFrameworkCore.SqlServer -OutputDir Models
```

Przykład szkielety mechanizmów tylko wybrane tabele i tworzy kontekst w oddzielnym folderze z określona nazwą:

```
Scaffold-DbContext "Server=(localdb)\mssqllocaldb;Database=Blogging;Trusted_Connection=True;"  
Microsoft.EntityFrameworkCore.SqlServer -OutputDir Models -Tables "Blog","Post" -ContextDir Context -Context  
BlogContext
```

Skrypt migracji

Generuje skrypt SQL, który dotyczy wszystkich zmian z jedną migrację wybranych innej wybranej migracji.

Parametry:

PARAMETR	OPIS
-From <ciąg >	Począwszy od migracji. Migracje mogą być określane według nazwy lub identyfikatora. Numer 0 jest przypadkiem szczególnym, oznacza to <i>przed migracją pierwszy</i> . Wartość domyślna to 0.
— Do <ciąg >	Końcowy migracji. Domyślnie do ostatniego migracji.
-Idempotentne	Generowanie skryptu, który może służyć w bazie danych w każdej migracji.
-Dane wyjściowe <ciąg >	Plik można zapisać wynik. Jeśli ten parametr zostanie pominięty, plik jest tworzony z użyciem nazwy wygenerowanej w tym samym folderze, ponieważ pliki środowiska uruchomieniowego aplikacji są tworzone, na przykład: /obj/Debug/netcoreapp2.1/ghbkztfz.sql/.

TIP

To, z danych wyjściowych z obsługą i parametrów rozszerzenia karty.

Poniższy przykład tworzy skrypt w przypadku migracji InitialCreate przy użyciu nazwy migracji.

```
Script-Migration -To InitialCreate
```

Poniższy przykład tworzy skrypt wszystkich migracji po migracji InitialCreate przy użyciu identyfikatora migracji.

```
Script-Migration -From 20180904195021_InitialCreate
```

Aktualizuj bazy danych

Aktualizuje bazę danych do ostatniej migracji lub określony migracji.

PARAMETR	OPIS
— Migracja <ciąg >	Migracja docelowego. Migracje mogą być określane według nazwy lub identyfikatora. Numer 0 jest przypadkiem szczególnym, oznacza to <i>przed migracją pierwszy</i> i powoduje, że wszystkich migracji, które można przywrócić. Jeśli migracja nie zostanie określony, polecenie domyślne ostatnio migracji.

TIP

Parametr migracji obsługuje rozszerzenia karty.

Poniższy przykład przywraca wszystkich migracji.

```
Update-Database -Migration 0
```

Poniższe przykłady aktualizują bazę danych do określonego migracji. Pierwszy używa nazwy migracji, a drugi używa Identyfikatora migracji:

```
Update-Database -Migration InitialCreate  
Update-Database -Migration 20180904195021_InitialCreate
```

Dodatkowe zasoby

- [Migracje](#)
- [Odtwarzanie](#)

Entity Framework Core odnoszą się narzędzia — interfejs wiersza polecenia platformy .NET

16.11.2018 • 15 minutes to read • [Edit Online](#)

Narzędzia interfejsu wiersza polecenia (CLI) dla platformy Entity Framework Core wykonywać zadania podczas projektowania. Na przykład można utworzyć [migracje](#), zastosuj migracje i wygenerować kod dla modelu, w oparciu o istniejącą bazę danych. Polecenia są rozszerzenie dla wielu platform [dotnet](#) polecenia, który jest częścią programu [zestawu .NET Core SDK](#). Te narzędzia działają w projektach .NET Core.

Jeśli używasz programu Visual Studio, firma Microsoft zaleca [narzędzia Konsola Menedżera pakietów](#) zamiast tego:

- Działają automatycznie z bieżącego projektu wybranego w **Konsola Menedżera pakietów** bez konieczności ręcznie Przełącz katalogi.
- One automatycznie otwarte pliki wygenerowane za pomocą polecenia, po zakończeniu polecenia.

Instalowanie narzędzi

Procedura instalacji zależy od typu projektu, a wersja:

- Platforma ASP.NET Core w wersji 2.1 i nowsze
- EF Core 2.x
- EF Core 1.x

Platforma ASP.NET Core 2.1 +

- Zainstaluj bieżącą [zestawu .NET Core SDK](#). Zestaw SDK został zainstalowany, nawet jeśli masz najnowszą wersję programu Visual Studio 2017.

To wszystko, co jest niezbędne dla platformy ASP.NET Core 2.1 +, ponieważ

`Microsoft.EntityFrameworkCore.Design` pakietu znajduje się w [meta Microsoft.aspnetcore.all](#)
`Microsoft.AspNetCore.App`.

EF Core 2.x (nie platformy ASP.NET Core)

`dotnet ef` Poleceń są zawarte w zestawie SDK programu .NET Core, ale aby włączyć polecenia należy zainstalować `Microsoft.EntityFrameworkCore.Design` pakietu.

- Zainstaluj bieżącą [zestawu .NET Core SDK](#). Zestaw SDK został zainstalowany, nawet jeśli masz najnowszą wersję programu Visual Studio 2017.
- Zainstaluj najnowszy stabilny `Microsoft.EntityFrameworkCore.Design` pakietu.

```
dotnet add package Microsoft.EntityFrameworkCore.Design
```

EF Core 1.x

- Zainstaluj zestaw .NET Core SDK w wersji 2.1.200. Nowsze wersje nie są zgodne z narzędzi interfejsu wiersza polecenia dla programu EF Core 1.0 i 1.1.
- Konfigurowanie aplikacji do użycia 2.1.200 wersja zestawu SDK, modyfikując jego [global.json](#) pliku. Ten plik zwykle znajduje się w katalogu rozwiązania (po jednym powyżej projektu).

- Edytowanie pliku projektu i dodawanie `Microsoft.EntityFrameworkCore.Tools.DotNet` jako `DotNetCliToolReference` elementu. Na przykład określić najnowszej wersji 1.x: 1.1.6. Zobacz przykład pliku projektu, na końcu tej sekcji.
- Zainstaluj najnowszą wersję 1.x `Microsoft.EntityFrameworkCore.Design` pakietów, na przykład:

```
dotnet add package Microsoft.EntityFrameworkCore.Design -v 1.1.6
```

Plik projektu za pomocą obu dodać odwołania do pakietu, wygląda mniej więcej tak:

```
<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <OutputType>Exe</OutputType>
    <TargetFramework>netcoreapp1.1</TargetFramework>
  </PropertyGroup>
  <ItemGroup>
    <PackageReference Include="Microsoft.EntityFrameworkCore.Design"
      Version="1.1.6"
      PrivateAssets="All" />
  </ItemGroup>
  <ItemGroup>
    <DotNetCliToolReference Include="Microsoft.EntityFrameworkCore.Tools.DotNet"
      Version="1.1.6" />
  </ItemGroup>
</Project>
```

Odwołania do pakietu przy użyciu `PrivateAssets="All"` nie jest widoczne dla projektów odwołujących się do tego projektu. To ograniczenie jest szczególnie przydatne w przypadku pakietów, które zazwyczaj są używane tylko podczas programowania.

Weryfikowanie instalacji

Uruchom następujące polecenia, aby zweryfikować poprawne zainstalowanie narzędzi interfejsu wiersza poleceń programu EF Core:

```
dotnet restore
dotnet ef
```

Dane wyjściowe polecenia identyfikuje wersję narzędzia w użyciu:

```

 _/\_
----/ \ \
| .   \| \
|_||_| |_)  \\ \
|_|_|_| \/_| //| \\
|_|_|_| /  \\\\_\\

Entity Framework Core .NET Command-line Tools 2.1.3-rtm-32065
<Usage documentation follows, not shown.>
```

Przy użyciu narzędzi

Przed rozpoczęciem korzystania z narzędzi, trzeba utworzyć projekt startowy, lub należy ustawić środowisko.

Projekt docelowy i projekt startowy

Polecenia można znaleźć *projektu* i *projekt startowy*.

- Projekt jest także znana jako *projekt docelowy* ponieważ jest za której polecenia Dodawanie lub usuwanie plików. Domyślnie projekt w bieżącym katalogu jest projekt docelowy. Należy określić inny projekt jako projekt docelowy za pomocą `--project` opcji.
- *Projekt startowy* jest tą, która narzędzia skompilować i uruchomić. Narzędzia konieczne wykonanie kodu aplikacji w czasie projektowania, aby uzyskać informacje na temat projektu, takie jak parametry połączenia bazy danych i Konfiguracja modelu. Domyślnie projekt w bieżącym katalogu jest projekt startowy. Należy określić inny projekt jako projekt startowy, za pomocą `--startup-project` opcji.

Projekt startowy i projekt docelowy często są tym samym projekcie. Jest to typowy scenariusz, w którym są one oddzielnych projektów, gdy:

- EF Core klasy kontekstu i jednostki są w bibliotece klas .NET Core.
- Aplikacją sieci web lub aplikacji konsolowej .NET Core odwołuje się do biblioteki klas.

Istnieje również możliwość [umieść kod migracji w bibliotece klas, niezależnie od kontekstu programu EF Core](#).

Innych platform docelowych

Narzędzia interfejsu wiersza polecenia pracować z projektami .NET Core i projektów programu .NET Framework. Aplikacje, które mają modelu platformy EF Core w bibliotece klas programu .NET Standard może nie mieć platformy .NET Core lub .NET Framework projektu. Na przykład dotyczy to aplikacji platformy Xamarin i platformy uniwersalnej Windows. W takich przypadkach można utworzyć projekt aplikacji konsoli .NET Core, którego jedynym celem jest do działania jako projekt startowy dla narzędzi. Projekt może być fikcyjnego projektu bez rzeczywistego kodu — jest wymagana tylko do zapewnienia celu narzędzi.

Dlaczego jest projektem fikcyjnego wymagane? Jak wspomniano wcześniej, narzędzia konieczne wykonanie kodu aplikacji w czasie projektowania. Aby to zrobić, muszą one korzystać ze środowiska uruchomieniowego .NET Core. Jeśli model programu EF Core jest w projekcie, który jest przeznaczony dla platformy .NET Core lub .NET Framework, narzędzia programu EF Core "pożyczają" środowisko uruchomieniowe z projektu. Nie można wykonać, jeśli model programu EF Core jest w bibliotece klas programu .NET Standard. .NET Standard nie jest rzeczywistą implementacją .NET; jest określenie zestawu interfejsów API, który musi obsługiwać implementacje platformy .NET. W związku z tym .NET Standard nie jest wystarczającą dla narzędzi programu EF Core do wykonywania kodu aplikacji. Projekt zastępczy, utworzone jako projekt startowy zawiera platformy konkretnego celu, w którym narzędzia można załadować biblioteki klas .NET Standard.

Środowiska ASP.NET Core

Aby określić środowisko dla projektów ASP.NET Core, ustaw **ASPNETCORE_ENVIRONMENT** zmiennej środowiskowej przed uruchomieniem polecenia.

Typowe opcje

	OPCJA	OPIS
	<code>--json</code>	Pokaż dane wyjściowe JSON.
<code>-c</code>	<code>--context <DBCONTEXT></code>	<code>DbContext</code> Klasy. Nazwa klasy tylko lub w pełni kwalifikowana za pomocą przestrzeni nazw. Jeśli ta opcja zostanie pominięty, programem EF Core znajdzie klasy kontekstu. W przypadku wielu klas kontekstu, ta opcja jest wymagana.

	OPCJA	OPIS
<code>-p</code>	<code>--project <PROJECT></code>	Względna ścieżka do folderu projektu do projektu docelowego. Wartością domyślną jest bieżący folder.
<code>-s</code>	<code>--startup-project <PROJECT></code>	Względna ścieżka do folderu projektu Projekt startowy. Wartością domyślną jest bieżący folder.
	<code>--framework <FRAMEWORK></code>	Moniker platformy docelowej dla platformę docelową. Użyj pliku projektu określa wiele platform docelowych, gdy chcesz wybrać jeden z nich.
	<code>--configuration <CONFIGURATION></code>	Konfigurację komplikacji, na przykład: <code>Debug</code> lub <code>Release</code> .
	<code>--runtime <IDENTIFIER></code>	Identyfikator docelowe środowisko uruchomieniowe w celu przywrócenia pakietów dla. Aby uzyskać listę identyfikatorów środowisk uruchomieniowych (RID), zobacz katalogu RID .
<code>-h</code>	<code>--help</code>	Pokaż informacje pomocy.
<code>-v</code>	<code>--verbose</code>	Wyświetlić pełne dane wyjściowe.
	<code>--no-color</code>	Nie kolorować danych wyjściowych.
	<code>--prefix-output</code>	Prefiks danych wyjściowych z poziomu.

docelowej bazie danych ef DotNet

Porzuca bazy danych.

Opcje:

	OPCJA	OPIS
<code>-f</code>	<code>--force</code>	Nie Potwierdź.
	<code>--dry-run</code>	Pokaż bazę danych, która będzie można usunąć, ale nie należy usuwać jej.

Aktualizacja bazy danych programu ef DotNet

Aktualizuje bazę danych do ostatniej migracji lub określony migracji.

Argumenty:

ARGUMENT	OPIS
<MIGRATION>	Migracja docelowego. Migracje mogą być określane według nazwy lub identyfikatora. Numer 0 jest przypadkiem szczególnym, oznacza to <i>przed migracją pierwszy</i> i powoduje, że wszystkich migracji, które można przywrócić. Jeśli migracja nie zostanie określony, polecenie domyślne ostatnio migracji.

Poniższe przykłady aktualizują bazę danych do określonego migracji. Pierwszy używa nazwy migracji, a drugi używa identyfikatora migracji:

```
dotnet ef database update InitialCreate
dotnet ef database update 20180904195021_InitialCreate
```

informacje kontekstu dbcontext ef DotNet

Pobiera informacje `DbContext` typu.

Lista typu dbcontext ef DotNet

Wyświetla dostępne `DbContext` typów.

Tworzenie szkieletu dbcontext ef DotNet

Generuje kod dla `DbContext` i typy jednostek bazy danych. Aby to polecenie, aby wygenerować typ jednostki w tabeli bazy danych musi mieć klucz podstawowy.

Argumenty:

ARGUMENT	OPIS
<CONNECTION>	Parametry połączenia z bazą danych. W przypadku projektów ASP.NET Core 2.x wartość może być <i>nazwa = <nazwa parametrów połączenia ></i> . W takim przypadku nazwa pochodzi od źródła konfiguracji, które są skonfigurowane dla projektu.
<PROVIDER>	Dostawca do użycia. Zazwyczaj jest to nazwa pakietu NuGet, na przykład: <code>Microsoft.EntityFrameworkCore.SqlServer</code> .

Opcje:

	OPCJA	OPIS
-d	<code>--data-annotations</code>	Aby skonfigurować model (tam, gdzie jest to możliwe), należy użyć atrybutów. Jeśli ta opcja zostanie pominięty, jest używany tylko interfejsu API fluent.
-c	<code>--context <NAME></code>	Nazwa <code>DbContext</code> klasy do wygenerowania.

	OPCJA	OPIS
	--context-dir <PATH>	Katalog, aby umieścić <code>DbContext</code> plik klasy. Ścieżki są względne wobec katalogu projektu. Przestrzenie nazw są uzyskiwane z nazwy folderów.
-f	--force	Nadpisz istniejące pliki.
-o	--output-dir <PATH>	Katalog, który można umieścić pliki klas jednostek w. Ścieżki są względne wobec katalogu projektu.
	--schema <SCHEMA_NAME>...	Schematów tabel w celu wygenerowania typów jednostek dla. Aby określić wiele schematów, powtórz <code>--schema</code> dla każdej z nich. Jeśli ta opcja zostanie pominięty, uwzględniono wszystkich schematów.
-t	--table <TABLE_NAME> ...	Tabele, aby wygenerować typy jednostek dla. Aby określić wiele tabel, powtórz <code>-t</code> lub <code>--table</code> dla każdej z nich. Jeśli ta opcja zostanie pominięty, wszystkie tabele są uwzględniane.
	--use-database-names	Użyj nazwy tabel i kolumn dokładnie tak, jak pojawiają się w bazie danych. Jeśli ta opcja zostanie pominięty, nazwy baz danych zostały zmienione, aby ściślej odpowiadają Konwencji stylistycznych nazwy języka C#.

W poniższym przykładzie scaffolds wszystkie schematy i tabele i umieszcza nowe pliki w *modeli* folderu.

```
dotnet ef dbcontext scaffold "Server=(localdb)\mssqllocaldb;Database=Blogging;Trusted_Connection=True;"  
Microsoft.EntityFrameworkCore.SqlServer -o Models
```

Poniższy przykład szkielety mechanizmów tylko wybrane tabele i tworzy kontekst w oddzielnym folderze z określona nazwą:

```
dotnet ef dbcontext scaffold "Server=(localdb)\mssqllocaldb;Database=Blogging;Trusted_Connection=True;"  
Microsoft.EntityFrameworkCore.SqlServer -o Models -t Post --context-dir Context -c BlogContext
```

Dodaj migracji ef DotNet

Dodaje nową migrację.

Argumenty:

ARGUMENT	OPIS
<NAME>	Nazwa migracji.

Opcje:

	OPCJA	OPIS
-o	--output-dir <PATH>	Katalog (i podrzędnej przestrzeni nazw) do użycia. Ścieżki są względne wobec katalogu projektu. Wartość domyślna to "Migracja".

List migracje ef DotNet

Wyświetla listę dostępnych migracji.

Usuń migracji ef DotNet

Usuwa ostatniej migracji (wycofanie zmian w kodzie, które zostały przygotowane do migracji).

Opcje:

	OPCJA	OPIS
-f	--force	Przywróć migracji (wycofać zmiany, które zostały zastosowane do bazy danych).

skrypt migracji ef DotNet

Generuje skrypt SQL z migracji.

Argumenty:

ARGUMENT	OPIS
<FROM>	Począwszy od migracji. Migracje mogą być określone według nazwy lub identyfikatora. Numer 0 jest przypadkiem szczególnym, oznacza to <i>przed migracją pierwszy</i> . Wartość domyślna to 0.
<TO>	Końcowy migracji. Domyślnie do ostatniego migracji.

Opcje:

	OPCJA	OPIS
-o	--output <FILE>	Plik można zapisać skryptu.
-i	--idempotent	Generowanie skryptu, który może służyć w bazie danych w każdej migracji.

Poniższy przykład tworzy skrypt migracji InitialCreate:

```
dotnet ef migrations script 0 InitialCreate
```

Poniższy przykład tworzy skrypt w przypadku wszystkich migracji po migracji InitialCreate.

```
dotnet ef migrations script 20180904195021_InitialCreate
```

Dodatkowe zasoby

- [Migracje](#)
- [Odtwarzanie](#)

Tworzenie typu DbContext w czasie projektowania

27.09.2018 • 4 minutes to read • [Edit Online](#)

Niekto re polecenia EF Core Tools (na przykład [migracje](#) polecen) wymagaj pochodnej `DbContext` wystapienia, ktore ma zosta utworzony w czasie projektowania, aby moa bylo zbiera szczegolowe informacje o aplikacji typy jednostek i sposobu mapowania ich na schemat bazy danych. W wiekszo ci przypadkow jest pozadane, `DbContext` utworzone w ten sposob jest skonfigurowany w podobny sposob, jak bedzie skonfigurowane w czasie wykonywania.

Istniej rzn sposoby, narzecia, sprbuj utworzy `DbContext`:

Z uslugi aplikacji

Jeśli Twój projekt startowy aplikacji ASP.NET Core, narzecia próby uzyskania obiektu `DbContext` od dostawcy usug w aplikacji.

Narzecia najpierw sprbuj uzyska dostawc usugi za pomoc wywoania `Program.BuildWebHost()` i uzyskiwania dostpu do `IWebHost.Services` właściwoci.

NOTE

Podczas tworzenia nowej aplikacji platformy ASP.NET Core 2.0 tego punktu zaczepienia jest domylnie. W poprzednich wersjach programu EF Core i ASP.NET Core, narzecia sprbuj wywoływa `Startup.ConfigureServices` bezporednio w celu uzyskania dostawcy usug aplikacji, ale ten wzorzec, przestanie dziaa poprawnie w aplikacjach ASP.NET Core 2.0. Jeili uaktualniasz aplikacji ASP.NET Core 1.x do 2.0, moesz to zrobic zmodyfikowa swoje `Program` klasy oparte na wzorcu nowe.

`DbContext` Sam, jak i wszelkich zaleooci w jego konstruktorze musz byc zarejestrowani jako usugi w aplikacji usugodawcy. Moa to łatwo osiągnac dzieli [Konstruktor w DbContext przyjmujcej wystapienie DbContextOptions<TContext>](#) jako argument i przy użyciu `AddDbContext<TContext>` —Metoda.

Z pomoc konstruktora bez parametrów

Jeśli kontekst `DbContext` nie moa uzyska od dostawcy usug w aplikacji, wyszukaj narzecia pochodnej `DbContext` typów wewnatrz projektu. Nastpnie proby utworzenia wystapienia przy użyciu konstruktora bez parametrów. Moa to byc domylnego konstruktora, jeili `DbContext` zosta skonfigurowana przy użyciu `OnConfiguring` metody.

Z fabryki czasu projektowania

Moesz równie poinformowa narzecia sposob tworzenia usugi `DbContext` implementuj `IDesignTimeDbContextFactory<TContext>` interfejsu: Jeili klasy implementujcej interfejs ten jest odnaleziony ani w tym samym projekcie jako pochodnej `DbContext` lub w projekcie uruchamiania aplikacji, obejcie narzecia inne sposoby tworzenia `DbContext` i używania fabryki czasu projektowania, a zamiast tego.

```
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Design;
using Microsoft.EntityFrameworkCore.Infrastructure;

namespace MyProject
{
    public class BloggingContextFactory : IDesignTimeDbContextFactory<BloggingContext>
    {
        public BloggingContext CreateDbContext(string[] args)
        {
            var optionsBuilder = new DbContextOptionsBuilder<BloggingContext>();
            optionsBuilder.UseSqlite("Data Source=blog.db");

            return new BloggingContext(optionsBuilder.Options);
        }
    }
}
```

NOTE

`args` Parametru jest aktualnie używana. Brak [problemu](#) śledzenia możliwość określenia czasu projektowania argumentów z poziomu narzędzi.

Fabryka projektowania może być szczególnie przydatne, jeśli musisz skonfigurować kontekstu DbContext inaczej niż w czasie wykonywania, jeśli `DbContext` Konstruktor przyjmuje dodatkowych parametrów nie są zarejestrowane w DI, jeśli nie używasz DI wcale, lub w przypadku niektórych przyczyny nie chcesz mieć `BuildWebHost` metody w aplikacji platformy ASP.NET Core `Main` klasy.

Usługi w czasie projektowania

28.08.2018 • 2 minutes to read • [Edit Online](#)

Niektóre usługi używana przez narzędzia są używane tylko w czasie projektowania. Te usługi są zarządzane oddzielnie od usługi czasu wykonywania programu EF Core, aby uniemożliwić im wdrażana razem z aplikacją. Aby zastąpić jedną z tych usług (na przykład usługa ma generować pliki migracji), Dodaj implementację `IDesignTimeServices` na Twój projekt startowy.

```
class MyDesignTimeServices : IDesignTimeServices
{
    public void ConfigureDesignTimeServices(IServiceCollection services)
        => services.AddSingleton<IMigrationsCodeGenerator, MyMigrationsCodeGenerator>()
}
```

Entity Framework 6

14.10.2018 • 3 minutes to read • [Edit Online](#)

Entity Framework 6 (EF6) jest sprawdzonych obiektowo relacyjny mapowania (O/RM) dla platformy .NET przy użyciu wielu lat pracy nad rozwój funkcji i stabilizacją.

Jako Obiektowo, EF6 zmniejsza niezgodności impedancji między rozwiązań relacyjnych i zorientowane obiektowo, dzięki czemu deweloperzy mogą pisać aplikacje, które współpracują z danymi przechowywanymi w relacyjnej bazy danych za pomocą silnie typizowanych obiektów platformy .NET, które reprezentują Aplikacja firmy domeny i eliminując potrzebę dużą część zazwyczaj wymaga, aby napisać kod "instalację" dostępu do danych.

EF6 implementuje wiele popularnych funkcji Obiektowo:

- Mapowanie **POCO** klas jednostek, które nie są zależne od żadnych typów EF
- Automatyczne śledzenie zmian
- Rozwiązywanie tożsamości i jednostki pracy
- Eager, opóźnieniem i jawne ładowanie
- Tłumaczenie silnie typizowane zapytań za pomocą LINQ (Language INtegrated Query)
- Obsługa zaawansowanych możliwości mapowania, w tym:
 - Relacje jeden do jednego, jeden do wielu i wiele do wielu
 - Dziedziczenie (Tabela w danej hierarchii, tabelę według typu, a tabela na konkretnej klasie)
 - Typy złożone
 - Procedury składowane
- Projektant wizualny pozwala tworzyć modele jednostki.
- "Code First" środowisko do tworzenia modeli entity przez napisanie kodu.
- Modele może być wygenerowany z istniejących baz danych, a następnie ręcznie modyfikować lub można je tworzyć od podstaw i następnie używany do generowania nowych baz danych.
- Integracja z modeli aplikacji .NET Framework, w tym ASP.NET i za pomocą wiązania danych oraz platforma WPF i WinForms.
- Łączność z bazą danych na podstawie ADO.NET i wielu dostawców można podłączyć do programu SQL Server, Oracle, MySQL, SQLite, PostgreSQL, DB2 itd.

Należy użyć EF6 i EF Core?

EF Core jest bardziej nowoczesny, uproszczone i rozszerzalny wersję platformy Entity Framework jest bardzo podobne funkcje i korzyści z platformy EF6. EF Core jest pełne ponowne zapisywanie adresów i zawiera wiele nowych funkcji nie jest dostępna w EF6, mimo że nadal brakuje niektórych najbardziej zaawansowanych możliwości mapowania EF6. Należy rozważyć użycie programu EF Core w nowej aplikacji, jeśli zestaw funkcji pasuje do wymagań. [Porównanie programów EF Core i EF6](#) sprawdza, czy ten wybór bardziej szczegółowo.

Wprowadzenie

Dodaj pakiet NuGet platformy EntityFramework do projektu lub instalowanie narzędzi Entity Framework Tools for Visual Studio. Następnie obejrzyj wideo, odczytu, samouczki i zaawansowane dokumentację, aby pomóc Państwu jak najlepiej wykorzystać możliwości platformy EF6.

Wcześniejsze wersje programu Entity Framework

Jest to dokumentację dla najnowszej wersji programu Entity Framework 6, ale część informacji ma zastosowanie

również do poprzednich wersjach. Zapoznaj się z [What's New](#) i [wydania w ciągu ostatnich](#) pełną listę wydań EF i funkcje one wprowadzone.

What's New in EF6

13.09.2018 • 3 minutes to read • [Edit Online](#)

Zdecydowanie zaleca się, że umożliwia najnowszą wersję oficjalną programu Entity Framework upewnić się, że możesz korzystać z najnowszych funkcji i najwyższą trwałość. Jednak firma Microsoft należy pamiętać, że może być konieczne użycie poprzedniej wersji lub czy może chcesz poeksperymentować z nowych ulepszeń w najnowszej wersji wstępnej. Aby zainstalować określonych wersji EF, zobacz [uzyskać Entity Framework](#).

Ta strona zawiera dokumentację funkcje, które znajdują się w kolejnych wersjach.

Najnowsze wersje

EF aktualizacji narzędzi programu Visual Studio 2017 wersji 15.7

W maju 2018 r. wydaliśmy zaktualizowaną wersję narzędzia EF jako część programu Visual Studio 2017 w wersji 15.7. Zawiera usprawnienia w zakresie niektóre typowe słabe punkty:

- Poprawki dla kilku błędów dostępności interfejsu użytkownika
- Obejście regresji wydajności programu SQL Server podczas generowania modeli z istniejących baz danych #4
- Obsługa aktualizacji modeli większych modeli w programie SQL Server #185

Innym sposobem na usprawnienie w tym tej nowej wersji zestawu narzędzi platformy EF jest, że instaluje środowiskiem uruchomieniowym EF 6.2, podczas tworzenia modelu w nowym projekcie. Ze starszymi wersjami programu Visual Studio jest możliwość użycia środowiska uruchomieniowego EF 6.2 (a także żadnych poprzednich wersji EF) przez zainstalowanie odpowiedniej wersji pakietu NuGet.

EF 6.2 środowiska uruchomieniowego

Środowiskiem uruchomieniowym EF 6.2 został wydany do narzędzia NuGet w październiku 2017 r. Dziękuję w Wielkiej części wysiłek naszej społeczności uczestników projektów typu open source, EF 6.2 zawiera wiele [poprawki usterek i ulepszenia produktu](#).

Poniżej przedstawiono krótki opis listy najważniejszych zmian wpływających na środowiskiem uruchomieniowym EF 6.2:

- Zmniejsz czas uruchamiania, ładując Zakończono pierwszą modele kodu z trwała pamięć podręczna #275
- Interfejs Fluent API, aby zdefiniować indeksy #274
- DbFunctions.Like() umożliwia zapisywanie zapytań LINQ, które dadzą podobne w języku SQL #241
- Migrate.exe obsługuje teraz — opcja skryptu #240
- EF6 teraz pracować z wartości klucza, wygenerowane za pomocą sekwencji w programie SQL Server #165
- Aktualizuj listę błędów przejściowych strategii wykonywania usługi Azure SQL #83
- Usterka: Ponowieniem próby wykonania zapytania lub polecenia SQL nie powiodło się z "parametr SqlParameter jest już zawarty w innym SqlParameterCollection" #81
- Błąd: Obliczanie DbQuery.ToString() często upływie limit czasu w debuggerze #73

Przyszłe wersje

Informacje na temat przyszłych wersjach programu EF6 można znaleźć w naszej [plan](#).

Poprzednich wersjach

[Poprzednich wersjach](#) strona zawiera archiwum wszystkich wcześniejszych wersji programu EF i główne funkcje,

które zostały wprowadzone w każdej wersji.

Przyszłych wersjach programu Entity Framework

25.10.2018 • 2 minutes to read • [Edit Online](#)

Tutaj znajdziesz informacje o kolejnych wersjach programu Entity Framework. Podczas gdy większość wysiłków zespołu EF znajduje się obecnie na dodajemy nowe funkcje i ulepszenia [programu EF Core](#), planujemy nadal naprawiaj usterki ważne, wdrożenia małych ulepszenia i dołączyć kod wniesiony przez społeczność w bazie kodu podlegającej EF6.

Zwalnia po EF 6.2

Plan po EF 6.2 wersji nadal jest opracowywany. Więcej informacji na zostaną opublikowane w tym miejscu po jego udostępnieniu.

Dokonywanie aktualizacji na bieżąco

Oprócz tej strony, nowe wersje są zwykle anonsowane w [blog zespołu .NET](#) i nasze konta w usłudze Twitter [@efmagicunicorns](#) .

Przeszłymi wersjami programu Entity Framework

27.09.2018 • 25 minutes to read • [Edit Online](#)

Pierwsza wersja programu Entity Framework został wydany w 2008 roku jako część .NET Framework 3.5 SP1 i Visual Studio 2008 z dodatkiem SP1.

Począwszy od wersji EF4.1 jest dostarczana jako [pakiet NuGet platformy EntityFramework](#) — obecnie jedną z najbardziej popularnych pakietów w witrynie NuGet.org.

Od wersji 4.1 i 5.0 pakiet NuGet platformy EntityFramework rozszerzone biblioteki EF, z którymi dostarczana jako część programu .NET Framework.

Począwszy od wersji 6 EF stało się to projekt open source i również przeniesiona całkowicie poza pasmem formularz programu .NET Framework. Oznacza to, że po dodaniu pakietu NuGet w wersji 6 EntityFramework do aplikacji, otrzymujesz pełną kopię biblioteki EF, która nie zależy od usługi bits EF, które są dostarczane jako część .NET Framework. Pomogła nieco przyspieszyć tempie rozwoju i dostarczania nowych funkcji.

W czerwcu 2016 roku opublikowaliśmy EF Core 1.0. EF Core opiera się na nowe bazy kodu i został zaprojektowany jako bardziej uproszczone i rozszerzalny wersja programu EF. Obecnie programem EF Core jest głównym celem tworzenia zespołu Entity Framework w firmie Microsoft. Oznacza to są planowane żadne nowe główne funkcje dla platformy EF6. Jednak EF6 nadal jest przechowywane jako projekt open source i obsługiwanych produktów firmy Microsoft.

Poniżej przedstawiono listę poprzednich wersjach, w odwrotnej kolejności chronologicznej z informacjami na temat nowych funkcji, które zostały wprowadzone w każdej wersji.

EF 6.1.3

EF 6.1.3 środowiska wykonawczego został wydany do narzędzia NuGet w październiku 2015 r. Ta wersja zawiera tylko poprawki usterek o wysokim priorytecie, regresji zgłoszone 6.1.2 wydania. Poprawki obejmują:

- Zapytanie: Regresję w programie EF 6.1.2: operatora OUTER APPLY wprowadzone i bardziej złożone zapytania dla relacji 1:1 i klauzuli "let"
- Ukrywanie właściwości klasy podstawowej w klasie dziedziczonej TPT problem
- DbMigration.Sql zakończy się niepowodzeniem, gdy wyraz "Przejdź" znajduje się w tekście
- Utwórz flagi zgodności UnionAll i Intersect Obsługa struktur płaskich
- Zapytanie o obejmuje wiele nie działa w 6.1.2 (Praca w 6.1.1)
- "Masz błąd w składni SQL" wyjątków po uaktualnieniu z programu EF 6.1.1 i 6.1.2

EF 6.1.2

EF 6.1.2 środowiska uruchomieniowego wydanej z grudnia 2014 r. do narzędzia NuGet. Ta wersja dotyczy przede wszystkim poprawki błędów. Możemy także akceptowane kilku warte zauważenia zmian z członkami społeczności:

- **Parametry pamięci podręcznej zapytania można konfigurować na podstawie pliku app/web.configuration**

```
<entityFramework>
  <queryCache size='1000' cleaningIntervalInSeconds=' -1' />
</entityFramework>
```

- **Metody SqlFile i SqlResource na DbMigration** umożliwiają uruchamianie SQL skrypt zapisane w formie pliku lub osadzonego zasobu.

EF 6.1.1

EF 6.1.1 środowiska uruchomieniowego wydanej w czerwcu 2014 do narzędzia NuGet. Ta wersja zawiera poprawki dotyczące problemów, których wystąpiło wiele osób. Między innymi:

- Projektant: Błąd podczas otwierania EF5 edmx z precyza dziesiątna w Projektancie EF6
- Domyślne wystąpienie logikę wykrywania LocalDB nie działa w przypadku programu SQL Server 2014

EF 6.1.0

EF 6.1.0 środowiska uruchomieniowego wydanej w marcu 2014 do narzędzia NuGet. Ta pomocnicza aktualizacja obejmuje szereg istotnych nowe funkcje:

- **Narzędzia konsolidacji** zapewnia spójny sposób do utworzenia nowego modelu platformy EF. Ta funkcja rozszerza kreatora ADO.NET Entity Data Model, aby obsługiwać tworzenie modele Code First, w tym odtwarzanie z istniejącej bazy danych. Funkcje te wcześniej były dostępne w wersji Beta jakość EF Power Tools.
- **Obsługa błędów zatwierdzania transakcji** zapewnia CommitFailureHandler, co sprawia, że korzystanie z możliwości nowo wprowadzonych w celu przechwycenia operacji transakcji. CommitFailureHandler umożliwia automatyczne odzyskiwanie po awarii połączenia, przy jednoczesnym Zatwierdzanie transakcji.
- **IndexAttribute** umożliwia indeksy, aby określić, umieszczając `[Index]` atrybutu na właściwości (lub właściwości) w modelu Code First. Kod najpierw następnie utworzy odpowiedni indeks w bazie danych.
- **Mapowania publicznego interfejsu API** zapewnia dostęp do informacji EF ma na sposobie mapowania właściwości i typy kolumn i tabel w bazie danych. W poprzednich wersjach tego interfejsu API były wewnętrzne.
- **Możliwość konfigurowania interceptory za pomocą pliku App/Web.config** umożliwia interceptory do dodania bez konieczności ponownego komplikowania aplikacji.
- **System.Data.Entity.Infrastructure.Interception.DatabaseLogger** jest nowy interceptor, która ułatwia wszystkie operacje bazy danych w pliku dziennika. W połączeniu z poprzedniej funkcji, dzięki temu można łatwo włączyć rejestrowanie operacji bazy danych dla wdrożonej aplikacji, bez konieczności ponownej komplikacji.
- **Wykrywanie zmiany modelu migracje** został ulepszony, aby były szkieletu migracje bardziej precyzyjne; także udoskonalono wydajności procesu wykrywania zmian.
- **Ulepszenia wydajności** w tym operacje niższych bazy danych podczas inicjowania, optymalizacje dla porównania równości wartości null w zapytaniach LINQ szybciej wyświetlać generowania (Tworzenie modelu) w dalszych scenariuszach i bardziej wydajne materializacja śledzonych jednostek z wielu skojarzeń.

EF 6.0.2

EF 6.0.2 środowiska wykonawczego został wydany do narzędzia NuGet w grudnia 2013 r. Ta wersja poprawki nie jest ograniczona do rozwiązywania problemów, które zostały wprowadzone w wersji EF6 (regresji wydajności/zachowanie od EF5).

EF 6.0.1

EF 6.0.1 środowiska wykonawczego został wydany do narzędzia NuGet w październiku 2013 razem z programem EF 6.0.0, ponieważ zostały one osadzone w wersjach programu Visual Studio, która była zablokowana kilka miesięcy przed. Ta wersja poprawki nie jest ograniczona do rozwiązywania problemów, które zostały wprowadzone w wersji EF6 (regresji wydajności/zachowanie od EF5). Najbardziej znaczące zmiany zostały wprowadzone rozwiązać pewne problemy z wydajnością podczas rozgrzewania dla modeli EF. Jest to ważne w trakcie rozgrzewania wydajności został obszar koncentracji uwagi w EF6 i te problemy zostały Negacja niektóre

inne wzrost wydajności w EF6.

EF 6.0

EF 6.0.0 środowiska wykonawczego został wydany do narzędzia NuGet w październiku 2013. Jest to pierwsza wersja jest uwzględniana pełną środowiska uruchomieniowego EF, w [pakiet NuGet platformy EntityFramework](#) która nie zależy od bitów EF, które są częścią programu .NET Framework. Przenoszenie pozostałe części środowiska uruchomieniowego do pakietu NuGet wymagana liczba przełomowe zmiany dla istniejącego kodu. Zobacz sekcję dotyczącą [uaktualnianie do programu Entity Framework 6](#) więcej informacji na temat ręcznego kroki wymagane do uaktualnienia.

Ta wersja zawiera wiele nowych funkcji. Następujące funkcje działają w przypadku modeli utworzonych za pomocą Code First i projektancie platformy EF:

- **Zapytania asynchroniczne i Zapisz** alokowanej dla wzorca asynchronicznego opartego na zadaniach, które zostały wprowadzone w .NET 4.5.
- **Elastyczność połączenia** umożliwia automatyczne odzyskiwanie po awarii przejściowych połączenia.
- **Konfiguracja na podstawie kodu** zapewnia możliwość wykonywania konfiguracji — tradycyjnie zostało wykonane w pliku konfiguracji — w kodzie.
- **Rozpoznawanie zależności** wprowadza obsługę wspólnym lokalizatorze usług wzorzec i firma Microsoft została uwzględniona się niektóre elementy funkcji, którą można zastąpić za pomocą niestandardowych implementacji.
- **Rejestrowanie przejmowanie/SQL** zapewnia niskiego poziomu bloków konstrukcyjnych przejmowanie EF operacji za pomocą prostego rejestrowania SQL zbudowany na górze.
- **Ulepszenia testowania** ułatwiają tworzenie testu wartości podwójnej precyzji dla typu DbContext i DbSet podczas [za pomocą platformy pozorowania lub napisanie własnego testu wartości podwójnej precyzji](#).
- **Kontekst DbContext można teraz tworzyć przy użyciu DbConnection, który jest już otwarty** umożliwiająca scenariuszach, gdzie byłaby pomocne może być otwarte połączenie, podczas tworzenia kontekstu (takie jak udostępnianie połączenia między składnikami gdzie Użytkownik może nie gwarantuje stan połączenia).
- **Ulepszona obsługa transakcji** zapewnia obsługę zewnętrznej framework, jak również ulepszoną sposoby tworzenia transakcji w ramach transakcji.
- **Typy wyliczeniowe, przestrzenne i lepszą wydajność, program .NET 4.0** —, przenosząc podstawowe składniki, które wcześniej w programie .NET Framework do pakietu NuGet programu EF, możemy teraz oferować pomoc techniczną wyliczenia, typów danych przestrzennych i ulepszenia wydajności z EF5 na platformie .NET 4.0.
- **Zwiększo wydajność Enumerable.Contains w zapytaniach LINQ.**
- **Ulepszone ciepło czasu (Generowanie widoku)**, szczególnie w przypadku dużych modeli.
- **Pluralizacja podłączanych & usługi Singularization.**
- **Niestandardowe implementacje Equals i GetHashCode** w jednostce klasy są teraz obsługiwane.
- **DbSet.AddRange/RemoveRange** zapewnia sposób zoptymalizowany, aby dodać lub usunąć wiele jednostek z zestawu.
- **DbChangeTracker.HasChanges** zapewnia prosty i skuteczny sposób sprawdzić, czy są oczekujących zmian do zapisania w bazie danych.
- **SqlCeFunctions** zapewnia SQL Compact odpowiednikiem SqlFunctions.

Następujące funkcje Zastosuj tylko do Code First:

- **Niestandardowe pierwszy konwencje związane z** Zezwalaj pisania własnych Konwencji odpowiadającym, aby zapobiec powstawaniu powtarzających się konfiguracji. Firma Microsoft udostępnia prosty interfejs API konwencje uproszczone, a także niektórych bardziej złożonych bloków konstrukcyjnych umożliwia tworzenie bardziej skomplikowanych Konwencji.

- **Kod pierwszy mapowanie procedur składowanych wstawiania/aktualizowania/usuwania** jest teraz obsługiwane.
- **Skrypty migracji Idempotentne** umożliwiają generowanie skryptu SQL, które można uaktualnić bazę danych w dowolnej wersji do najnowszej wersji.
- **Tabela migracji można skonfigurować w historii** umożliwia dostosowanie definicji tabeli historii migracji. Jest to szczególnie przydatne dla dostawcy bazy danych wymagają odpowiedni typ danych itp. może być określony dla tabeli migracji historii, aby działać poprawnie.
- **Wiele kontekstów na bazę danych** usunie poprzednie ograniczenie jeden model Code First na bazę danych, korzystając z migracji lub Code First automatycznie utworzyła bazę danych dla Ciebie.
- **DbModelBuilder.HasDefaultSchema** jest nowy kod pierwszy interfejs API umożliwiający domyślny schemat bazy danych dla modelu Code First, należy skonfigurować w jednym miejscu. Wcześniej Code First schemat domyślny: stałe zakodowany "dbo" i jedynym sposobem, aby skonfigurował schemat, do której będzie należał tabela za pomocą ToTable interfejsu API.
- **Metoda DbModelBuilder.Configurations.AddFromAssembly** pozwala łatwo dodać wszystkich klas konfiguracji zdefiniowany w zestawie, w przypadku korzystania z klas konfiguracji przy użyciu kodu pierwszy Fluent interfejsu API.
- **Operacje migracji niestandardowe** umożliwia dodawanie dodatkowych operacji do użycia w migracji opartego na kodzie.
- **Domyślny poziom izolacji transakcji jest zmieniana na READ_COMMITTED_SNAPSHOT** dla baz danych utworzonych przy użyciu Code First, co zapewnia lepszą skalowalność i mniej zakleszczenia.
- **Jednostek i typów złożonych można teraz klasy nestedinside.** |

EF 5.0

EF 5.0.0 środowiska uruchomieniowego wydanej w sierpniu 2012 do narzędzia NuGet. W tej wersji wprowadzono kilka nowych funkcji, w tym obsługa wyliczenia, zwracających tabelę, typów danych przestrzennych i różne ulepszenia wydajności.

Entity Framework Designer w programie Visual Studio 2012 wprowadza również obsługę wielu — diagramy, na model, Kolorowanie kształtów na projekt powierzchni i batch importu procedur składowanych.

Oto lista zawartości, które razem utworzona Państwu specjalnie dla wersji programów EF 5.

- [Wpis w wersji 5 EF](#)
- Nowe funkcje w EF5
 - [Wyliczenie obsługi w kodzie najpierw](#)
 - [Obsługa typu wyliczeniowego w Projektancie platformy EF](#)
 - [Typy w kodzie najpierw danych przestrzennych](#)
 - [Typy danych przestrzennych w Projektancie platformy EF](#)
 - [Obsługa dostawców dla typów przestrzennych](#)
 - [Funkcje zwracające tabelę](#)
 - [Wiele diagramów na modelu](#)
- Konfigurowanie modelu
 - [Tworzenie modelu](#)
 - [Połączenia i modeli](#)
 - [Zagadnienia dotyczące wydajności](#)
 - [Praca z programem Microsoft SQL Azure](#)
 - [Ustawienia pliku konfiguracji](#)
 - [Słownik](#)
 - [Najpierw kod](#)

- Najpierw kod do nowej bazy danych (wskazówki i wideo)
- Najpierw kod istniejącej bazy danych (wskazówki i wideo)
- Konwencje
- Adnotacje danych
- Interfejs API Fluent — Konfigurowanie/mapowania typów & właściwości
- Interfejs API Fluent — Konfigurowanie relacji
- Interfejs Fluent API przy użyciu VB.NET
- Migracje Code First
- Migracje automatyczne Code First
- Migrate.exe
- Definiowanie DbSets
- Projektancie platformy EF
 - Pierwszy model (wskazówki i wideo)
 - Baza danych pierwszy (wskazówki i wideo)
 - Typy złożone
 - Skojarzenia/relacji
 - Wzorzec dziedziczenia TPT
 - Wzorzec dziedziczenia TPH
 - Zapytanie za pomocą procedur składowanych
 - Procedury składowane z wielu zestawów wyników
 - Wstawianie, aktualizowanie i usuwanie za pomocą procedur składowanych
 - Mapuj jednostki z wieloma tabelami (dzielenie jednostki)
 - Mapowanie wielu jednostek do jednej tabeli (dzielenie tabeli)
 - Definiowanie zapytań
 - Szablonów generowania kodu
 - Powracanie do obiektu ObjectContext
- Przy użyciu modelu
 - Praca z klasą DbContext
 - Wykonywanie zapytań wyszukiwania jednostek
 - Praca z relacjami
 - Trwa ładowanie powiązanych jednostek
 - Praca z danymi lokalnymi
 - N-warstwowych
 - Pierwotne zapytania SQL
 - Wzorce optymistycznej współbieżności
 - Praca z serwerów proxy
 - Automatyczne wykrywanie zmian
 - Bez śledzenia zapytań
 - Metoda Load
 - Dodaj/Dodłączanie i Stany jednostki
 - Praca z wartościami właściwości
 - Dane wiązania przy użyciu platformy WPF (Windows Presentation Foundation)
 - Danych powiązanych z WinForms (Windows Forms)

EF 4.3.1

EF 4.3.1 środowiska wykonawczego został wydany do narzędzia NuGet w lutym 2012 wkrótce po EF 4.3.0. Ta

wersja poprawki zawarte kilka poprawek w wersji 4.3 platformy EF i wprowadzono lepszą obsługę LocalDB dla klientów korzystających z EF 4.3 w programie Visual Studio 2012.

Oto lista zawartości, której umieściliśmy razem specjalnie w celu wydania EF 4.3.1, większość zawartości parametru EF 4.1 nadal obowiązuje ograniczenie do EF 4.3 także.

- [EF 4.3.1 wersji wpis w blogu](#)

EF 4.3

EF 4.3.0 środowiska uruchomieniowego wydanej w lutym 2012 do narzędzia NuGet. W tej wersji uwzględnione nową funkcję migracje Code First, która umożliwia bazy danych utworzonej przez rozwiązanie Code First przyrostowo zostać zmienione w miarę rozwoju model Code First.

Oto lista zawartości, które razem utworzona Państwu specjalnie dla wersji 4.3 platformy EF, większość zawartości parametru EF 4.1 nadal obowiązuje ograniczenie do EF 4.3 także:

- [Wpis w wersji 4.3 platformy EF](#)
- [Przewodnik 4.3 migracje oparte na kod programem EF](#)
- [Przewodnik 4.3 automatycznej migracji EF](#)

EF 4.2

EF 4.2.0 środowiska wykonawczego został wydany do narzędzia NuGet w listopad 2011. Ta wersja zawiera poprawki błędów EF 4.1.1 wydania. Ponieważ tylko zawarte w tej wersji poprawki błędów, które wystąpiły EF 4.1.2 poprawki wersji, ale firma Microsoft zgody, aby przejść do 4.2, aby umożliwić nam odbiegać od daty na podstawie numerów wersji poprawki zostały użyte podczas 4.1.x zwalnia i przyjmuje [semantycznego Versionsing](#) standardowych wersji semantycznej.

Oto lista zawartości, które razem utworzona Państwu specjalnie dla wersji EF 4.2, treść podana dla platformy EF 4.1 nadal mają zastosowanie EF 4.2 także.

- [Wpis w wersji 4.2 EF](#)
- [Pierwszy instruktaż dotyczący kodu](#)
- [Pierwszym omówieniem modelu i bazy danych](#)

EF 4.1.1

EF 4.1.10715 środowiska wykonawczego został wydany do narzędzia NuGet w lipca 2011 roku. Oprócz poprawek błędów Ta wersja poprawki wprowadzone niektórych składników, aby ułatwić czasu projektowania narzędzi do pracy z modelem Code First. Te składniki są używane przez migracje Code First (zawarte w wersji 4.3 platformy EF) i programem EF Power Tools.

Można zauważać, że otrzymano nieoczekiwany wersji numer 4.1.10715 pakietu. Użyliśmy korzystają z wersji poprawki na podstawie daty, zanim podjęliśmy decyzję o przyjęcie [Semantic Versioning](#). Pomyśl o tej wersji EF 4.1 poprawka 1 (lub EF 4.1.1).

Oto lista zawartości razem utworzona Państwu dla 4.1.1 wersji:

- [EF 4.1.1 wersji wpisu](#)

EF 4.1

EF 4.1.10331 środowiska uruchomieniowego była pierwszą mają zostać opublikowane w usłudze NuGet, w kwietnia 2011 r. W tej wersji są uwzględnione uproszczony interfejs API typu DbContext i Code First przepływu pracy.

Zauważysz, numer wersji dziwne 4.1.10331, które naprawdę powinny zostać 4.1. Ponadto istnieje 4.1.10311 wersji, które powinny zostać 4.1.0-rc ("rc" oznacza "Wersja Release candidate"). Użyliśmy korzystają z wersji poprawki na podstawie daty, zanim podjęliśmy decyzję o przyjęcie [Semantic Versioning](#).

Oto lista zawartości, które ze sobą umieściliśmy w wersji 4.1. Nadal obowiązuje ograniczenie ilości go do nowszej wersji programu Entity Framework:

- [Wpis w wersji 4.1 EF](#)
- [Pierwszy instruktaż dotyczący kodu](#)
- [Pierwszym omówieniem modelu i bazy danych](#)
- [SQL Azure Federacji i Entity Framework](#)

EF 4.0

W tej wersji zostało uwzględnione w .NET Framework 4 i Visual Studio 2010, w kwietniu 2010. Ważnych nowych funkcji w tej wersji uwzględnione POCO pomocy technicznej, mapowanie klucza obcego, ładowanie z opóźnieniem, ulepszenia testowania, generowanie kodu można dostosowywać i modelu pierwszego przepływu pracy.

Mimo że pochodzi drugie wydanie programu Entity Framework, został o nazwie 4 EF, aby było zgodne z .NET Framework w wersji dostarczanej z programem. Po tej wersji firma Microsoft pracę, udostępniając dla narzędzia NuGet programu Entity Framework i przyjęte semantycznego przechowywania wersji, ponieważ zostały firma Microsoft nie jest już powiązany z wersji programu .NET Framework.

Należy pamiętać, że niektóre kolejne wersje programu .NET Framework zostały dostarczone z istotne aktualizacje uwzględnione bitów EF. W rzeczywistości wiele nowych funkcji EF 5.0 zostały zaimplementowane jako ulepszenia na te usługi bits. Jednak aby racjonalizować z historią przechowywania wersji dla platformy EF, firma Microsoft nadal odwoływać się do bitów EF, które należą do programu .NET Framework jako środowiska uruchomieniowego EF 4.0, gdy wszystkie nowsze wersje składają się z [pakiet NuGet platformy EntityFramework](#).

EF 3.5

Początkowa wersja platformy Entity Framework została uwzględniona w .NET 3.5 z dodatkiem SP1 i Visual Studio 2008 z dodatkiem SP1, wydanej w sierpniu 2008. Ta wersja podana podstawowej Obiektywnej pomocy technicznej przy użyciu bazy danych pierwszego przepływu pracy.

Uaktualnianie do programu Entity Framework 6

07.01.2019 • 7 minutes to read • [Edit Online](#)

W poprzednich wersjach programu EF kod został podzielony między podstawowe biblioteki (przede wszystkim System.Data.Entity.dll) dostarczana jako część programu .NET Framework i poza pasmem (OOB) biblioteki (przede wszystkim EntityFramework.dll) dostarczana z pakietu NuGet. Programy EF6 przyjmuje kod z bibliotekami podstawowych i dołącza go do biblioteki OOB. Jest to niezbędne w celu umożliwienia EF ma zostać wykonane "open source" i aby można było podlegać ewolucji w różnym tempie z .NET Framework. Skutkiem tego jest aplikacji będzie konieczne można odbudować przeniesionych typów.

Powinna to być proste dla aplikacji, które korzystają z typu DbContext jako wysłane w programie EF 4.1 i nowszych. Trochę więcej pracy jest wymagane dla aplikacji, które korzystają z obiektu ObjectContext, ale nadal nie jest ciężko.

Poniżej przedstawiono listę kontrolną czynności, które należy wykonać, aby uaktualnić istniejącą aplikację platformy EF6.

1. Zainstaluj pakiet NuGet platformy EF6

Należy uaktualnić do nowego środowiska uruchomieniowego platformy Entity Framework 6.

1. Kliknij prawym przyciskiem myszy na projekt i wybierz **Zarządzaj pakietami NuGet...**
2. W obszarze **Online** wybierz opcję **EntityFramework** i kliknij przycisk **Instalacji**

NOTE

Jeśli poprzednią wersję została zainstalowany pakiet NuGet platformy EntityFramework to spowoduje uaktualnienie go do platformy EF6.

Alternatywnie można uruchomić następujące polecenie z poziomu konsoli Menedżera pakietów:

```
Install-Package EntityFramework
```

2. Upewnij się, czy odwołania do zestawów do System.Data.Entity.dll zostały usunięte

Instalowanie pakietu EF6 NuGet należy automatycznie usunąć wszystkie odwołania do System.Data.Entity z projektu.

3. Zamień wszystkie modele EF projektanta (EDMX) umożliwia generowanie kodu w wersji 6.x EF

Jeśli masz wszystkie modele utworzone za pomocą projektanta EF, należy zaktualizować szablonów generowania kodu, aby wygenerować kod zgodny EF6.

NOTE

Brak dostępnych obecnie tylko EF 6.x DbContext Generator szablonów dla programu Visual Studio 2012 i 2013.

1. Usuwanie istniejących szablonów generowania kodu. Te pliki będą zwykle miały nazwy nadane `<edmx_file_name>.tt` i `<edmx_file_name>.Context.TT` i można zagnieździć w pliku edmx w Eksploratorze rozwiązań. Szablony można wybrać w Eksploratorze rozwiązań, a następnie naciśnij klawisz **Del** klawisz, aby je usunąć.

NOTE

W projektach witryny sieci Web szablonów będzie nie być zagnieźdzony w pliku edmx, ale wymienione obok niej w Eksploratorze rozwiązań.

NOTE

W projektach VB.NET, musisz włączyć "Pokaż wszystkie pliki" można było wyświetlić pliki zagnieżdzonych szablonów.

2. Dodaj odpowiedni szablon generowanie kodu w wersji 6.x EF. Otwieranie modelu w Projektancie platformy EF, kliknij prawym przyciskiem myszy projekt powierzchni i wybierz **Dodaj element generowanie kodu...**

- Jeśli używasz interfejsu API typu DbContext (zalecane) następnie **EF 6.x DbContext Generator** będą dostępne w ramach **danych** kartę.

NOTE

Jeśli używasz programu Visual Studio 2012 należy zainstalować narzędzia EF 6, aby ten szablon. Zobacz [uzyskać Entity Framework](#) Aby uzyskać szczegółowe informacje.

- Jeśli korzystasz z interfejsu API obiektu ObjectContext to będzie konieczne wybranie **Online** kartę i wyszukaj **EF 6.x EntityObject Generator**.

3. Jeśli jakiekolwiek dostosowania są stosowane do szablonów generowania kodu, należy ponownie zastosować je do zaktualizowanych szablonów.

4. Aktualizowanie przestrzeni nazw dla wszystkich typów EF core używane

Przestrzenie nazw dla typu DbContext i Code First nie uległy zmianie. To oznacza dla wielu aplikacji, które używają programu EF 4.1 lub nowszym będzie trzeba wprowadzić zmiany.

Typów, takich jak ObjectContext, które były wcześniej System.Data.Entity.dll zostały przeniesione do nowej przestrzeni nazw. Oznacza to, że może być konieczne zaktualizowanie swoje *przy użyciu lub importu* dyrektywy algorytmie EF6.

Ogólną zasadą zmian przestrzeni nazw jest, że dowolnego typu w System.Data.* jest przenoszony do System.Data.Entity.Core.*. Innymi słowy, wystarczy wstawić **Entity.Core**. Po dane systemowe. Na przykład:

- System.Data.EntityException = > dane systemowe. **Entity.Core**. EntityException
- System.Data.Objects.ObjectContext = > dane systemowe. **Entity.Core**. Objects.ObjectContext
- System.Data.Objects.DataClasses.RelationshipManager = > dane systemowe. **Entity.Core**. Objects.DataClasses.RelationshipManager

Te typy są w Core przestrzenie nazw, ponieważ nie są używane bezpośrednio w przypadku większości aplikacji na podstawie typu DbContext. Niektóre typy, które były częścią System.Data.Entity.dll są nadal używane najczęściej i bezpośrednio do aplikacji opartych na DbContext i dlatego nie zostały przeniesione do Core przestrzeni nazw. Są

to:

- System.Data.EntityState = > dane systemowe. **Jednostki**. EntityState
- System.Data.Objects.DataClasses.EdmFunctionAttribute = > dane systemowe. **Entity.DbFunctionAttribute**

NOTE

Ta klasa została zmieniona; Klasa o starej nazwie nadal istnieje i działa, ale jej oznaczone jako przestarzałe.

- System.Data.Objects.EntityFunctions = > dane systemowe. **Entity.DbFunctions**

NOTE

Ta klasa została zmieniona; Klasa o starej nazwie nadal istnieje i działa, ale teraz oznaczone jako przestarzałe).

- Przestrzenne klasy (na przykład DbGeography, DbGeometry) zostały przeniesione z System.Data.Spatial = > dane systemowe. **Jednostki**. Przestrzenne

NOTE

Niektóre typy w przestrzeni nazw System.Data znajdują się w System.Data.dll, który nie jest zestawem EF. Te typy nie zostały przeniesione, i dlatego ich przestrzenie nazw pozostają niezmienione.

Wersje programu Visual Studio

25.10.2018 • 7 minutes to read • [Edit Online](#)

Firma Microsoft zaleca, aby zawsze używać najnowszej wersji programu Visual Studio, ponieważ zawiera najnowsze narzędzia dla platformy .NET, NuGet i Entity Framework. W rzeczywistości różne przykłady i wskazówki dotyczące różnych dokumentacji programu Entity Framework przyjęto założenie, że używasz najnowszej wersji programu Visual Studio.

Możliwe jest jednak używać starszych wersji programu Visual Studio z różnymi wersjami programu Entity Framework tak długo, jak długo potrwać do konta pewne różnice:

Visual Studio 2017 w wersji 15.7 lub nowszej

- Ta wersja programu Visual Studio zawiera najnowszą wersję narzędzia Entity Framework i środowiskiem uruchomieniowym EF 6.2 i nie jest wymagane dodatkowe ustawienia. Zobacz [What's New](#) Aby uzyskać więcej informacji o tych wersjach.
- Dodawanie programu Entity Framework do nowych projektów przy użyciu narzędzi programu EF automatycznie doda pakietu EF 6.2 NuGet. Można ręcznie zainstalować lub uaktualnić do dowolnego pakietu NuGet programu EF dostępne online.
- Domyślnie wystąpienie programu SQL Server dostępnych w tej wersji programu Visual Studio jest wystąpieniem LocalDB o nazwie MSSQLLocalDB. Sekcja serwera parametrów połączenia, należy użyć "(localdb)\MSSQLLocalDB". Pamiętaj, aby używać ciąg verbatim prefiksem @ lub podwójny ukośnik odwrotny "\\\" podczas określania parametrów połączenia w kodzie języka C#.

Visual Studio 2015, Visual Studio 2017 w wersji 15.6

- Te wersje programu Visual Studio obejmują narzędzia Entity Framework i środowisko uruchomieniowe 6.1.3. Zobacz [wydania w ciągu ostatnich](#) Aby uzyskać więcej informacji o tych wersjach.
- Dodawanie programu Entity Framework do nowych projektów przy użyciu narzędzi programu EF automatycznie doda EF 6.1.3 pakietu NuGet. Można ręcznie zainstalować lub uaktualnić do dowolnego pakietu NuGet programu EF dostępne online.
- Domyślnie wystąpienie programu SQL Server dostępnych w tej wersji programu Visual Studio jest wystąpieniem LocalDB o nazwie MSSQLLocalDB. Sekcja serwera parametrów połączenia, należy użyć "(localdb)\MSSQLLocalDB". Pamiętaj, aby używać ciąg verbatim prefiksem @ lub podwójny ukośnik odwrotny "\\\" podczas określania parametrów połączenia w kodzie języka C#.

Visual Studio 2013

- Ta wersja programu Visual Studio zawiera i starszą wersję narzędzi Entity Framework i środowiska uruchomieniowego. Zaleca się uaktualnienie do programu Entity Framework Tools 6.1.3, za pomocą [Instalator](#) dostępne w programie Microsoft Download Center. Zobacz [wydania w ciągu ostatnich](#) Aby uzyskać więcej informacji o tych wersjach.
- Dodawanie programu Entity Framework do nowych projektów przy użyciu uaktualnionego narzędzia EF automatycznie doda EF 6.1.3 pakietu NuGet. Można ręcznie zainstalować lub uaktualnić do dowolnego pakietu NuGet programu EF dostępne online.
- Domyślnie wystąpienie programu SQL Server dostępnych w tej wersji programu Visual Studio jest wystąpieniem LocalDB o nazwie MSSQLLocalDB. Sekcja serwera parametrów połączenia, należy użyć "(localdb)\MSSQLLocalDB". Pamiętaj, aby używać ciąg verbatim prefiksem @ lub podwójny ukośnik odwrotny "\\\" podczas określania parametrów połączenia w kodzie języka C#.

Visual Studio 2012

- Ta wersja programu Visual Studio zawiera i starszą wersję narzędzi Entity Framework i środowiska uruchomieniowego. Zaleca się uaktualnienie do programu Entity Framework Tools 6.1.3, za pomocą [Instalator](#) dostępne w programie Microsoft Download Center. Zobacz [wydania w ciągu ostatnich](#) Aby uzyskać więcej informacji o tych wersjach.
- Dodawanie programu Entity Framework do nowych projektów przy użyciu uaktualnionego narzędzia EF automatycznie doda EF 6.1.3 pakietu NuGet. Można ręcznie zainstalować lub uaktualnić do dowolnego pakietu NuGet programu EF dostępne online.
- Domyślnie wystąpienie programu SQL Server dostępnych w tej wersji programu Visual Studio jest wystąpienia LocalDB, wywoływana w wersji 11.0. Sekcja serwera parametrów połączenia, należy użyć "(localdb)\w w wersji 11.0". Pamiętaj, aby używać ciąg verbatim prefiksem `@` lub podwójny ukośnik odwrotny "`\\" podczas określania parametrów połączenia w kodzie języka C#.`

Visual Studio 2010

- Wersja narzędzi Entity Framework Tools dostępnych w tej wersji programu Visual Studio nie jest zgodny ze środowiskiem uruchomieniowym platformy Entity Framework 6 i nie może zostać uaktualniona.
- Domyślnie narzędzia Entity Framework doda Entity Framework 4.0 do swoich projektów. Aby tworzyć aplikacje przy użyciu dowolnej nowszych wersji EF, najpierw musisz zainstalować [rozszerzenia Menedżera pakietów NuGet](#).
- Domyślnie wszystkie generowanie kodu w wersji EF narzędzia opiera się na EntityObject i Entity Framework 4. Firma Microsoft zaleca, przełącz się generowanie kodu była oparta na DbContext i Entity Framework 5, instalując DbContext szablonów generowania kodu dla [C#](#) lub [języka Visual Basic](#).
- Po zainstalowaniu rozszerzenia Menedżera pakietów NuGet można ręcznie zainstalować lub uaktualnić do dowolnego pakietu NuGet programu EF dostępne online i za pomocą platformy EF6 Code First platformy, które nie wymagają projektanta.
- Domyślnie wystąpienie programu SQL Server dostępnych w tej wersji programu Visual Studio jest program SQL Server Express, o nazwie SQLEXPRESS. Sekcja serwera parametrów połączenia, należy użyć ".\SQLEXPRESS ". Pamiętaj, aby używać ciąg verbatim prefiksem `@` lub podwójny ukośnik odwrotny "`\\" podczas określania parametrów połączenia w kodzie języka C#.`

Wprowadzenie do platformy Entity Framework 6

13.09.2018 • 3 minutes to read • [Edit Online](#)

Ten przewodnik zawiera zbiór linków do artykułów dotyczących dokumentacji wybranej, przewodników i filmów wideo, które mogą pomóc Ci szybko rozpocząć pracę.

Podstawy

- [Pobieranie platformy Entity Framework](#)

W tym miejscu będą dowiesz się, jak dodawanie Entity Framework do aplikacji i, jeśli chcesz używać projektancie platformy EF, sprawdź, czy otrzymasz ona zainstalowana w programie Visual Studio.

- [Tworzenie modelu: kod najpierw projektancie platformy EF i przepływów pracy programu EF](#)

Wolisz do określania modelu platformy EF pisania kodu lub rysunku pól i wierszy? Będą mapowanie obiektów do istniejącej bazy danych przy użyciu programu EF lub chcesz EF, tworzenia zoptymalizowanych pod kątem obiektów bazy danych? W tym miejscu Twoje informacje o dwa różne podejścia do użycia EF6: Code First i projektancie platformy EF. Upewnij się, postępuj zgodnie z dyskusją i obejrzeć wideo dotyczące różnicy.

- [Praca z klasą DbContext](#)

DbContext jest pierwszy i najważniejszy EF typ który należy poznać sposób użycia. Jego służy jako launchpad dla zapytań bazy danych i śledzi zmiany wprowadzane do obiektów, dzięki czemu mogą zostać utrwalone w bazie danych.

- [Zadaj pytanie](#)

Dowiedz się, jak uzyskać pomoc od ekspertów i Współtworzenie odpowiedzi społeczności.

- [Współtworzenie](#)

Entity Framework 6 używa otwarty model opracowywania. Dowieź się, jak możesz pomóc ulepszyć EF, odwiedzając repozytorium GitHub.

Pierwszy zasoby kodu

- [Kod najpierw istniejący przepływ pracy bazy danych](#)
- [Najpierw kod do nowej bazy danych przepływu pracy](#)
- [Mapowanie typów wyliczeniowych najpierw przy użyciu kodu](#)
- [Mapowanie typów przestrzennych, najpierw przy użyciu kodu](#)
- [Pisanie pierwszego konwencje kod niestandardowy](#)
- [Za pomocą pierwsza Konfiguracja Fluent kodu za pomocą Visual Basic](#)
- [Migracje Code First](#)
- [Migracje Code First na środowiska zespołowe](#)
- [Automatyczne migracje Code First \(to nie jest już zalecany\)](#)

Zasoby projektanta EF

- [Baza danych pierwszego przepływu pracy](#)
- [Model pierwszego przepływu pracy](#)

- [Mapowanie typów wyliczeniowych](#)
- [Mapowanie typów przestrzennych](#)
- [Tabela wg hierarchii dziedziczenia mapowania](#)
- [Tabela wg typu dziedziczenia mapowania](#)
- [Procedura składowana mapowania aktualizacji](#)
- [Zapisane mapowanie procedur dla zapytania](#)
- [Dzielenie jednostki](#)
- [Dzielenie tabeli](#)
- [Definiowanie zapytania \(zaawansowane\)](#)
- [Funkcje zwracające tabelę \(zaawansowane\)](#)

Inne zasoby

- [Zapytania asynchroniczne i Zapisz](#)
- [Wiązanie danych z WinForms](#)
- [Powiązanie danych przy użyciu platformy WPF](#)
- [Rozłączonych scenariuszy z jednostkami Self-Tracking \(to nie jest już zalecany\)](#)

Podstawowe informacje dotyczące programu Entity Framework 6

13.09.2018 • 2 minutes to read • [Edit Online](#)

Tematy w tej sekcji opisano różne podstawowych aspektów pracy z usługą platformy EF6.

Pobieranie programu Entity Framework

13.09.2018 • 3 minutes to read • [Edit Online](#)

Entity Framework składa się z narzędziami EF dla programu Visual Studio i środowiska uruchomieniowego EF.

EF Tools for Visual Studio

Narzędzia Entity Framework Tools for Visual Studio obejmują w kreatorze modelu platformy EF i projektancie platformy EF są wymagane dla bazy danych po raz pierwszy i modelowanie pierwszy przepływów pracy.

Narzędzia platformy EF znajdują się w najnowszych wersjach programu Visual Studio. Jeśli wykonać niestandardową instalację programu Visual Studio, musisz upewnić się, że element "Entity Framework 6 Tools" jest zaznaczone, wybierając obciążenia zawierający go lub wybierając go jako poszczególnych składników.

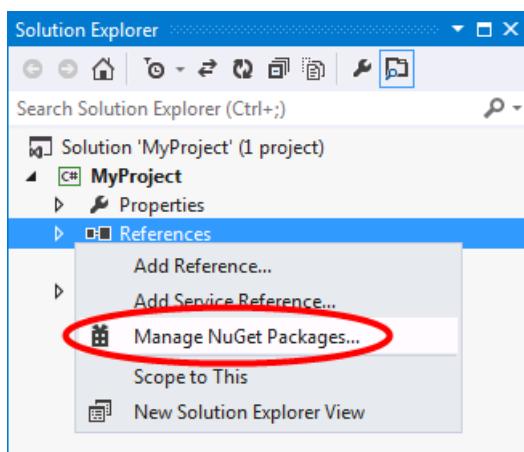
W przypadku niektórych poprzednich wersji programu Visual Studio zaktualizowanych narzędzi EF są dostępne do pobrania. Zobacz [wersje serwera Visual Studio](#) wskazówki dotyczące sposobu uzyskania najnowszej wersji programu EF narzędzi dostępnych dla używanej wersji programu Visual Studio.

Środowiska uruchomieniowego EF

Najnowszą wersję programu Entity Framework jest dostępna jako [pakiet NuGet platformy EntityFramework](#). Jeśli użytkownik nie jest zaznajomiony z Menedżerem pakietów NuGet, zachęcamy do odczytania [Przegląd NuGet](#).

Instalowanie pakietu NuGet programu EF

Można zainstalować pakiet platformy EntityFramework, klikając prawym przyciskiem myszy **odwołania** folderu projektu i wybierając polecenie **Zarządzaj pakietami NuGet...**



Instalowanie za pomocą konsoli Menedżera pakietów

Alternatywnie, można zainstalować platformy EntityFramework, uruchamiając następujące polecenie w [Konsola Menedżera pakietów](#).

```
Install-Package EntityFramework
```

Instalowanie określonej wersji programu EF

Od wersji 4.1 EF wydano nowe wersje środowiska uruchomieniowego EF jako [pakiet NuGet platformy EntityFramework](#). Dowolne z tych wersji można dodać do projektu na podstawie .NET Framework, uruchamiając następujące polecenie w programie Visual Studio [Konsola Menedżera pakietów](#):

```
Install-Package EntityFramework -Version <number>
```

Należy pamiętać, że <number> reprezentuje określoną wersję platformy EF do zainstalowania. Na przykład 6.2.0 stanowi wersję numer EF 6.2.

EF środowisk wykonawczych, zanim 4.1 były częścią środowiska .NET Framework i nie można zainstalować oddziennie.

Instalowanie najnowszej wersji zapoznawczej

Powyższych metod prześle Ci najnowsze wydania Entity Framework w pełni obsługiwane. Często są wersje wstępne programu Entity Framework jest dostępna, że chętnie można wypróbować, a następnie Prześlij nam opinię na temat.

Do zainstalowania najnowszej wersji zapoznawczej platformy EntityFramework, możesz wybrać **Uwzględnij wydania wstępne** w oknie Zarządzanie pakietami NuGet. Jeśli nie wstępnej wersji są dostępne najnowsze zostanie automatycznie pobrana w pełni obsługiwana wersja programu Entity Framework.



Alternatywnie, możesz uruchomić następujące polecenie w [Konsola Menedżera pakietów](#).

```
Install-Package EntityFramework -Pre
```

Praca z typu DbContext

13.09.2018 • 6 minutes to read • [Edit Online](#)

Aby można było używać programu Entity Framework do zapytania, wstawiania, aktualizowania i usuwania danych, używając obiektów platformy .NET, należy najpierw [utworzyć Model](#) która mapuje jednostek i relacji, które są zdefiniowane w modelu służącym do tabel w bazie danych.

Po utworzeniu modelu, jest podstawową klasą aplikacji współdziałała z `System.Data.Entity.DbContext` (często określany jako klasy kontekstu). Można użyć typu DbContext powiązanych z modelem do:

- Pisanie i wykonywania zapytań
- Zmaterializowania wyników zapytania jako obiekty jednostki
- Śledź zmiany wprowadzone do tych obiektów
- Utrwalanie zmian obiektów na bazie danych
- Powiązanie obiektów w pamięci na kontrolkę interfejsu użytkownika

Ta strona udostępnia wskazówki na temat zarządzania z klasy kontekstu.

Definiowanie klasy pochodne typu DbContext

Zalecany sposób pracy z kontekstem jest Definiowanie klasy, która pochodzi od typu DbContext i udostępnia właściwości DbSet, które reprezentują kolekcji określonej jednostki w kontekście. Jeśli pracujesz w Projektancie platformy EF kontekście zostanie wygenerowany dla Ciebie. Jeśli pracujesz z Code First, zwykle napiszesz kontekście samodzielnie.

```
public class ProductContext : DbContext
{
    public DbSet<Category> Categories { get; set; }
    public DbSet<Product> Products { get; set; }
}
```

Po utworzeniu kontekstu, należy wyszukać Dodaj (przy użyciu `Add` lub `Attach` metody) lub Usuń (przy użyciu `Remove`) jednostki w kontekście za pomocą tych właściwości. Uzyskiwanie dostępu do `DbSet` właściwość obiektu kontekstu reprezentują począwszy od zapytania, które zwraca wszystkich jednostek określonego typu. Należy zauważać, że tylko dostęp do właściwości nie będą wykonywane zapytania. Zapytanie jest wykonywane gdy:

- Są wyliczane przez `foreach` (C#) lub `For Each` — instrukcja (Visual Basic).
- Jego są wyliczane przez operację kolekcji takich jak `ToArray`, `ToDictionary`, lub `ToList`.
- Operatory LINQ, takich jak `First` lub `Any` są określone w najbardziej zewnętrznej części zapytania.
- Noszą nazwę jednej z następujących metod: `Load` metody rozszerzenia `DbEntityEntry.Reload`, `Database.ExecuteSqlCommand`, i `DbSet<T>.Find`, jeśli jednostki z określonym kluczem nie znajduje się już ładowane w kontekście.

Okres istnienia

Okres istnienia w kontekście rozpoczyna się, gdy wystąpienie zostanie utworzona i kończy się, gdy wystąpienie jest usunięty lub zebranych elementów bezużytecznych. Użyj **przy użyciu** Jeśli chcesz, aby wszystkie zasoby, które kontekstu kontrolki usuwana na końcu bloku. Kiedy używasz **przy użyciu**, kompilator automatycznie tworzy blok try/finally i wywołuje metodę dispose w **na koniec** bloku.

```
public void UseProducts()
{
    using (var context = new ProductContext())
    {
        // Perform data access using the context
    }
}
```

Poniżej przedstawiono ogólne wskazówki przy podejmowaniu decyzji o okresie istnienia w kontekście:

- Podczas pracy z aplikacjami sieci Web, należy użyć wystąpienia kontekstu na żądanie.
- Podczas pracy z Windows Presentation Foundation (WPF) lub Windows Forms, należy użyć wystąpienia kontekstu dla formularza. Dzięki temu można korzystać z funkcji śledzenia zmian zapewnia tego kontekstu.
- Jeśli wystąpienie kontekstu jest tworzony przez kontener iniekcji zależności, zazwyczaj przyczyną jest odpowiedzialność kontenera można zlikwidować kontekstu.
- Jeśli kontekst jest tworzony w kodzie aplikacji, pamiętaj, aby dysponować kontekstem, gdy nie jest już wymagany.
- Podczas pracy z kontekstem długoterminowych należy rozważyć następujące kwestie:
 - Jak załadować więcej obiektów i ich odwołania do pamięci, zmniejszenie zużycia pamięci kontekstu może szybko wzrosnąć. Może to spowodować problemy z wydajnością.
 - Kontekst nie jest bezpieczna dla wątków, w związku z tym go nie może być współużytkowany przez wiele wątków jednocześnie wykonywania pracy.
 - Jeśli wyjątek powoduje, że kontekst będzie w stanie nieodwracalny, może rozwiązać niniejszą całą aplikacji.
 - Ryzyko związane z współbieżnością problemy podczas zwiększyć wraz ze wzrostem natężenia przerwy między czasem, gdy dane są badane i aktualizowane.

Połączenia

Domyślnie w kontekście zarządza połączenia z bazą danych. Kontekst otwiera i zamyka połączenia, zgodnie z potrzebami. Na przykład kontekście otwiera połączenie, aby wykonać zapytanie, a następnie zamyka połączenia, po przetworzeniu wszystkich zestawów wyników.

Istnieją przypadki, gdy użytkownik chce mieć większą kontrolę nad, gdy połączenie otwiera i zamyka. Na przykład podczas pracy z programu SQL Server Compact, często zaleca się zachować oddzielne Otwieranie połączenia z bazą danych dla cyklu życia aplikacji w celu zwiększenia wydajności. Ten proces można zarządzać ręcznie za pomocą `Connection` właściwości.

Relacje, właściwości nawigacji i kluczy obcych

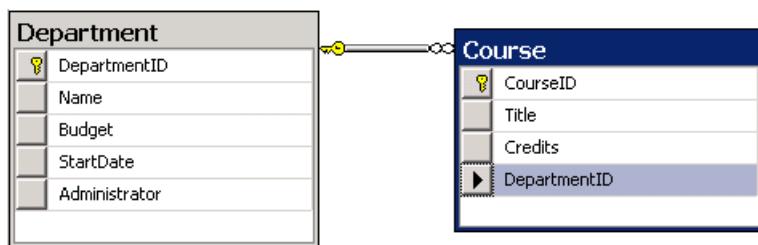
14.10.2018 • 16 minutes to read • [Edit Online](#)

Ten temat zawiera omówienie sposobu zarządzania relacjami między jednostkami w Entity Framework. Daje ona również pewne wskazówki na temat sposobu mapowania i manipulowania nimi relacji.

Relacje w programie EF

Relacyjne bazy danych (zwane również skojarzeniami) relacje między tabelami są definiowane za pomocą kluczy obcych. Klucz obcy (klucz OBCY) to kolumna lub połączenie kolumn, który służy do ustanawiania i wymuszają połączenie między danymi w dwóch tabelach. Zazwyczaj są trzy typy relacji: jeden do jednego, jeden do wielu i wiele do wielu. W relacji jeden do wielu klucz obcy jest zdefiniowany w tabeli, która reprezentuje liczbę końca relacji. Relacja wiele do wielu obejmuje zdefiniowanie trzeciej tabeli (nazywany połączeniem lub łączonym albo oczekującą tabeli), którego klucza podstawowego składa się z kluczy obcych z obu tabel powiązanych. W relacji jeden do jednego klucz podstawowy działa również jako klucz obcy, i nie ma żadnych oddzielnych kolumn klucza obcego dla każdej tabeli.

Na poniższej ilustracji przedstawiono dwie tabele, które uczestniczą w relacji jeden do wielu. **Kurs** tabela jest tabelą zależną, ponieważ zawiera on **DepartmentID** kolumny, która łączy się **działu** tabeli.



Platformy Entity Framework jednostki mogą być powiązane z innymi obiektami za pośrednictwem stowarzyszenia lub relacji. Każda relacja zawiera dwa końce, które opisują typ jednostki i liczebność typu (jednego, zero lub jeden lub wiele) dwie jednostki w tej relacji. Relacje mogą podlegać ograniczenia referencyjnego, w tym artykule opisano, które zakończenia w relacji jest główną rolę i który jest zależny roli.

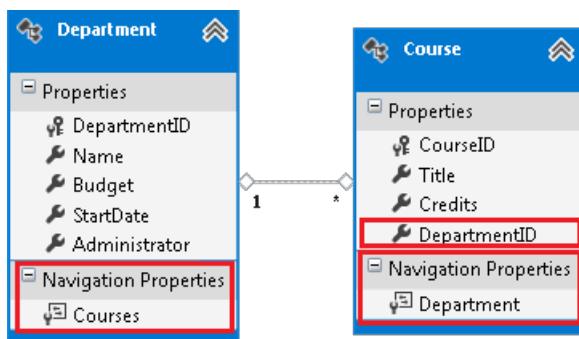
Właściwości nawigacji Podaj sposób przechodzenia skojarzenia między dwoma typami encji. Każdy obiekt może mieć właściwości nawigacji dla każdej relacji, w których uczestniczy. Właściwości nawigacji pozwala na przechodzenie relacji i zarządzanie nimi w obu kierunkach, zwracając obiekt odwołania (Jeśli liczebność jest jedną lub zero lub jeden) lub kolekcji (Jeśli liczebność to wiele). Może również wybierzesz jednokierunkowe nawigacji, w którym to przypadku zdefiniować właściwości nawigacji na tylko jeden z typów, które uczestniczy w relacji, a nie w obu.

Zalecane jest, aby uwzględnić właściwości w modelu, które są mapowane na klucze obce w bazie danych. Dołączone właściwości klucza obcego można utworzyć lub zmienić relacji, zmieniając wartość klucza obcego w obiekcie zależnym. Tego rodzaju skojarzenia nazywa się to skojarzenie klucza obcego. Przy użyciu kluczy obcych jest nawet bardziej istotne, podczas pracy z odłączoną jednostką. Należy pamiętać, że w przypadku pracy z 1-do-1 lub 1-0.. Relacje 1 jest nie oddzielne kolumny klucza obcego, właściwość klucza podstawowego działa jako klucz obcego i zawsze znajduje się w modelu.

Kolumny klucza obcego nie są uwzględnione w modelu informacji o skojarzeniu odbywa się jako niezależny obiekt. Relacje są śledzone za pomocą odwołania do obiektów zamiast właściwości klucza obcego. Skojarzenie tego typu jest nazywana *niezależnym skojarzeniem*. Najczęszym sposobem modyfikowania *niezależnych skojarzeń* jest zmodyfikowanie właściwości nawigacji, które są generowane dla każdego obiektu, który uczestniczy w skojarzeniu.

Można użyć jednego lub obu typów skojarzeń w modelu. Jednak w przypadku czystego relacji wiele do wielu, która jest połączona za pomocą tabeli sprzężenia, która zawiera tylko klucze obce EF używa niezależnych skojarzenia do zarządzania takich relacji wiele do wielu.

Na poniżej ilustracji przedstawiono model koncepcyjny, który został utworzony za pomocą programu Entity Framework Designer. Model zawiera dwie jednostki, które uczestniczą w relacji jeden do wielu. Zarówno jednostki mają właściwości nawigacji. **Kurs** jednostki depend i ma **DepartmentID** zdefiniowana właściwość klucza obcego.



Poniższy fragment kodu przedstawia ten sam model, który został utworzony za pomocą Code First.

```
public class Course
{
    public int CourseID { get; set; }
    public string Title { get; set; }
    public int Credits { get; set; }
    public int DepartmentID { get; set; }
    public virtual Department Department { get; set; }
}

public class Department
{
    public Department()
    {
        this.Course = new HashSet<Course>();
    }
    public int DepartmentID { get; set; }
    public string Name { get; set; }
    public decimal Budget { get; set; }
    public DateTime StartDate { get; set; }
    public int? Administrator { get; set; }
    public virtual ICollection<Course> Courses { get; set; }
}
```

Konfigurowanie lub Mapowanie relacji

Pozostałą części tej strony opisano, jak uzyskać dostęp i manipulowanie danymi za pomocą relacji. Aby uzyskać informacje na temat konfigurowania relacji w modelu zobacz następujące strony.

- Aby skonfigurować relacje w Code First, zobacz [adnotacje danych i Fluent API — relacje](#).
- Aby skonfigurować relacji za pomocą programu Entity Framework Designer, zobacz [relacji z projektancie platformy EF](#).

Tworzenie i modyfikowanie relacji

W skojarzenie klucza obcego, gdy zmienisz tę relację, stan obiektu zależnego za pomocą `EntityState.Unchanged` stan zmieni się na `EntityState.Modified`. W relacji niezależne zmiana relacji nie aktualizuje stan obiektu zależnego.

Poniższe przykłady pokazują, jak korzystać z właściwości klucza obcego i właściwości nawigacji do skojarzenia powiązanych obiektów. Za pomocą powiązań kluczy obcych można użyć jednej z metod można zmienić, tworzyć

lub modyfikować relacje. Za pomocą skojarzeń niezależnych nie można użyć właściwości klucza obcego.

- Przypisując nową wartość do właściwości klucza obcego, jak w poniższym przykładzie.

```
course.DepartmentID = newCourse.DepartmentID;
```

- Poniższy kod usuwa relację, ustawiając klucz obcy **null**. Należy zauważyć, że właściwość klucza obcego, musi być typ zerowalny.

```
course.DepartmentID = null;
```

NOTE

W przypadku odwołania w stanie dodany (w tym przykładzie obiekt kurs) referencyjna właściwość nawigacji nie zostaną zsynchronizowane przy użyciu wartości kluczy nowy obiekt do momentu SaveChanges jest wywoływana. Synchronizacja nie występuje, ponieważ kontekst nie zawiera stałego klucza dla dodanych obiektów, dopóki nie zostaną one zapisane. Jeśli konieczne jest posiadanie nowych obiektów w pełni zsynchronizowane tak szybko, jak ustawić relację, użyj jednej z następujących metod.*

- Przypisując nowy obiekt z właściwością nawigacji. Poniższy kod tworzy relację między kurs i `department`. Jeśli obiekty są dołączone do kontekstu, `course` jest także dodawane do `department.Courses` kolekcji i obce odpowiedniej właściwości klucza na `course` obiektu ma ustawioną wartość właściwości klucza działu.

```
course.Department = department;
```

- Aby usunąć relację, ustaw właściwość nawigacji na `null`. Pracy z platformą Entity Framework, który jest oparty na .NET 4.0 powiązanego zakończenia musi być załadowany, zanim zostanie ustawiona na wartość `null`. Na przykład:

```
context.Entry(course).Reference(c => c.Department).Load();
course.Department = null;
```

Począwszy od Entity Framework 5.0, który jest oparty na .NET 4.5, można ustawić relację `null` bez powiązaniem zakończeniem ładowania. Można również ustawić bieżącą wartość `null`, przy użyciu następującej metody.

```
context.Entry(course).Reference(c => c.Department).CurrentValue = null;
```

- Przez usunięcie lub dodanie obiektu w kolekcji jednostek. Na przykład można dodać obiektu typu `course` do `department.Courses` kolekcji. Ta operacja tworzy relację między określonego **kurs** i określonego `department`. Jeśli obiekty są przyłączone do kontekstu, odwołanie działu i właściwość klucza obcego **kurs** obiektu zostanie ustawiony na odpowiednie `department`.

```
department.Courses.Add(newCourse);
```

- Z pomocą `ChangeRelationshipState` metodę, aby zmienić stan określonej relacji między dwoma obiektami jednostki. Ta metoda jest najczęściej używana podczas pracy z usługą aplikacji N-warstwowych i *niezależnych skojarzeń* (nie można używać z skojarzenie klucza obcego). Ponadto do używania tej metody należy usunąć w dół do `ObjectContext`, jak pokazano w poniższym przykładzie.

W poniższym przykładzie istnieje relacja wiele do wielu między szkolenia i kursy. Wywoływanie

`ChangeRelationshipState` metody i przekazywanie `EntityState.Added` parametr umożliwia `SchoolContext` wiedzieć, że dodano relację między dwoma obiektami:

```
((IObjectContextAdapter)context).ObjectContext.  
    ObjectStateManager.  
    ChangeRelationshipState(course, instructor, c => c.Instructor, EntityState.Added);
```

Należy pamiętać, że Jeśli aktualizujesz (nie tylko dodanie) relację, należy usunąć stare relację po dodaniu nowego:

```
((IObjectContextAdapter)context).ObjectContext.  
    ObjectStateManager.  
    ChangeRelationshipState(course, oldInstructor, c => c.Instructor, EntityState.Deleted);
```

Synchronizowanie zmian między kluczy obcych i właściwości nawigacji

Po zmianie relacji obiektów dołączyć do kontekstu przy użyciu jednej z metod opisanych wyżej Entity Framework musi synchronizować klucze obce, odwołania i kolekcji. Entity Framework automatycznie zarządza tej synchronizacji (znany także jako relacja poprawki) dla obiektów POCO z serwerami proxy. Aby uzyskać więcej informacji, zobacz [pracy z serwerami proxy](#).

Jeśli używasz jednostki POCO bez serwera proxy należy upewnić się, że **metody DetectChanges** metoda jest wywoływana, aby zsynchronizować obiekty powiązane w kontekście. Dokonany wybór, następujące interfejsy API automatycznie wyzwoli **metody DetectChanges** wywołania.

- `DbSet.Add`
- `DbSet.AddRange`
- `DbSet.Remove`
- `DbSet.RemoveRange`
- `DbSet.Find`
- `DbSet.Local`
- `DbContext.SaveChanges`
- `DbSet.Attach`
- `DbContext.GetValidationErrors`
- `DbContext.Entry`
- `DbChangeTracker.Entries`
- LINQ do wykonywania zapytań względem `DbSet`

Trwa ładowanie powiązanych obiektów

Platformy Entity Framework, którego używasz najczęściej ładowanie jednostek that are related to jednostki zwracanego przez skojarzenie zdefiniowane przy użyciu właściwości nawigacji. Aby uzyskać więcej informacji, zobacz [ładowanie powiązanych obiektów](#).

NOTE

W to skojarzenie klucza obcego w przypadku ładowania powiązanym zakończeniem obiektu zależnego powiązany obiekt zostaną załadowane zależnie od wartości klucza obcego zależnych od, która jest aktualnie w pamięci:

```
// Get the course where currently DepartmentID = 2.  
Course course2 = context.Courses.First(c=>c.DepartmentID == 2);  
  
// Use DepartmentID foreign key property  
// to change the association.  
course2.DepartmentID = 3;  
  
// Load the related Department where DepartmentID = 3  
context.Entry(course).Reference(c => c.Department).Load();
```

Skojarzenie niezależnych powiązane koniec obiektu zależnego zostaje przesłane zapytanie, oparte na wartości klucza obcego, który jest obecnie dostępna w bazie danych. Jednak jeśli zmodyfikowano relację i spróbuje utworzyć relację jako referencyjna właściwość w punktach obiektu zależnego do innego obiektu podmiotu zabezpieczeń, który jest ładowany w kontekście obiektu programu Entity Framework jest zdefiniowana na komputerze klienckim.

Zarządzanie współbieżnością

W kluczu obcym i skojarzeniu niezależnych kontrolach współbieżności są oparte na klucze jednostek i inne właściwości jednostki, które są zdefiniowane w modelu. Tworzenie modelu za pomocą projektancie platformy EF, ustaw `ConcurrencyMode` atrybutu **stalej** do określenia, czy właściwość powinna być sprawdzana współbieżność.

Podczas korzystania Code First do definiowania modelu należy używać `ConcurrencyCheck` adnotacja dla właściwości, które mają być sprawdzone współbieżności. Podczas pracy z usługą Code First umożliwia także `TimeStamp` adnotacji, aby określić, czy właściwość powinna być sprawdzana współbieżność. Może mieć tylko jedną właściwość sygnatury czasowej w danej klasie. Mapy kodu najpierw tę właściwość z polem wartości null w bazie danych.

Firma Microsoft zaleca, zawsze używaj skojarzenie klucza obcego podczas pracy z obiektami, które uczestniczą w sprawdzaniu współbieżności i rozwiązywanie.

Aby uzyskać więcej informacji, zobacz [Obsługa konfliktów współbieżności](#).

Praca z nakładającymi się kluczami

Nakładające się klucze są klucze złożone, gdzie niektóre właściwości klucza są również częścią innego klucza w jednostce. Skojarzenie niezależne, nie może mieć nakładających się klucza. Aby zmienić to skojarzenie klucza obcego obejmującą nakładającymi się kluczami, zaleca się zmodyfikowanie wartości klucza obcego zamiast odwołania do obiektu.

Asynchroniczne zapytania i Zapisz

07.01.2019 • 10 minutes to read • [Edit Online](#)

NOTE

EF6 począwszy tylko — funkcje, interfejsów API itp. z opisem na tej stronie zostały wprowadzone w programie Entity Framework 6. Jeśli używasz starszej wersji, niektóre lub wszystkie informacje, nie ma zastosowania.

EF6 wprowadzono obsługę dla zapytania asynchroniczne, a następnie Zapisz przy użyciu **async i await** słowa kluczowe wprowadzone w .NET 4.5. Gdy nie wszystkie aplikacje mogą korzystać z asynchroniczności, może służyć do poprawy skalowalności krótki czas reakcji i serwera klienta podczas obsługi długotrwałych, sieci lub zadania I/O-powiązane z.

Kiedy należy używać naprawdę async

Ten przewodnik ma na celu wprowadzenie pojęcia asynchronicznej w sposób, który ułatwia obserwować różnicę między wykonywania programu synchronicznego i asynchronicznego. W tym przewodniku nie jest przeznaczona do zilustrowanie jednego z kluczowych scenariuszy, których programowanie async oferuje korzyści.

Programowanie asynchroniczne koncentruje się głównie na zwalnianie bieżącego wątku zarządzanych (wątek uruchamianie kodu platformy .NET), wykonywanie innych zadań, aż do operacji, która nie wymaga żadnych czas obliczeń w wątków zarządzanych. Na przykład podczas gdy aparat bazy danych jest przetwarzanie kwerendy nie ma nic do można przeprowadzić za pomocą kodu platformy .NET.

W aplikacjach klienckich (WinForms, WPF, itd.) bieżący wątek może służyć do zachowania dynamicznego interfejsu użytkownika podczas operacji asynchronicznej. W aplikacji serwera (ASP.NET itp.), wątek może służyć do przetwarzania innych żądań przychodzących — to może zmniejszyć użycie pamięci i/lub zwiększyć przepływność serwera.

W większości aplikacji przy użyciu async będzie zawierać żadnych zauważalnego korzyści, a nawet może być szkodliwe. Umożliwia testy, profilowanie i zdroworozsądowe mierzenie wpływu asynchronicznych w konkretnym scenariuszu przed zatwierdzeniem do niego.

Poniżej przedstawiono niektóre inne zasoby, aby dowiedzieć się więcej na temat async:

- [Omówienie Brandon Bray async/await w .NET 4.5](#)
- [Programowanie asynchroniczne](#) stron w bibliotece MSDN
- [Jak tworzyć ASP.NET sieci Web aplikacji przy użyciu Async](#) (w tym pokaz serwera większą przepływność)

Tworzenie modelu

Będziemy używać [Code First przepływu pracy](#) utworzyć nasz model i wygenerować bazę danych, jednak w asynchronicznej funkcji będzie działać z wszystkich modeli EF, w tym te, utworzone za pomocą projektanta EF.

- Utwórz aplikację Konsolową i wywołać go **AsyncDemo**
- Dodaj pakiet NuGet platformy EntityFramework
 - W Eksploratorze rozwiązań kliknij prawym przyciskiem myszy **AsyncDemo** projektu
 - Wybierz **Zarządzaj pakietami NuGet...**
 - W oknie dialogowym pakiety zarządzania NuGet wybierz **Online** kartę i wybierz polecenie **EntityFramework** pakietu

- Kliknij przycisk **instalacji**
- Dodaj **Model.cs** klasy następującą implementacją

```
using System.Collections.Generic;
using System.Data.Entity;

namespace AsyncDemo
{
    public class BloggingContext : DbContext
    {
        public DbSet<Blog> Blogs { get; set; }
        public DbSet<Post> Posts { get; set; }
    }

    public class Blog
    {
        public int BlogId { get; set; }
        public string Name { get; set; }

        public virtual List<Post> Posts { get; set; }
    }

    public class Post
    {
        public int PostId { get; set; }
        public string Title { get; set; }
        public string Content { get; set; }

        public int BlogId { get; set; }
        public virtual Blog Blog { get; set; }
    }
}
```

Utwórz program synchroniczne

Teraz, gdy mamy już modelu platformy EF, umożliwia pisanie kodu w celu zastosowania do wykonania niektórych dostęp do danych.

- Zastąp zawartość **Program.cs** następującym kodem

```

using System;
using System.Linq;

namespace AsyncDemo
{
    class Program
    {
        static void Main(string[] args)
        {
            PerformDatabaseOperations();

            Console.WriteLine("Quote of the day");
            Console.WriteLine(" Don't worry about the world coming to an end today... ");
            Console.WriteLine(" It's already tomorrow in Australia.");

            Console.WriteLine();
            Console.WriteLine("Press any key to exit...");
            Console.ReadKey();
        }

        public static void PerformDatabaseOperations()
        {
            using (var db = new BloggingContext())
            {
                // Create a new blog and save it
                db.Blogs.Add(new Blog
                {
                    Name = "Test Blog #" + (db.Blogs.Count() + 1)
                });
                Console.WriteLine("Calling SaveChanges.");
                db.SaveChanges();
                Console.WriteLine("SaveChanges completed.");

                // Query for all blogs ordered by name
                Console.WriteLine("Executing query.");
                var blogs = (from b in db.Blogs
                            orderby b.Name
                            select b).ToList();

                // Write all blogs out to Console
                Console.WriteLine("Query completed with following results:");
                foreach (var blog in blogs)
                {
                    Console.WriteLine(" " + blog.Name);
                }
            }
        }
    }
}

```

Ten kod wywołuje **PerformDatabaseOperations** metodę, która zapisuje nową **Blog** do bazy danych, a następnie pobiera wszystkie **blogi** z bazy danych i wysłania ich do **Konsoli**. Dzięki temu program zapisuje oferty dnia, aby **konsoli**.

Ponieważ kod jest synchroniczne, możemy zaobserwować następujący przepływ wykonania, gdy Uruchamiamy program:

1. **SaveChanges** rozpoczyna się wypychania nowego **blogu** do bazy danych
2. **SaveChanges** kończy
3. Zapytanie o wszystkie **blogi** są wysyłane do bazy danych
4. Zapytanie zwraca, a wyniki są zapisywane do **konsoli**
5. Cytat dnia są zapisywane do **konsoli**

```
Calling SaveChanges.  
SaveChanges completed.  
Executing query.  
Query completed with following results:  
- Test Blog #1  
Quote of the day  
Don't worry about the world coming to an end today...  
It's already tomorrow in Australia.  
Press any key to exit...
```

Dzięki czemu asynchroniczne

Teraz, gdy nasz program działa, możemy rozpoczęć co użycie nowego `async` i `await` słów kluczowych.

Wprowadziliśmy następujące zmiany do pliku Program.cs

1. Wiersz 2: Za pomocą instrukcji `for System.Data.Entity` przestrzeni nazw daje nam dostęp do metod rozszerzenia asynchronicznych EF.
2. Wiersz 4: Za pomocą instrukcji `for System.Threading.Tasks` przestrzeni nazw pozwala nam korzystać **zadań** typu.
3. Wiersz 12 i 18: Firma Microsoft jest przechwytywany jako zadanie, które monitoruje postęp **PerformSomeDatabaseOperations** (wierszem 12) i następnie blokuje wykonywanie programu w tym zadań raz ukończone wszystkie prace dla programu odbywa się (wiersz 18).
4. Wiersz 25: Udostępniliśmy aktualizacji **PerformSomeDatabaseOperations** być oznaczony jako **async** i zwracają **zadań**.
5. Wiersz 35: Teraz możemy wywołanie asynchroniczne wersję `SaveChanges` i oczekiwaniem na jej ukończenie.
6. Wiersz 42: Teraz dwonimy wersji asynchronicznej `tolist` — i oczekiwaniem na wynik.

Pełną listę metod rozszerzenia dostępne w przestrzeni nazw `System.Data.Entity` można znaleźć klasy `QueryableExtensions`. Należy także dodać "za pomocą `System.Data.Entity`" do sieci za pomocą instrukcji.

```

using System;
using System.Data.Entity;
using System.Linq;
using System.Threading.Tasks;

namespace AsyncDemo
{
    class Program
    {
        static void Main(string[] args)
        {
            var task = PerformDatabaseOperations();

            Console.WriteLine("Quote of the day");
            Console.WriteLine(" Don't worry about the world coming to an end today... ");
            Console.WriteLine(" It's already tomorrow in Australia.");

            task.Wait();

            Console.WriteLine();
            Console.WriteLine("Press any key to exit...");
            Console.ReadKey();
        }

        public static async Task PerformDatabaseOperations()
        {
            using (var db = new BloggingContext())
            {
                // Create a new blog and save it
                db.Blogs.Add(new Blog
                {
                    Name = "Test Blog #" + (db.Blogs.Count() + 1)
                });
                Console.WriteLine("Calling SaveChanges.");
                await db.SaveChangesAsync();
                Console.WriteLine("SaveChanges completed.");

                // Query for all blogs ordered by name
                Console.WriteLine("Executing query.");
                var blogs = await (from b in db.Blogs
                                   orderby b.Name
                                   select b).ToListAsync();

                // Write all blogs out to Console
                Console.WriteLine("Query completed with following results:");
                foreach (var blog in blogs)
                {
                    Console.WriteLine(" - " + blog.Name);
                }
            }
        }
    }
}

```

Teraz, gdy kod jest asynchroniczna, można zaobserwować przepływu wykonywania różnych gdy Uruchamiamy program:

1. **SaveChanges** rozpoczyna się wypychania nowego **Blog** w bazie danych *po wysłaniu polecenia do bazy danych co obliczenia, konieczna jest w bieżącym wątku zarządzanych.* **PerformDatabaseOperations** metoda zwróci wartość (nawet jeśli nie zostało zakończone, wykonując) i kontynuuje przepływu programu dla metody *Main*.
2. **Cytat dnia są zapisywane do konsoli** *ponieważ nie ma więcej pracy w metody *Main*, wątków zarządzanych jest zablokowany na czas oczekiwania wywołania do momentu ukończenia operacji bazy danych. Po zakończeniu pozostała część naszego **PerformDatabaseOperations** * zostaną wykonane.

3. **SaveChanges** kończy
4. Zapytanie o wszystkie **blogi** są wysyłane do bazy danych *ponownie wątków zarządzanych jest bezpłatna wykonywanie innych zadań, podczas gdy zapytania są przetwarzane w bazie danych. Od wszystkich innych wykonanie zostało ukończone, wątek będzie po prostu zatrzymanie przy wywołaniu oczekiwania jednak.*
5. Zapytanie zwraca, a wyniki są zapisywane do **konsoli**



```
Calling SaveChanges.
Quote of the day
  Don't worry about the world coming to an end today...
  It's already tomorrow in Australia.
SaveChanges completed.
Executing query.
Query completed with following results:
  - Test Blog #1
  - Test Blog #2

Press any key to exit...
```

Wnioskiem

Teraz widzieliśmy, jak łatwo jest zapewnienie użycie metod asynchronicznych EF firmy. Mimo że zalety asynchronicznych może nie być jasne, za pomocą aplikacji konsoli proste, strategie tego samego można stosować w sytuacjach, w której długotrwałych lub powiązane z siecią działania mogą w przeciwnym razie zablokować aplikacji lub spowodować dużej liczby wątków do Zwiększać ilość pamięci zajmowaną.

Konfiguracja na podstawie kodu

13.09.2018 • 7 minutes to read • [Edit Online](#)

NOTE

EF6 począwszy tylko — funkcje, interfejsów API itp. z opisem na tej stronie zostały wprowadzone w programie Entity Framework 6. Jeśli używasz starszej wersji, niektóre lub wszystkie informacje, nie ma zastosowania.

Konfigurację dla aplikacji platformy Entity Framework można określić w pliku konfiguracji (app.config/web.config) lub za pomocą kodu. Który jest znany jako Konfiguracja na podstawie kodu.

Konfiguracja w pliku konfiguracji jest opisana w [oddzielnym artykule](#). Plik konfiguracji, który ma pierwszeństwo przed konfiguracją na podstawie kodu. Innymi słowy Jeśli opcja konfiguracji jest ustawiona, zarówno kod i w pliku konfiguracji, następnie ustawienie w pliku konfiguracji jest używany.

Za pomocą DbConfiguration

Konfiguracja oparta na kod w EF6 i nowszych można uzyskać, tworząc podklasę `System.Data.Entity.Config.DbConfiguration`. Poniższe wskazówki dotyczą sytuacji, gdy Tworzenie podklasy `DbConfiguration`:

- Utwórz tylko jedną klasę `DbConfiguration` dla aplikacji. Ta klasa określa ustawienia dla całego domeny aplikacji.
- Umieść klasy `DbConfiguration` z tego samego zestawu jako swojej klasy `DbContext`. (Zobacz [przenoszenie `DbConfiguration` sekcji](#), jeśli chcesz to zmienić.)
- Nadaj swojej klasie `DbConfiguration` publicznego konstruktora bez parametrów.
- Ustawianie opcji konfiguracji, wywołując `DbConfiguration` metody chronionej z w ramach tego konstruktora.

Następujące wskazówki umożliwia EF wykrywanie i używanie konfigurację automatycznie, zarówno narzędzi, który ma dostęp do modelu i uruchamiania aplikacji.

Przykład

Klasy pochodzącej od `DbConfiguration` może wyglądać następująco:

```
using System.Data.Entity;
using System.Data.Entity.Infrastructure;
using System.Data.Entity.SqlServer;

namespace MyNamespace
{
    public class MyConfiguration : DbConfiguration
    {
        public MyConfiguration()
        {
            SetExecutionStrategy("System.Data.SqlClient", () => new SqlAzureExecutionStrategy());
            SetDefaultConnectionFactory(new LocalDBConnectionFactory("mssqllocaldb"));
        }
    }
}
```

Ta klasa konfiguruje EF, aby użyć strategii wykonywania SQL Azure — do automatycznego ponawiania próby

wykonania operacji bazy danych nie powiodło się — i Użyj lokalnej bazy danych dla baz danych, które są tworzone zgodnie z Konwencją z Code First.

Przenoszenie DbConfiguration

Istnieją przypadki, w którym nie jest możliwe do umieszczenia na klasę DbConfiguration z tego samego zestawu jako swojej klasy DbContext. Na przykład masz dwie klasy DbContext, każdy w różnych zestawach. Dostępne są dwie opcje do obsługi tego.

Pierwszym z nich jest określenie wystąpienia DbConfiguration do użycia przy użyciu pliku konfiguracji. Aby to zrobić, należy ustawić atrybut codeConfigurationType sekcji platformy entityFramework. Na przykład:

```
<entityFramework codeConfigurationType="MyNamespace.MyDbConfiguration, MyAssembly">
    ...Your EF config...
</entityFramework>
```

Wartość codeConfigurationType musi być zestawu i przestrzeni nazw kwalifikowana nazwa klasy DbConfiguration.

Drugą opcją jest umieścić DbConfigurationTypeAttribute na klasie kontekstu. Na przykład:

```
[DbConfigurationType(typeof(MyDbConfiguration))]
public class MyContextContext : DbContext
{}
```

Wartość przekazywana do atrybutu mogą być typu DbConfiguration — jak pokazano powyżej — lub zestawu i przestrzeni nazw kwalifikowana ciąg nazwy typu. Na przykład:

```
[DbConfigurationType("MyNamespace.MyDbConfiguration, MyAssembly")]
public class MyContextContext : DbContext
{}
```

Jawne ustawianie DbConfiguration

Istnieją sytuacje, w których konfiguracji może być wymagane przed dowolnego typu DbContext został już użyty. Przykłady to między innymi:

- Za pomocą DbModelBuilder w celu zbudowania modelu bez kontekstu
- Przy użyciu innych framework/narzędzie kodu korzystającej z typu DbContext, gdzie tego kontekstu jest używana, zanim kontekst aplikacji jest używana

W takich sytuacjach EF jest nie można automatycznie wykryć konfiguracji i zamiast tego należy wykonać jedną z następujących czynności:

- Ustaw typ DbConfiguration w pliku konfiguracji, zgodnie z opisem w *przenoszenie DbConfiguration* powyżej sekcji
- Wywołanie metody statycznej DbConfiguration.SetConfiguration podczas uruchamiania aplikacji

Zastępowanie DbConfiguration

Istnieją sytuacje, w których trzeba zastąpić konfigurację w DbConfiguration. Nie jest to zazwyczaj wykonywane przez deweloperów aplikacji, ale raczej przez dostawców innych firm i wtyczek, które nie mogą używać klasy pochodnej DbConfiguration.

W tym celu EntityFramework umożliwia program obsługie zdarzeń do zarejestrowania, które można zmodyfikować istniejącą konfigurację, po prostu, zanim zostanie zablokowane w dół. Zapewnia także metody sugar specjalnie w celu zastąpienia dowolnej usługi zwrócony przez EF wspólnym lokalizatorze usług. Jest to, jak jest przeznaczony do użycia:

- Przy uruchamianiu aplikacji (przed użyciem EF) wtyczki lub dostawcy należy zarejestrować metody obsługi zdarzeń dla tego zdarzenia. (Zwróć uwagę, że to musi się zdarzyć, zanim aplikacja używa EF).
- Program obsługi zdarzeń wywołuje ReplaceService dla każdej usługi, który ma zostać zastąpione.

Na przykład repalce IDbConnectionFactory i DbProviderService należy zarejestrować program obsługi podobnie do następującej:

```
DbConfiguration.Loaded += (_ , a) =>
{
    a.ReplaceService<DbProviderServices>((s, k) => new MyProviderServices(s));
    a.ReplaceService<IDbConnectionFactory>((s, k) => new MyConnectionFactory(s));
};
```

W kodzie powyżej MyProviderServices i MyConnectionFactory reprezentują usługi implementacji usługi.

Można również dodać dodatkową zależność programów obsługi, aby uzyskać ten sam efekt.

Należy pamiętać, że DbProviderFactory można również opakować w ten sposób, ale spowoduje to więc mają wpływ tylko na EF i nie używa DbProviderFactory poza EF. Z tego powodu należy prawdopodobnie opakować DbProviderFactory, jak mają przed w dalszym ciągu.

Należy również mieć na uwadze usługi, które uruchamiasz zewnętrznie do Twojej aplikacji — na przykład podczas uruchamiania migracji z konsoli Menedżera pakietów. Po uruchomieniu migracji z konsoli, spróbuje ona znaleźć Twoje DbConfiguration. Niezależnie od tego czy pobierze opakowana usługi zależy jednak gdzie on zarejestrowany program obsługi zdarzeń. Jeśli jest zarejestrowany w ramach konstrukcji swoje DbConfiguration kod powinien zostać wykonany, a powinien pobrać opakowane usługi. Zazwyczaj nie będzie to mieć miejsce, i oznacza to, że narzędzia nie będą otrzymywać opakowana usługi.

Ustawienia pliku konfiguracji

29.09.2018 • 11 minutes to read • [Edit Online](#)

Entity Framework umożliwia wiele ustawień, należy określić w pliku konfiguracji. Ogólnie rzecz biorąc EF jest zgodna z zasadą "Konwencja, za pośrednictwem konfiguracji": wszystkie ustawienia, które są omawiane w tym wpisie ma domyślne zachowanie, musisz się martwić o zmianę ustawienia, gdy domyślny nie spełnia wymagań.

To oparte na kodzie alternatywa

Wszystkie te ustawienia można również będą stosowane przy użyciu kodu. Począwszy od platformy EF6 wprowadziliśmy [konfiguracja na podstawie kodu](#), który zapewnia centralny sposób stosowania konfiguracji z poziomu kodu. Przed EF6 nadal można zastosować konfiguracji z kodu, ale trzeba skonfigurować różne obszary za pomocą różnych interfejsów API. Opcja pliku konfiguracji umożliwia te ustawienia można łatwo zmienić podczas wdrażania bez aktualizowania kodu.

Sekcja konfiguracji programu Entity Framework

Uruchamianie EF4.1 można ustawić inicjator bazy danych przy użyciu kontekstu **appSettings** sekcję pliku konfiguracji. W wersji 4.3 platformy EF wprowadziliśmy niestandardowej **entityFramework** sekcji, aby obsługiwać nowe ustawienia. Entity Framework nadal będzie także rozpoznawał inicjatory bazy danych można ustawić przy użyciu starego formatu, ale zaleca się przejście na nowy format, gdzie to możliwe.

EntityFramework sekcji została automatycznie dodana do pliku konfiguracji projektu, po zainstalowaniu pakiet NuGet platformy EntityFramework.

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <configSections>
    <!-- For more information on Entity Framework configuration, visit http://go.microsoft.com/fwlink/?LinkID=237468 -->
    <section name="entityFramework"
      type="System.Data.Entity.Internal.ConfigFile.EntityFrameworkSection, EntityFramework, Version=4.3.0.0,
      Culture=neutral, PublicKeyToken=b77a5c561934e089" />
  </configSections>
</configuration>
```

Parametry połączenia

Ta strona więcej szczegółów na temat jak Entity Framework określa bazy danych ma być używana, w tym parametry połączenia w pliku konfiguracji.

Parametry połączenia, przejdź w standardzie **connectionStrings** elementu i nie wymagają **entityFramework** sekcji.

Modele kodu najpierw na podstawie Użyj normalnej parametry połączenia ADO.NET. Na przykład:

```
<connectionStrings>
  <add name="BlogContext"
    providerName="System.Data.SqlClient"
    connectionString="Server=.\SQLEXPRESS;Database=Blogging;Integrated Security=True;" />
</connectionStrings>
```

Projektancie platformy EF na podstawie parametrów połączenia platformy EF specjalne użycia modeli. Na przykład:

```
<connectionStrings>
  <add name="BlogContext"
    connectionString=
      "metadata=
        res://*/BloggingModel.csdl|
        res://*/BloggingModel.ssdl|
        res://*/BloggingModel.msl;
      provider=System.Data.SqlClient
      provider connection string=
        "data source=(localdb)\mssqllocaldb;
        initial catalog=Blogging;
        integrated security=True;
        multipleactiveresultsets=True;"
      providerName="System.Data.EntityClient" />
</connectionStrings>
```

Typ konfiguracji oparte na kodzie (od wersji EF6)

Począwszy od platformy EF6, można określić DbConfiguration na platformie EF na potrzeby [konfiguracja na podstawie kodu](#) w aplikacji. W większości przypadków nie trzeba określić tego ustawienia, jak EF automatycznie wykryje Twoje DbConfiguration. Zobacz szczegółowo po użytkowniku może być konieczne określenie DbConfiguration w pliku config **przenoszenie DbConfiguration** części [konfiguracja na podstawie kodu](#).

Aby ustawić typ DbConfiguration, należy określić nazwę typu kwalifikowanego zestawu w **codeConfigurationType** elementu.

NOTE

Kwalifikowana nazwa zestawu jest kwalifikowana nazwa przestrzeni nazw, a następnie przecinek, następnie zestawu, który typ, który znajduje się w. Opcjonalnie możesz również określić zestaw wersji, kulturę i token klucza publicznego.

```
<entityFramework codeConfigurationType="MyNamespace.MyConfiguration, MyAssembly">
</entityFramework>
```

Dostawcy baz danych EF (od wersji EF6)

Przed EF6, musiała być dołączane jako część dostawcy ADO.NET core Framework określonej części dostawcy bazy danych. Począwszy od platformy EF6 EF określone fragmenty są teraz zarządzane i zarejestrowane oddziennie.

Zwykle nie trzeba samodzielnie zarejestrować dostawców. To są zwykle wykonywane przez dostawcę po jego zainstalowaniu.

Dostawcy są rejestrowane przez dołączenie **dostawcy** pod **dostawców** części podrzędnej **entityFramework** sekcji. Istnieją dwa atrybuty wymagane dla wpisu dostawcy:

- **Invaliantname** identyfikuje dostawcy ADO.NET core że EF dostawcy cele
- **Typ** jest kwalifikowaną nazwą typu zestawu EF implementacji dostawcy

NOTE

Kwalifikowana nazwa zestawu jest kwalifikowana nazwa przestrzeni nazw, a następnie przecinek, następnie zestawu, który typ, który znajduje się w. Opcjonalnie możesz również określić zestaw wersji, kulturę i token klucza publicznego.

Na przykład Oto wpis utworzone w celu zarejestrowania domyślnej dostawcy programu SQL Server po zainstalowaniu programu Entity Framework.

```
<providers>
  <provider invariantName="System.Data.SqlClient" type="System.Data.Entity.SqlServer.SqlProviderServices,
    EntityFramework.SqlServer" />
</providers>
```

Interceptory (EF6.1 lub nowszy)

Uruchamianie EF6.1 można zarejestrować interceptory w pliku konfiguracji. Interceptory umożliwiają uruchamianie dodatkowej logiki, gdy EF wykonuje niektóre operacje, takie jak wykonywanie kwerend bazy danych otwarcia połączeń, itp.

Interceptory są rejestrowane przez dołączenie **interceptor** pod **interceptors** części podrzędnej **entityFramework** sekcji. Na przykład następująca konfiguracja rejestruje wbudowane **DatabaseLogger** interceptor, która zarejestruje wszystkie operacje bazy danych do konsoli.

```
<interceptors>
  <interceptor type="System.Data.Entity.Infrastructure.Interception.DatabaseLogger, EntityFramework"/>
</interceptors>
```

Rejestrowanie operacji bazy danych do pliku (EF6.1 lub nowszy)

Rejestrowanie interceptory za pomocą pliku konfiguracji jest szczególnie przydatne w przypadku, gdy użytkownik chce dodać rejestrowania do istniejącej aplikacji, aby pomóc w debugowaniu problemu. **DatabaseLogger** obsługuje rejestrowanie do pliku przez podanie nazwy pliku jako parametr konstruktora.

```
<interceptors>
  <interceptor type="System.Data.Entity.Infrastructure.Interception.DatabaseLogger, EntityFramework">
    <parameters>
      <parameter value="C:\Temp\LogOutput.txt"/>
    </parameters>
  </interceptor>
</interceptors>
```

Domyślnie to spowoduje, że plik dziennika zostaną zastąpione za pomocą nowego pliku każdym uruchomieniu aplikacji. Aby dołączyć do dziennika pliku Jeśli już istnieje Użyj podobny do:

```
<interceptors>
  <interceptor type="System.Data.Entity.Infrastructure.Interception.DatabaseLogger, EntityFramework">
    <parameters>
      <parameter value="C:\Temp\LogOutput.txt"/>
      <parameter value="true" type="System.Boolean"/>
    </parameters>
  </interceptor>
</interceptors>
```

Aby uzyskać dodatkowe informacje na temat **DatabaseLogger** i rejestrowanie interceptory, zobacz wpis w blogu [EF 6.1: włączenie rejestrowania bez konieczności ponownego komplikowania](#).

Fabryka połączenia domyślne pierwszy kodu

Sekcja konfiguracji pozwala określić domyślną fabrykę połączenia, która Code First powinna być używana do lokalizowania bazę danych do użycia dla kontekstu. Domyślna fabryka połączenia jest używane tylko w sytuacji, gdy parametry połączenia, nie został dodany do pliku konfiguracji dla kontekstu.

Po zainstalowaniu pakietu NuGet programu EF domyślną fabrykę połączenia został zarejestrowany, wskazujący SQL Express lub LocalDB, w zależności od tego, który z nich został zainstalowany.

Aby ustawić fabryka połączenia, określ nazwę typu kwalifikowanego zestawu w **defaultConnectionFactory** elementu.

NOTE

Kwalifikowana nazwa zestawu jest kwalifikowana nazwa przestrzeni nazw, a następnie przecinek, następnie zestawu, który typ, który znajduje się w. Opcjonalnie możesz również określić zestaw wersji, kulturę i token klucza publicznego.

Poniżej przedstawiono przykładową konfigurację własnych domyślnej fabryki połączenia:

```
<entityFramework>
  <defaultConnectionFactory type="MyNamespace.MyCustomFactory, MyAssembly"/>
</entityFramework>
```

Powyższy przykład wymaga niestandardowych fabryki, aby mieć konstruktora bez parametrów. Jeśli to konieczne, można określić parametry konstruktora przy użyciu **parametry** elementu.

Na przykład SqlCeConnectionFactory, który znajduje się w programie Entity Framework, wymaga podania nazwę niezmienną dostawcy do konstruktora. Nazwa niezmienna dostawcy identyfikuje wersję programu SQL Compact chcesz użyć. Następująca konfiguracja spowoduje, że kontekst do użycia w wersji SQL Compact 4.0 domyślnie.

```
<entityFramework>
  <defaultConnectionFactory type="System.Data.Entity.Infrastructure.SqlCeConnectionFactory, EntityFramework">
    <parameters>
      <parameter value="System.Data.SqlClient.4.0" />
    </parameters>
  </defaultConnectionFactory>
</entityFramework>
```

Jeśli nie ustawisz domyślnej fabryki połączenia Code First używa SqlConnectionFactory, wskazując `.\\SQLEXPRESS`. SqlConnectionFactory również ma konstruktora, który zezwala na zastąpienie części ciągu połączenia. Jeśli chcesz użyć wystąpienia programu SQL Server w innych niż `.\\SQLEXPRESS` można skonfigurować serwera, można użyć tego konstruktora.

Następująca konfiguracja spowoduje, że Code First użyć **MyDatabaseServer** dla kontekstów, które nie mają ciągu jawne połączenie zestawu.

```
<entityFramework>
  <defaultConnectionFactory type="System.Data.Entity.Infrastructure.SqlConnectionFactory, EntityFramework">
    <parameters>
      <parameter value="Data Source=MyDatabaseServer; Integrated Security=True;
MultipleActiveResultSets=True" />
    </parameters>
  </defaultConnectionFactory>
</entityFramework>
```

Domyślnie zakłada się, że argumenty konstruktora są typu ciąg. Aby zmienić to ustawienie, można użyć tego typu atrybutu.

```
<parameter value="2" type="System.Int32" />
```

Inicjatory bazy danych

Inicjatory bazy danych są skonfigurowane na podstawie kontekstów. Można je skonfigurować przy użyciu pliku konfiguracji **kontekstu** elementu. Ten element używa nazwy kwalifikowanej zestawu do zidentyfikowania kontekstu jest skonfigurowany.

Domyślnie program Code First kontekstów są skonfigurowane do używania inicjatora `CreateDatabaseIfNotExists`. Brak **disableDatabaseInitialization** atrybutu na **kontekstu** element, który może służyć do wyłączenia inicjowania bazy danych.

Na przykład następująca konfiguracja wyłącza inicjowanie bazy danych dla kontekstu `Blogging.BlogContext` zdefiniowane w `MyAssembly.dll`.

```
<contexts>
  <context type=" Blogging.BlogContext, MyAssembly" disableDatabaseInitialization="true" />
</contexts>
```

Możesz użyć **databaseInitializer** elementu, aby ustawić niestandardowe inicjatora.

```
<contexts>
  <context type=" Blogging.BlogContext, MyAssembly">
    <databaseInitializer type="Blogging.MyCustomBlogInitializer, MyAssembly" />
  </context>
</contexts>
```

Parametry Konstruktora użyć tej samej składni jako domyślnego połączenia fabryk.

```
<contexts>
  <context type=" Blogging.BlogContext, MyAssembly">
    <databaseInitializer type="Blogging.MyCustomBlogInitializer, MyAssembly">
      <parameters>
        <parameter value="MyConstructorParameter" />
      </parameters>
    </databaseInitializer>
  </context>
</contexts>
```

Można skonfigurować jeden inicjator ogólną bazę danych, które są objęte Entity Framework. **Typu** atrybutu używa formatu .NET Framework dla typów ogólnych.

Na przykład, jeśli używasz migracje Code First można skonfigurować bazy danych powinny być migrowane automatycznie przy użyciu `MigrateDatabaseToLatestVersion<TContext, TMigrationsConfiguration>` inicjatora.

```
<contexts>
  <context type="Blogging.BlogContext, MyAssembly">
    <databaseInitializer type="System.Data.Entity.MigrateDatabaseToLatestVersion`2[[Blogging.BlogContext,
MyAssembly], [Blogging.Migrations.Configuration, MyAssembly]], EntityFramework" />
  </context>
</contexts>
```

Parametry połączenia i modeli

13.09.2018 • 9 minutes to read • [Edit Online](#)

W tym temacie opisano, jak Entity Framework umożliwia odnalezienie połączenie bazy danych i jak można ją zmienić. Modele utworzone przy użyciu Code First i projektancie platformy EF zostały omówione w tym temacie.

Zazwyczaj aplikacja programu Entity Framework używa klasy pochodzącej od typu DbContext. Ta klasa pochodna wywoła jednym z konstruktorów w klasie bazowej DbContext do sterowania:

- Jak kontekst połączy się z bazą danych — oznacza to, jak ciąg połączenia jest znaleziono użyć
- Kontekst używa obliczania modelu za pomocą funkcji Code First czy załadować model utworzony za pomocą projektanta EF
- Dodatkowe opcje zaawansowane

Następujące fragmenty pokazano niektóre sposoby konstruktorów typu DbContext może być używana.

Za pomocą Code First połączenia zgodnie z Konwencją

W przypadku dowolnej innej konfiguracji nie zostało wykonane w aplikacji, wywołując konstruktora bez parametrów na DbContext spowoduje, że DbContext do pracy w trybie Code First przy użyciu połączenia z bazą danych utworzonych przez Konwencję. Na przykład:

```
namespace Demo.EF
{
    public class BloggingContext : DbContext
    {
        public BloggingContext()
        // C# will call base class parameterless constructor by default
        {
        }
    }
}
```

W tym przykładzie DbContext używa w przestrzeni nazw kwalifikowana nazwa Twojej class—Demo.EF.BloggingContext—as pochodnej kontekstu nazwy bazy danych i tworzy ciąg połączenia dla tej bazy danych przy użyciu programu SQL Express lub LocalDB. Jeśli obie są zainstalowane, będzie używany program SQL Express.

Visual Studio 2010 zawiera programu SQL Express, domyślnie i programu Visual Studio 2012 i nowszych obejmują LocalDB. Podczas instalacji pakietu EntityFramework NuGet sprawdza, czy serwer bazy danych, która jest dostępna. Pakiet NuGet zostanie następnie zaktualizuj plik konfiguracji ustawienia domyślnego serwera bazy danych, który używa Code First, podczas tworzenia połączenia zgodnie z Konwencją. Jeśli program SQL Express jest uruchomiona, będzie używany. Jeśli program SQL Express nie jest dostępna następnie LocalDB zostanie zarejestrowana jako domyślnej zamiast tego. Nie zmian do pliku konfiguracji, jeśli zawiera on już ustawienie domyślnej fabryki połączenia.

Za pomocą Code First połączenia, Konwencji i określona nazwę bazy danych

Jeśli nie wykonano dowolnej innej konfiguracji w aplikacji, następnie wywoływanie konstruktora ciągu na DbContext, nazwą bazy danych, którego chcesz użyć spowoduje, że DbContext do pracy w trybie Code First przy użyciu połączenia z bazą danych utworzony zgodnie z Konwencją do bazy danych tę nazwę. Na przykład:

```
public class BloggingContext : DbContext
{
    public BloggingContext()
        : base("BloggingDatabase")
    {
    }
}
```

W tym przykładzie DbContext używa "BloggingDatabase" jako nazwy bazy danych i tworzy ciąg połączenia dla tej bazy danych przy użyciu programu SQL Express (zainstalowany za pomocą programu Visual Studio 2010) lub LocalDB (zainstalowany za pomocą programu Visual Studio 2012). Jeśli obie są zainstalowane, będzie używany program SQL Express.

Za pomocą Code First parametrów połączenia w pliku app.config/web.config

Można umieścić ciąg połączenia w pliku app.config lub web.config. Na przykład:

```
<configuration>
  <connectionStrings>
    <add name="BloggingCompactDatabase"
        providerName="System.Data.SqlServerCe.4.0"
        connectionString="Data Source=Blogging.sdf"/>
  </connectionStrings>
</configuration>
```

Jest to prosty sposób sprawdzić DbContext do korzystania z serwera bazy danych innych niż SQL Express lub LocalDB — w powyższym przykładzie określa bazę danych programu SQL Server Compact Edition.

Jeśli nazwa ciągu połączenia jest zgodna z nazwą kontekstu (z lub bez kwalifikacji przestrzeni nazw) następnie go będą mogli odnaleźć DbContext stosowania konstruktora bez parametrów. Nazwa parametrów połączenia jest inna niż Nazwa kontekstu można określić typu DbContext, aby używać tego połączenia w trybie Code First, przekazując nazwę parametrów połączenia do konstruktora typu DbContext. Na przykład:

```
public class BloggingContext : DbContext
{
    public BloggingContext()
        : base("BloggingCompactDatabase")
    {
    }
}
```

Alternatywnie można użyć formy "nazwa = <nazwa parametrów połączenia>" dla ciągu przekazany do konstruktora typu DbContext. Na przykład:

```
public class BloggingContext : DbContext
{
    public BloggingContext()
        : base("name=BloggingCompactDatabase")
    {
    }
}
```

Ten formularz, sprawia, że jawne, oczekiwany ciąg połączenia, można znaleźć w pliku konfiguracji. Jeśli nie można odnaleźć parametrów połączenia o podanej nazwie, zostanie zgłoszony wyjątek.

Pierwszy Model/bazy danych przy użyciu parametrów połączenia w pliku app.config/web.config

Modele utworzone za pomocą projektanta EF różnią się od Code First w tym modelu już istnieje i nie jest generowany z kodu, gdy aplikacja zostanie uruchomiona. Ten model jest zwykle istnieje jako plik EDMX w projekcie.

Projektant doda EF parametry połączenia do pliku app.config lub web.config. Parametry połączenia są specjalne, ponieważ zawiera informacje o sposobach znajdowania informacji w pliku EDMX. Na przykład:

```
<configuration>
  <connectionStrings>
    <add name="Northwind_Entities"
      connectionString="metadata=res://*/Northwind.csdl|
                        res://*/Northwind.ssdl|
                        res://*/Northwind.msl;
      provider=System.Data.SqlClient;
      provider connection string=
        "Data Source=.\sqlexpress;
          Initial Catalog=Northwind;
          Integrated Security=True;
          MultipleActiveResultSets=True";
      providerName="System.Data.EntityClient"/>
  </connectionStrings>
</configuration>
```

W Projektancie platformy EF także wygeneruje kod, który zawiera informacje dla kontekstu DbContext dla tego połączenia, przekazując nazwę parametrów połączenia do konstruktora typu DbContext. Na przykład:

```
public class NorthwindContext : DbContext
{
    public NorthwindContext()
        : base("name=Northwind_Entities")
    {
    }
}
```

Kontekst DbContext wie, można załadować istniejącego modelu (a nie za pomocą Code First go obliczyć z kodu), ponieważ parametry połączenia są parametry połączenia platformy EF zawierającego szczegółowe informacje o modelu do użycia.

Inne opcje konstruktora typu DbContext

Klasy DbContext zawiera inne konstruktory i wzorce użycia, które umożliwiają niektórych bardziej zaawansowanych scenariuszy. Niektóre z nich to:

- Klasa DbModelBuilder umożliwia tworzenie model Code First bez tworzenia wystąpienia wystąpienia typu DbContext. Z tego powodu jest obiektem DbModel. Możesz następnie przekazać ten obiekt DbModel do jednego z konstruktorów typu DbContext, gdy jesteś gotowy do utworzenia wystąpienia typu DbContext.
- Pełny ciąg połączenia można przekazać do typu DbContext, zamiast tylko nazwy ciągu bazy danych lub połączenie. Domyslnie ten ciąg połączenia jest używana z dostawcą System.Data.SqlClient; Można to zmienić, ustawiając z inną implementacją IConnectionFactory na kontekście. Database.DefaultConnectionFactory.
- Można użyć istniejącego obiektu DbConnection, przekazując go do konstruktora typu DbContext. Jeśli obiekt połączenia jest wystąpieniem EntityConnection, określony w połączeniu model nie będą używane zamiast obliczania modelu przy użyciu najpierw kod. Jeśli obiekt jest wystąpieniem innego typu — na przykład element SqlConnection — a następnie kontekst używa go w trybie Code First.

- Istniejący obiekt ObjectContext można przekazać do konstruktora typu DbContext do utworzenia typu DbContext zawijania istniejącego kontekstu. Może to służyć do istniejących aplikacji, które używają obiektu ObjectContext, ale którym chcesz korzystać z zalet DbContext w niektórych części aplikacji.

Rozpoznawanie zależności

13.09.2018 • 14 minutes to read • [Edit Online](#)

NOTE

EF6 począwszy tylko — funkcje, interfejsów API itp. z opisem na tej stronie zostały wprowadzone w programie Entity Framework 6. Jeśli używasz starszej wersji, niektóre lub wszystkie informacje, nie ma zastosowania.

Począwszy od platformy EF6 Entity Framework zawiera mechanizm ogólnego przeznaczenia do uzyskania implementacji usług, które są wymagane. Oznacza to kiedy EF używa wystąpienia niektóre interfejsy lub klas bazowych zostanie wyświetlony monit dla konkretnej implementacji interfejsu lub klasy bazowej do użycia. Jest to osiągane przy użyciu interfejsu IDbDependencyResolver:

```
public interface IDbDependencyResolver
{
    object GetService(Type type, object key);
}
```

Metoda GetService zwykle jest wywoływana przez EF i jest obsługiwane przez implementację IDbDependencyResolver EF lub aplikacji. Gdy zostanie wywołana, argument typu jest typem klasy interfejsu lub base żądanej usługi i klucz obiektu jest wartość null lub obiekt dostarczający informacje kontekstowe o żądanej usługi.

Jeżeli nie określono inaczej, dowolny obiekt zwrócony musi być metodą o bezpiecznych wątkach, ponieważ może służyć jako pojedynczą. W wielu przypadkach, w których obiekt zwrócony, w którym to przypadku to fabryka sama fabryka musi być metodą o bezpiecznych wątkach, ale obiekt zwrócony z fabryki nie musi być metodą o bezpiecznych wątkach, ponieważ zażądano nowe wystąpienie z fabryki dla każdego zastosowania.

Ten artykuł zawiera szczegółowe informacje o sposobie implementacji IDbDependencyResolver, ale zamiast tego działa jako odwołanie dla typów usługi (oznacza to, że interfejs i podstawowej klasy typy), dla których EF wywołuje GetService i semantyka obiekt klucza dla każdego z nich wywołuje.

System.Data.Entity.IDatabaseInitializer < TContext >

Wprowadzona w wersji: EF6.0.0

Obiekt zwrócony: Inicjator bazy danych dla typu podanym kontekście

Klucz: nie jest używany; będzie miał wartość null

FUNC < System.Data.Entity.Migrations.Sql.SqlMigrationGenerator >

Wprowadzona w wersji: EF6.0.0

Obiekt zwrócony: fabrykę do tworzenia generator SQL, który może służyć do migracji i inne akcje, które powodują bazy danych ma zostać utworzony, takie jak tworzenie bazy danych z inicjatorami bazy danych.

Klucz: ciąg zawierający nazwę niezmienną dostawcy ADO.NET, określanie typu bazy danych, dla którego zostanie wygenerowany SQL. Na przykład generator programu SQL Server SQL jest zwracana dla klucza "System.Data.SqlClient".

NOTE

Zobacz szczegółowe informacje na temat związane z dostawcą usług w EF6 [modelu dostawca EF6](#) sekcji.

System.Data.Entity.Core.Common.DbProviderServices

Wprowadzona w wersji: EF6.0.0

Obiekt zwrócony: Dostawca EF do użycia dla danego dostawcy o niezmiennej nazwie

Klucz: ciąg zawierający nazwę niezmienną dostawcy ADO.NET, określanie typu bazy danych, dla której dostawca jest wymagana. Na przykład dostawca programu SQL Server jest zwracana dla klucza "System.Data.SqlClient".

NOTE

Zobacz szczegółowe informacje na temat związane z dostawcą usług w EF6 [modelu dostawca EF6](#) sekcji.

System.Data.Entity.Infrastructure.IDbConnectionFactory

Wprowadzona w wersji: EF6.0.0

Obiekt zwrócony: fabryka połączenia, który będzie używany podczas EF, tworzenia połączenia z bazą danych według Konwencji. Oznacza to gdy nie połączenia lub parametry połączenia znajduje się do programu EF, a nie ciągu połączenia można znaleźć w pliku app.config lub web.config, następnie ta usługa służy do tworzenia połączenia zgodnie z Konwencją. Zmiana fabryka połączenia można zezwolić EF użyć innego typu bazy danych (na przykład SQL Server Compact Edition) domyślnie.

Klucz: nie jest używany; będzie miał wartość null

NOTE

Zobacz szczegółowe informacje na temat związane z dostawcą usług w EF6 [modelu dostawca EF6](#) sekcji.

System.Data.Entity.Infrastructure.IManifestTokenService

Wprowadzona w wersji: EF6.0.0

Obiekt zwrócony: usługi, który można wygenerować token manifestu dostawcy z połączenia. Ta usługa jest zwykle używana na dwa sposoby. Po pierwsze może służyć w celu uniknięcia Code First połączenie z bazą danych, podczas tworzenia modelu. Po drugie może służyć do wymuszenia Code First na budowanie modelu na potrzeby wersji konkretnej bazy danych — na przykład, aby wymusić model dla programu SQL Server 2005, nawet jeśli czasami jest używany program SQL Server 2008.

Okres istnienia obiektu: pojedyncze — ten sam obiekt może być używana wiele razy, a jednocześnie przez inne wątki

Klucz: nie jest używany; będzie miał wartość null

System.Data.Entity.Infrastructure.IDbProviderFactoryService

Wprowadzona w wersji: EF6.0.0

Obiekt zwrócony: usługi, który można uzyskać fabryki dostawcy z danego połączenia. W .NET 4.5 dostawca jest publicznie dostępny z poziomu połączenia. W .NET 4 Domyślna implementacja tej usługi używa niektóre

heurystyki w celu odnalezienia pasującego dostawcy. Jeśli te nie powiodą się następnie nową metodę implementacji tej usługi można zarejestrować zapewniające odpowiednie rozwiązanie.

Klucz: nie jest używany; będzie miał wartość null

FUNC < DbContext, System.Data.Entity.Infrastructure.IDbModelCacheKey>

Wprowadzona w wersji: EF6.0.0

Obiekt zwrócony: fabryki, który zostanie wygenerowany klucz pamięci podręcznej modelu dla danego kontekstu. Domyślnie program EF buforuje jednego modelu dla typu DbContext dla dostawcy. Inną implementację tej usługi można dodać inne informacje, takie jak nazwa schematu do klucza pamięci podręcznej.

Klucz: nie jest używany; będzie miał wartość null

System.Data.Entity.Spatial.DbSpatialServices

Wprowadzona w wersji: EF6.0.0

Obiekt zwrócony: EF przestrzenne dostawcy, który dodaje obsługę podstawowych EF dostawcy typów przestrzennych geometry i położenia geograficznego.

Klucz: DbSpatialServices zostanie poproszony o na dwa sposoby. Po pierwsze, właściwe dla dostawcy usług przestrzennych, są żądane przy użyciu obiektu DbProviderInfo (zawierający niezmienną token nazwy, a manifestu) jako klucza. Po drugie DbSpatialServices można monit o wpisanie bez klucza. Służy to rozwiązać "globalne przestrzenne dostawcę" który jest używany podczas tworzenia autonomicznego DbGeography lub DbGeometry typów.

NOTE

Zobacz szczegółowe informacje na temat związane z dostawcą usług w EF6 [modelu dostawca EF6](#) sekcji.

FUNC < System.Data.Entity.Infrastructure.IDbExecutionStrategy>

Wprowadzona w wersji: EF6.0.0

Obiekt zwrócony: fabryki, aby utworzyć usługę, która umożliwia dostawcy zaimplementować ponownych prób lub inne zachowanie, gdy zapytań i poleceń, które są wykonywane względem bazy danych. Jeśli nie dostarczono żadnej implementacji, EF będzie po prostu wykonaj polecenia i propagowanie wyjątki zgłoszone. Dla programu SQL Server ta usługa służy do zapewnienia zasady ponawiania, co jest szczególnie przydatne, jeśli działających w odniesieniu do serwerów bazy danych opartej na chmurze, takich jak SQL Azure.

Klucz: obiekt ExecutionStrategyKey, który zawiera nazwę niezmienną dostawcy i opcjonalnie nazwy serwera, dla której będzie służyć strategii wykonywania.

NOTE

Zobacz szczegółowe informacje na temat związane z dostawcą usług w EF6 [modelu dostawca EF6](#) sekcji.

FUNC < DbConnection, string, System.Data.Entity.Migrations.History.HistoryContext>

Wprowadzona w wersji: EF6.0.0

Obiekt zwrócony: fabrykę, która umożliwia dostawcy skonfigurować mapowanie HistoryContext do `_MigrationHistory` tabeli używanej przez migracje EF. HistoryContext jest pierwszy typu DbContext kodu i można skonfigurować przy użyciu normalnych wygodnego interfejsu API można zmienić elementów, takich jak nazwy tabeli i specyfikacji mapowania kolumn.

Klucz: nie jest używany; będzie miał wartość null

NOTE

Zobacz szczegółowe informacje na temat związane z dostawcą usług w EF6 [modelu dostawca EF6](#) sekcji.

System.Data.Common.DbProviderFactory

Wprowadzona w wersji: EF6.0.0

Obiekt zwrócony: Dostawca ADO.NET do użycia dla danego dostawcy o niezmiennej nazwie.

Klucz: ciąg zawierający nazwę niezmienną dostawcy ADO.NET

NOTE

Ta usługa nie jest zazwyczaj zmieniany bezpośrednio od implementacji domyślnej jest używana normalnej rejestracji dostawcy ADO.NET. Zobacz szczegółowe informacje na temat związane z dostawcą usług w EF6 [modelu dostawca EF6](#) sekcji.

System.Data.Entity.Infrastructure.IProviderInvariantName

Wprowadzona w wersji: EF6.0.0

Obiekt zwrócony: usługa, która jest używana do określenia dla danego typu DbProviderFactory nazwę niezmienną dostawcy. Domyślna implementacja tej usługi używa rejestracji dostawcy ADO.NET. Oznacza to, że jeśli dostawcy ADO.NET nie jest zarejestrowany w normalny sposób, ponieważ DbProviderFactory jest rozwiązywany za EF, następnie również będzie wymagany do rozwiązania tej usługi.

Klucz: DbProviderFactory wystąpienia, dla których niezmienna nazwa jest wymagana.

NOTE

Zobacz szczegółowe informacje na temat związane z dostawcą usług w EF6 [modelu dostawca EF6](#) sekcji.

System.Data.Entity.Core.Mapping.ViewGeneration.IViewAssemblyCache

Wprowadzona w wersji: EF6.0.0

Obiekt zwrócony: pamięć podręczna zestawów, które zawierają wstępnie wygenerowanych widoków. Zazwyczaj służy zamiennika umożliwiające EF wiedzieć, zestawy, które zawierają wstępnie wygenerowanych widoków bez żadnych odnajdywania.

Klucz: nie jest używany; będzie miał wartość null

System.Data.Entity.Infrastructure.Pluralization.IPluralizationService

Wprowadzona w wersji: EF6.0.0

Obiekt zwrócony: Usługa używana przez EF w liczbie mnogiej do nazw końcówek. Domyślnie usługa angielskiej

pluralizacja jest używana.

Klucz: nie jest używany; będzie miał wartość null

System.Data.Entity.Infrastructure.Interception.IDbInterceptor

Wprowadzona w wersji: EF6.0.0

Obiekty zwrócone: wszelkie interceptory, które powinny być rejestrowane podczas uruchamiania aplikacji. Należy zauważyć, że te obiekty są żądane za pomocą wywołania funkcji GetServices i interceptory wszystkie zwrócone przez dowolnego mechanizmu rozpoznawania zależności zostanie zarejestrowana.

Klucz: nie jest używany; będzie miał wartość null.

FUNC < System.Data.Entity.DbContext, akcja < ciąg>, System.Data.Entity.Infrastructure.Interception.DatabaseLogFormatter>

Wprowadzona w wersji: EF6.0.0

Obiekt zwrocony: fabrykę, która będzie służyć do tworzenia elementu formatującego dziennika bazy danych, który będzie używany podczas kontekstu. Database.Log właściwość jest ustawiona w danym kontekście.

Klucz: nie jest używany; będzie miał wartość null.

FUNC < System.Data.Entity.DbContext>

Wprowadzona w wersji: EF6.1.0

Obiekt zwrocony: fabrykę, która będzie służyć do tworzenia wystąpień kontekstu dla migracji, gdy kontekst nie jest dostępny konstruktor bez parametrów.

Klucz: typ obiektu dla typu pochodnego typu DbContext potrzeby fabrykę.

FUNC < System.Data.Entity.Core.Metadata.Edm.IMetadataAnnotationSerializer >

Wprowadzona w wersji: EF6.1.0

Obiekt zwrocony: fabrykę, która będzie służyć do utworzyć serializatorów serializacji silnie typizowane niestandardowe adnotacje taki sposób, że może być serializowany i desterilized do formatu XML do użycia w migracji Code First.

Klucz: Nazwa adnotacji, który jest serializowany lub deserializowany.

FUNC < System.Data.Entity.Infrastructure.TransactionHandler>

Wprowadzona w wersji: EF6.1.0

Obiekt zwrocony: fabrykę, która będzie służyć do tworzenie programów do obsługi transakcji, tak aby specjalnej obsługi można zastosować w sytuacjach, takich jak obsługa zatwierdzenia awarie.

Klucz: obiekt ExecutionStrategyKey, który zawiera nazwę niezmienną dostawcy i opcjonalnie nazwy serwera, dla której będzie używany program obsługi transakcji.

Zarządzanie połączeniami

13.09.2018 • 7 minutes to read • [Edit Online](#)

Na tej stronie opisano zachowanie programu Entity Framework w odniesieniu do przekazywania połączenia do kontekstu i funkcjonalność **Database.Connection.Open()** interfejsu API.

Przekazywanie połączeń do kontekstu

Zachowanie EF5 i wcześniejszymi wersjami

Istnieją dwa konstruktory, które akceptują połączenia:

```
public DbContext(DbConnection existingConnection, bool contextOwnsConnection)
public DbContext(DbConnection existingConnection, DbCompiledModel model, bool contextOwnsConnection)
```

Istnieje możliwość, aby móc ich używać, ale trzeba pracować na kilka ograniczeń:

1. W przypadku przekazania otwartego połączenia do jednego z tych następnie po raz pierwszy ramach podejmuję próbę użycia go, który jest generowany, InvalidOperationException stwierdzającego nie można ponownie otworzyć już otwartego połączenia.
2. Flaga contextOwnsConnection jest interpretowana oznaczającą, czy połączenie podstawowe magazynu powinny zostać usunięte, jeśli kontekst zostanie usunięty. Jednak niezależnie od tego ustawnienia połączenia magazynu jest zawsze zamknięte, jeśli kontekst zostanie usunięty. Dlatego jeśli masz więcej niż jednego typu DbContext w ramach tego samego niezależnie od kontekst zostanie usunięty, najpierw połączenie zostanie zamknięte (podobnie jeśli mają różne istniejącego połączenia ADO.NET z typu DbContext, DbContext zawsze zamknie połączenie po jego usunięciu) .

Istnieje możliwość obejść pierwszy ograniczenie powyżej, przekazując zamkniętego połączenia i tylko wykonywanie kodu, który otwiera go po utworzeniu wszystkich kontekstach:

```

using System.Collections.Generic;
using System.Data.Common;
using System.Data.Entity;
using System.Data.Entity.Infrastructure;
using System.Data.EntityClient;
using System.Linq;

namespace ConnectionManagementExamples
{
    class ConnectionManagementExampleEF5
    {
        public static void TwoDbContextsOneConnection()
        {
            using (var context1 = new BloggingContext())
            {
                var conn =
                    ((EntityConnection)
                        ((IObjectContextAdapter)context1).ObjectContext.Connection)
                        .StoreConnection;

                using (var context2 = new BloggingContext(conn, contextOwnsConnection: false))
                {
                    context2.Database.ExecuteSqlCommand(
                        @"UPDATE Blogs SET Rating = 5" +
                        " WHERE Name LIKE '%Entity Framework%'");

                    var query = context1.Posts.Where(p => p.Blog.Rating > 5);
                    foreach (var post in query)
                    {
                        post.Title += "[Cool Blog]";
                    }
                    context1.SaveChanges();
                }
            }
        }
    }
}

```

Drugi ograniczenie po prostu oznacza, że musisz punktowanych — usuwanie wszystkich obiektów typu DbContext, dopóki nie będziesz gotowy dla połączenia zostanie zamknięty.

Zachowanie w przyszłych wersjach i EF6

W przyszłych wersjach i EF6 kontekstu DbContext ma ten sam dwa konstruktory, ale nie wymaga już zamknięte połączenia przekazany do konstruktora po ich odebraniu. Dlatego teraz jest to możliwe:

```

using System.Collections.Generic;
using System.Data.Entity;
using System.Data.SqlClient;
using System.Linq;
using System.Transactions;

namespace ConnectionManagementExamples
{
    class ConnectionManagementExample
    {
        public static void PassingAnOpenConnection()
        {
            using (var conn = new SqlConnection("{connectionString}"))
            {
                conn.Open();

                var sqlCommand = new SqlCommand();
                sqlCommand.Connection = conn;
                sqlCommand.CommandText =
                    @"UPDATE Blogs SET Rating = 5" +
                    " WHERE Name LIKE '%Entity Framework%'";
                sqlCommand.ExecuteNonQuery();

                using (var context = new BloggingContext(conn, contextOwnsConnection: false))
                {
                    var query = context.Posts.Where(p => p.Blog.Rating > 5);
                    foreach (var post in query)
                    {
                        post.Title += "[Cool Blog]";
                    }
                    context.SaveChanges();
                }

                var sqlCommand2 = new SqlCommand();
                sqlCommand2.Connection = conn;
                sqlCommand2.CommandText =
                    @"UPDATE Blogs SET Rating = 7" +
                    " WHERE Name LIKE '%Entity Framework Rocks%'";
                sqlCommand2.ExecuteNonQuery();
            }
        }
    }
}

```

Flaga `contextOwnsConnection` teraz kontroluje również informację, czy połączenie jest zamknięte i usunięty po usunięciu kontekstu `DbContext`. Dlatego w powyższym przykładzie połączenie nie jest zamknięte w kontekście usunięty (wiersz 32), ponieważ miałoby to miejsce w poprzednich wersjach programu EF, ale raczej, jeśli jest to samo połączenie zostanie usunięty (wiersz 40).

Oczywiście jest nadal możliwe dla kontekstu `DbContext` przejąć kontrolę nad połączenia (tylko zestaw `contextOwnsConnection` na wartość `true`, lub użyj jednego z innych konstruktory). Jeśli więc chcesz:

NOTE

Istnieje kilka dodatkowych kwestii dotyczących podczas korzystania z transakcji za pomocą tego nowego modelu. Aby uzyskać szczegółowe informacje, zobacz [Praca z transakcją](#).

Database.Connection.Open()

Zachowanie EF5 i wcześniejszymi wersjami

EF5 i wcześniejszych wersji jest to błąd, `ObjectContext.Connection.State` nie został zaktualizowany w celu

odzwierciedlenia rzeczywisty stan bazowego połączenia magazynu. Na przykład, jeśli zostanie wykonane następujący kod należy mogą być zwracane stan **zamknięte** mimo, że w rzeczywistości bazowego przechowywania połączenia **Otwórz**.

```
((IObjectContextAdapter)context).ObjectContext.Connection.State
```

Oddzielnie, jeśli otworzysz połączenie z bazą danych, wywołując Database.Connection.Open() będą one otwarte do momentu przy następnym wykonania kwerendy lub wywołanie niczego, co wymaga połączenia z bazą danych (na przykład SaveChanges()), ale po czym bazowego przechowywania połączenie zostanie zamknięte. Kontekst będzie, a następnie ponownie otworzyć i ponownie zamknij połączenie, każdym razem, gdy inną operacją bazy danych jest wymagane:

```
using System;
using System.Data;
using System.Data.Entity;
using System.Data.Entity.Infrastructure;
using System.Data.EntityClient;

namespace ConnectionManagementExamples
{
    public class DatabaseOpenConnectionBehaviorEF5
    {
        public static void DatabaseOpenConnectionBehavior()
        {
            using (var context = new BloggingContext())
            {
                // At this point the underlying store connection is closed

                context.Database.Connection.Open();

                // Now the underlying store connection is open
                // (though ObjectContext.Connection.State will report closed)

                var blog = new Blog { /* Blog's properties */ };
                context.Blogs.Add(blog);

                // The underlying store connection is still open

                context.SaveChanges();

                // After SaveChanges() the underlying store connection is closed
                // Each SaveChanges() / query etc now opens and immediately closes
                // the underlying store connection

                blog = new Blog { /* Blog's properties */ };
                context.Blogs.Add(blog);
                context.SaveChanges();
            }
        }
    }
}
```

Zachowanie w przyszłych wersjach i EF6

EF6 i przyszłych wersji iż podejmujemy podejście, jeśli kod wywołujący wybierze do otwierania połączenia przez kontekst wywołania Database.Connection.Open(), a następnie go ma przyczyna to i platformę będzie założono, że chce kontrolować otwierające i zamykające połączenia i nie jest już połączenie zostało zamknięte automatycznie.

NOTE

Może to prowadzić do połączenia, które są otwarte przez długi czas, więc używać ostrożnie.

Zaktualizowaliśmy również kod tak, aby `ObjectContext.Connection.State` teraz śledzi informacje o stan połączenia podstawowej poprawnie.

```
using System;
using System.Data;
using System.Data.Entity;
using System.Data.Entity.Core.EntityClient;
using System.Data.Entity.Infrastructure;

namespace ConnectionManagementExamples
{
    internal class DatabaseOpenConnectionBehaviorEF6
    {
        public static void DatabaseOpenConnectionBehavior()
        {
            using (var context = new BloggingContext())
            {
                // At this point the underlying store connection is closed

                context.Database.Connection.Open();

                // Now the underlying store connection is open and the
                // ObjectContext.Connection.State correctly reports open too

                var blog = new Blog { /* Blog's properties */ };
                context.Blogs.Add(blog);
                context.SaveChanges();

                // The underlying store connection remains open for the next operation

                blog = new Blog { /* Blog's properties */ };
                context.Blogs.Add(blog);
                context.SaveChanges();

                // The underlying store connection is still open

            } // The context is disposed - so now the underlying store connection is closed
        }
    }
}
```

Logika połączenia odporności, a następnie spróbuj ponownie

13.09.2018 • 12 minutes to read • [Edit Online](#)

NOTE

EF6 począwszy tylko — funkcje, interfejsów API itp. z opisem na tej stronie zostały wprowadzone w programie Entity Framework 6. Jeśli używasz starszej wersji, niektóre lub wszystkie informacje, nie ma zastosowania.

Aplikacje nawiązywania połączenia z serwerem bazy danych zawsze były podatne na podziały połączenia z powodu niepowodzenia zaplecza i niestabilności sieci. Jednak w środowisku sieci LAN, na podstawie działa z serwerów dedykowanych bazy danych te błędy są wystarczająco rzadkie, że dodatkowa logika do obsługi tych błędów nie jest często wymagane. Za pomocą w chmurze oparte mniej niezawodnej sieci, z którymi jest teraz bardziej powszechnie podziału połączenia wystąpienia serwerów baz danych, takich jak Windows Azure SQL Database oraz połączeń. Może to być spowodowane obrony technik, które w chmurze Użyj bazy danych, aby zapewnić sprawiedliwe usługi, takie jak ograniczanie przepustowości połączenia lub do niestabilności w sieci, powodują sporadyczne przekroczeń limitu czasu i innych błędów przejściowych.

Elastyczność połączenia odnosi się do możliwości EF do automatycznego ponawiania próby wykonania dowolne polecenia, które się nie powieść z powodu przerwy te połączenia.

Strategii wykonywania

Ponów próbę połączenia są wykonywane przez implementację interfejsu `IDbExecutionStrategy`. Implementacje `IDbExecutionStrategy` będzie odpowiedzialny za akceptować operacji, a jeśli wystąpi wyjątek, określającą, czy ponowienie próby jest odpowiednia i ponawianie próby, jeśli jest. Istnieją cztery strategii wykonywania, które są dostarczane z programem EF:

1. **DefaultExecutionStrategy:** Ta strategia wykonywania nie ponawia próby żadnych operacji, jest ustawieniem domyślnym dla baz danych innych niż sql server.
2. **DefaultSqlExecutionStrategy:** jest to strategii wykonywania wewnętrzny używany domyślnie. Ta strategia nie ponawia próby w ogóle, jednak opakować wszystkie wyjątki, które mogą być przejściowe poinformować użytkowników, którzy chcą włączyć elastyczność połączenia.
3. **DbExecutionStrategy:** Ta klasa jest odpowiednie, jako klasę bazową dla innych strategii wykonywania, w tym własne niestandardowe. Implementuje zasady ponawiania wykładniczego, gdzie początkowej ponawiania się dzieje z zero opóźnienia i opóźnienia rośnie wykładniczo, dopóki nie zostanie osiągnięty jako maksymalna liczba ponowień. Ta klasa posiada metodę `ShouldRetryOn` abstrakcyjną, która może być implementowany w strategii wykonywania pochodnych do kontrolowania, wyjątki, które należy wykonać ponownie.
4. **SqlAzureExecutionStrategy:** Ta strategia wykonywania dziedziczy `DbExecutionStrategy` i spróbuje ponowić operację na wyjątki, które są znane jako prawdopodobnie przejściowy podczas pracy z usługą Azure SQL Database.

NOTE

Strategii wykonywania 2 i 4 są uwzględnione w dostawcy programu Sql Server, który jest dostarczany z programem EF, który znajduje się w zestawie `EntityFramework.SqlServer` i zostały zaprojektowane do pracy z programem SQL Server.

Włączanie strategii wykonywania

Najprostszym sposobem Poinformuj EF, aby użyć strategii wykonywania jest przy użyciu metody SetExecutionStrategy **DbConfiguration** klasy:

```
public class MyConfiguration : DbConfiguration
{
    public MyConfiguration()
    {
        SetExecutionStrategy("System.Data.SqlClient", () => new SqlAzureExecutionStrategy());
    }
}
```

Ten kod informuje EF, aby użyć **SqlAzureExecutionStrategy** podczas nawiązywania połączenia z programem SQL Server.

Konfigurowanie strategii wykonywania

Konstruktor obiektu **SqlAzureExecutionStrategy** może akceptować dwa parametry: wartość **MaxRetryCount** i **MaxDelay**. Liczba **MaxRetry** jest maksymalna liczba strategii ponownych prób. **MaxDelay** jest element **TimeSpan** reprezentujący Maksymalne opóźnienie między kolejnymi próbami używających strategii wykonywania.

Aby ustawić maksymalną liczbę ponownych prób 1 i maksymalne opóźnienie do 30 sekund będzie się execute następujące czynności:

```
public class MyConfiguration : DbConfiguration
{
    public MyConfiguration()
    {
        SetExecutionStrategy(
            "System.Data.SqlClient",
            () => new SqlAzureExecutionStrategy(1, TimeSpan.FromSeconds(30)));
    }
}
```

SqlAzureExecutionStrategy ponowi próbę natychmiast po raz pierwszy błąd przejściowy, który występuje, ale będzie już opóźnienie między kolejnymi próbami, dopóki albo maksymalny limit ponownych prób zostanie przekroczony lub łącznego czasu trafienia Maksymalne opóźnienie.

Strategii wykonywania ponowi tylko ograniczoną liczbę wyjątków, które są zwykle transient, nadal konieczne będzie obsługiwać inne błędy, a także przechwytywanie wyjątku **RetryLimitExceeded** w przypadku których błąd nie jest przejściowy lub trwa zbyt długo rozwiązać samego siebie.

Korzystając z Trwa ponawianie próby strategii wykonywania są niektóre znane ograniczenia:

Zapytania przesyłania strumieniowego nie są obsługiwane.

Domyślnie programy EF6 i nowszym będzie buforować wyniki zapytania, a nie ich streaming. Jeśli chcesz mieć wyniki przesyłane strumieniowo można metoda **AsStreaming** służy do zmiany LINQ zapytania jednostki w celu przesyłania strumieniowego.

```
using (var db = new BloggingContext())
{
    var query = (from b in db.Blogs
                 orderby b.Url
                 select b).AsStreaming();
}
```

Przesyłania strumieniowego nie jest obsługiwana, gdy Trwa ponawianie próby strategii wykonywania jest zarejestrowany. To ograniczenie istnieje, ponieważ połączenie można porzucić część sposob wyniki są zwracane. W takiej sytuacji EF musi zostać ponownie uruchomiony całe zapytanie, ale nie ma niezawodne możliwości informacji o tym, co powoduje już zostały zwrócone (dane mogły ulec zmianie od momentu początkowego zapytania została wysłana, wyniki mogą wrócić w innej kolejności, wyniki nie może mieć unikatowy identyfikator itp.).

Transakcji zainicjowanej przez użytkownika nie są obsługiwane.

Po skonfigurowaniu strategii wykonywania, który skutkuje ponownych prób, istnieją pewne ograniczenia dotyczące użycia transakcji.

Domyślnie program EF będzie wykonywać żadnych aktualizacji bazy danych w obrębie transakcji. Nie trzeba nic robić, aby włączyć tę opcję, EF zawsze wykonuje to automatycznie.

Na przykład w poniższym kodzie SaveChanges jest wykonywana automatycznie w ramach transakcji. Gdyby SaveChanges się niepowodzeniem po Wstawianie jedną nową lokację, a następnie będzie można z powrotem obniżyć transakcji i nie zmiany zostały wprowadzone do bazy danych. Kontekst jest również pozostanie w stanie umożliwiającym Funkcja SaveChanges zapisuje do ponownie wywołany, aby ponowić próbę zastosowania zmian.

```
using (var db = new BloggingContext())
{
    db.Blogs.Add(new Site { Url = "http://msdn.com/data/ef" });
    db.Blogs.Add(new Site { Url = "http://blogs.msdn.com/adonet" });
    db.SaveChanges();
}
```

Bez korzystania z Trwa ponawianie próby strategii wykonywania może zawijać się wiele operacji w ramach jednej transakcji. Na przykład poniższy kod opakowuje dwóch wywołań funkcji SaveChanges w ramach jednej transakcji. Jeśli którykolwiek którykolwiek z tych operacji nie powiedzie się następnie zmiany zostaną zastosowane.

```
using (var db = new BloggingContext())
{
    using (var trn = db.Database.BeginTransaction())
    {
        db.Blogs.Add(new Site { Url = "http://msdn.com/data/ef" });
        db.Blogs.Add(new Site { Url = "http://blogs.msdn.com/adonet" });
        db.SaveChanges();

        db.Blogs.Add(new Site { Url = "http://twitter.com/efmagicunicorns" });
        db.SaveChanges();

        trn.Commit();
    }
}
```

To nie jest obsługiwana, gdy za pomocą Trwa ponawianie próby strategii wykonywania, ponieważ EF nie jest świadomy żadnych poprzedniej operacji i sposób ponowić próbę ich wykonania. Na przykład jeśli drugi SaveChanges nie powiodła się następnie EF już ma wymagane informacje, aby ponowić próbę wykonania pierwszego wywołania funkcji SaveChanges.

Obejście: Wstrzymywanie strategii wykonywania

Jednym możliwym obejściem jest wstrzymania Trwa ponawianie próby strategia wykonywania dla fragmentu kodu, który musi używać użytkownik zainicjował transakcji. Najprostszym sposobem, w tym celu jest dodać flagę SuspendExecutionStrategy w kodzie na podstawie klasy konfiguracji, a także zmienić lambda strategii wykonywania do zwrócenia strategii wykonywania (inne niż retying) domyślny, jeśli flaga jest ustawiona.

```
using System.Data.Entity;
using System.Data.Entity.Infrastructure;
using System.Data.Entity.SqlServer;
using System.Runtime.Remoting.Messaging;

namespace Demo
{
    public class MyConfiguration : DbConfiguration
    {
        public MyConfiguration()
        {
            this.SetExecutionStrategy("System.Data.SqlClient", () => SuspendExecutionStrategy
                ? ( IDbExecutionStrategy )new DefaultExecutionStrategy()
                : new SqlAzureExecutionStrategy());
        }

        public static bool SuspendExecutionStrategy
        {
            get
            {
                return ( bool? )CallContext.LogicalGetData( "SuspendExecutionStrategy" ) == false;
            }
            set
            {
                CallContext.LogicalSetData( "SuspendExecutionStrategy", value );
            }
        }
    }
}
```

Należy pamiętać, że używasz CallContext do przechowywania wartości flagi. Zapewnia funkcje podobne do pamięci lokalnej wątku, ale jest bezpieczne, za pomocą kodu asynchronicznego — w tym zapytania asynchroniczne i zapisywać przy użyciu platformy Entity Framework.

Firma Microsoft jest teraz wstrzymywanie strategii wykonywania dla sekcji kodu, który używa transakcji zainicjowanej przez użytkownika.

```
using ( var db = new BloggingContext() )
{
    MyConfiguration.SuspendExecutionStrategy = true;

    using ( var trn = db.Database.BeginTransaction() )
    {
        db.Blogs.Add( new Blog { Url = "http://msdn.com/data/ef" } );
        db.Blogs.Add( new Blog { Url = "http://blogs.msdn.com/adonet" } );
        db.SaveChanges();

        db.Blogs.Add( new Blog { Url = "http://twitter.com/efmagicunicorns" } );
        db.SaveChanges();

        trn.Commit();
    }

    MyConfiguration.SuspendExecutionStrategy = false;
}
```

Obejście: Ręcznego wywoływania strategii wykonywania

Inną opcją jest ręczne Użyj strategii wykonywania i nadaj cały zestaw logiki, aby uruchomić, dzięki czemu można ponów wszystko, jeśli jedna z operacji zakończy się niepowodzeniem. Nadal trzeba wstrzymywanie strategii wykonywania - korzystające z techniki powyżej — tak aby kontekstów używana wewnątrz bloku kodu powtarzający operację nie należy podejmować prób.

Należy pamiętać, że w bloku kodu, ponowienie próby powinien być konstruowany kontekstów. Dzięki temu rozpoczynamy przy użyciu czystego stanu przeznaczonego do każdego ponownych prób.

```
var executionStrategy = new SqlAzureExecutionStrategy();

MyConfiguration.SuspendExecutionStrategy = true;

executionStrategy.Execute(
    () =>
{
    using (var db = new BloggingContext())
    {
        using (var trn = db.Database.BeginTransaction())
        {
            db.Blogs.Add(new Blog { Url = "http://msdn.com/data/ef" });
            db.Blogs.Add(new Blog { Url = "http://blogs.msdn.com/adonet" });
            db.SaveChanges();

            db.Blogs.Add(new Blog { Url = "http://twitter.com/efmagicunicorns" });
            db.SaveChanges();

            trn.Commit();
        }
    }
});

MyConfiguration.SuspendExecutionStrategy = false;
```

Obsługa błędów zatwierdzenie transakcji

27.09.2018 • 5 minutes to read • [Edit Online](#)

NOTE

EF6.1 począwszy tylko — funkcje, interfejsów API itp. z opisem na tej stronie zostały wprowadzone w programie Entity Framework 6.1. Jeśli używasz starszej wersji, niektóre lub wszystkie informacje, nie ma zastosowania.

Jako część 6.1 wprowadziliśmy nową funkcję odporności połączenia na platformie EF: możliwość wykrywania i automatycznego odzyskania po awarii przejściowych połączenia mają wpływ na potwierdzenia zatwierdzeń transakcji. Pełne szczegóły tego scenariusza są najlepiej opisać wpis w blogu [Łączność z bazą danych SQL i problem idempotentności](#). Podsumowując scenariusz zakłada, że gdy wyjątek jest zgłoszany podczas zatwierdzania transakcji dwie możliwe przyczyny:

1. Zatwierdzanie transakcji nie powiodło się na serwerze
2. Zatwierdzanie transakcji zakończyło się pomyślnie na serwerze, ale problem z łącznością uniemożliwił powiadomienie o powodzeniu, zanim klient

Po pierwszym sytuacji się dzieje w aplikacji lub użytkownika, można, spróbuj ponownie wykonać operację, ale gdy druga sytuacja ma miejsce należy unikać ponownych prób i aplikacji może automatycznie odzyskać. Żądania jest to, że bez możliwości wykrywania, jaki był rzeczywista Przyczyna, wyjątek został zgłoszony podczas zatwierdzania, aplikacja nie może wybrać właściwą drogą akcji. Nowa funkcja w EF 6.1 umożliwia EF dokładnie z bazą danych, jeśli transakcja zakończyła się pomyślnie i podając właściwą drogą akcji w sposób niewidoczny dla użytkownika.

Za pomocą funkcji

Aby włączyć tę funkcję musi zawierać wywołanie `SetTransactionHandler` w Konstruktorze typu usługi **DbConfiguration**. Jeśli znasz **DbConfiguration**, zobacz [konfiguracja na podstawie kodu](#). Ta funkcja może być używana w połączeniu z automatycznym ponawianiem wprowadzona w EF6, które pomagają w sytuacji, w którym faktycznie nie można zatwierdzić transakcji na serwerze z powodu przejściowego błędu:

```
using System.Data.Entity;
using System.Data.Entity.Infrastructure;
using System.Data.Entity.SqlServer;

public class MyConfiguration : DbConfiguration
{
    public MyConfiguration()
    {
        SetTransactionHandler(SqlProviderServices.ProviderInvariantName, () => new CommitFailureHandler());
        SetExecutionStrategy(SqlProviderServices.ProviderInvariantName, () => new SqlAzureExecutionStrategy());
    }
}
```

Jak są śledzone transakcji

Po włączeniu funkcji EF automatycznie Dodaj nową tabelę w bazie danych o nazwie `_Transactions`. Nowy wiersz jest wstawiana w tej tabeli, każdym razem, gdy transakcja jest tworzona przez EF i tym wierszu są sprawdzane pod kątem istnienia. Jeśli wystąpi awaria transakcji podczas zatwierdzania.

Mimo że EF wykona najlepszy nakład pracy, aby oczyścić wiersze z tabeli, gdy nie są już potrzebne, tabelę można

powiększać, jeśli aplikacja kończy działanie przedwcześnie i z tego powodu, czego potrzebujesz do przeczyzszczenia tabeli ręcznie w niektórych przypadkach.

Sposób obsługi błędów zatwierdzenia z poprzednimi wersjami

Przed EF 6.1 był nie mechanizmem do obsługi zatwierdzenia awarie w produkcie EF. Istnieje kilka sposobów radzenia sobie z tą sytuacją, które mogą być stosowane do poprzednich wersji EF6:

- Opcja 1 — nic nie rób

Prawdopodobieństwo awarii połączenia podczas zatwierdzania transakcji jest niska, dlatego może być akceptowalne, aby aplikacja została właśnie się niepowodzeniem, jeśli rzeczywiście występuje ten problem.

- Opcja 2 — umożliwia resetowanie stanu bazy danych

1. Odrzuć bieżącego kontekstu DbContext
2. Tworzenie nowego typu DbContext i przywrócenia stanu aplikacji z bazy danych
3. Informuje użytkownika, że ostatnia operacja może nie zostały zakończone pomyślnie

- Opcja 3 — ręcznie śledzić transakcji

1. Dodaj tabelę — śledzone w bazie danych używane do śledzenia stanu transakcji.
2. Wstaw wiersz do tabeli na początku każdej transakcji.
3. Jeśli połączenie nie powiedzie się podczas zatwierdzania, sprawdź, czy obecność odpowiedni wiersz w bazie danych.
 - o Jeśli wiersz jest obecny, kontynuował normalne, ponieważ transakcja została pomyślnie zatwierdzona
 - o Jeśli wiersz jest nieobecne, należy użyć strategii wykonywania, aby ponowić próbę wykonania bieżącej operacji.
4. Jeśli zatwierdzenie zakończy się pomyślnie, usuń odpowiedni wiersz w celu uniknięcia wzrostu tabeli.

Ten wpis w blogu zawiera przykładowy kod realizacji tego zadania na platformie Azure SQL.

Wiązanie danych z WinForms

13.09.2018 • 22 minutes to read • [Edit Online](#)

Ten przewodnik krok po kroku pokazano, jak powiązać POCO typy formantów w formularzach systemu Windows (WinForms) w postaci "elementy główne szczegóły". Aplikacja używa platformy Entity Framework do wypełniania obiekty z danymi z bazy danych, śledzenie zmian i utrwalanie danych w bazie danych.

Model definiuje dwa typy, które uczestniczą w relacji jeden do wielu: kategorii (jednostki\głównego) i produktów (zależne\szczegółów). Następnie narzędzia programu Visual Studio są używane do powiązania typów zdefiniowanych w modelu do kontrolki WinForms. Struktura powiązanie danych formularzy WinForm umożliwia nawigacji między powiązane obiekty: Wybieranie wierszy w widoku głównego powoduje, że widok szczegółów aktualizacji za pomocą odpowiednich danych podzielonych.

Zrzuty ekranu i zamieszczone w tym przewodniku są pobierane z programu Visual Studio 2013, ale możesz ukończyć ten przewodnik przy użyciu programu Visual Studio 2012 lub Visual Studio 2010.

Wymagania wstępne

Musisz mieć program Visual Studio 2013, Visual Studio 2012 lub Visual Studio 2010 należy zainstalować w celu przeprowadzenia tego instruktażu.

Jeśli używasz programu Visual Studio 2010 należy zainstalować NuGet. Aby uzyskać więcej informacji, zobacz [Instalowanie systemu NuGet](#).

Tworzenie aplikacji

- Otwórz program Visual Studio
- **Plik —> New -> projektu...**
- Wybierz **Windows** w okienku po lewej stronie i **Windows FormsApplication** w okienku po prawej stronie
- Wprowadź **WinFormswithEFSample** jako nazwę
- Wybierz **OK**

Instalowanie pakietu NuGet programu Entity Framework

- W Eksploratorze rozwiązań kliknij prawym przyciskiem myszy **WinFormswithEFSample** projektu
- Wybierz **Zarządzaj pakietami NuGet...**
- W oknie dialogowym pakiety zarządzania NuGet wybierz **Online** kartę i wybierz polecenie **EntityFramework** pakietu
- Kliknij przycisk **instalacji**

NOTE

Oprócz zestawu EntityFramework dodawany jest również odwołania do System.ComponentModel.DataAnnotations. Jeśli projekt zawiera odwołanie do System.Data.Entity, następnie zostanie on usunięty po zainstalowaniu pakietu platformy EntityFramework. Zestaw System.Data.Entity nie jest już używany w przypadku aplikacji platformy Entity Framework 6.

Implementowanie IListSource dla kolekcji

Właściwości kolekcji musi implementować interfejsu **IListSource** umożliwiające powiązanie dwukierunkowe

danych za pomocą sortowania, korzystając z Windows Forms. W tym celu użyjemy rozszerzenie observablecollection — Dodawanie funkcji IListSource.

- Dodaj **ObservableListSource** klasy do projektu:
 - Kliknij prawym przyciskiem myszy nazwę projektu
 - Wybierz **Add -> nowy element**
 - Wybierz **klasy** i wprowadź **ObservableListSource** nazwy klasy
- Zastąp kod wygenerowany przez domyślną, używając następującego kodu:

Ta klasa umożliwia dwukierunkowe danych powiązania, a także sortowanie. Klasa pochodzi z observablecollection —*<T>* i dodaje Jawną implementację interfejsu *IListSource*. Metoda *GetList()* *IListSource* zaimplementowano by zwrócić implementacje *IBindingList*, która pozostaje zsynchronizowany z observablecollection —. Implementacja *IBindingList* generowane przez *ToBindingList* obsługuje sortowanie. Metoda rozszerzenie *ToBindingList* jest zdefiniowany w zestawie platformy EntityFramework.

```
using System.Collections;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.ComponentModel;
using System.Diagnostics.CodeAnalysis;
using System.Data.Entity;

namespace WinFormswithEFSample
{
    public class ObservableListSource<T> : ObservableCollection<T>, IListSource
        where T : class
    {
        private IBindingList _bindingList;

        bool IListSource.ContainsListCollection { get { return false; } }

        IList IListSource.GetList()
        {
            return _bindingList ?? (_bindingList = this.ToBindingList());
        }
    }
}
```

Zdefiniuj Model

W tym przewodniku, możesz wybrać do zaimplementowania model przy użyciu Code First i projektancie platformy EF. Wykonaj jedną z dwóch poniższych sekcjach.

Opcja 1: Definiowanie modelu za pomocą funkcji Code First

W tej sekcji przedstawiono sposób tworzenia modelu i jego skojarzonej bazy danych za pomocą funkcji Code First. Przejdź do następnej sekcji (**opcja 2: Definiowanie model przy użyciu pierwszej bazy danych**) Jeśli zostanie wykorzystany raczej Database First odwrócić inżynier modelu z bazy danych za pomocą projektanta EF

Korzystając z rozwiązania deweloperskiego Code First zwykle Rozpocznij od pisania klas .NET Framework, które definiują model koncepcyjny (domena).

- Dodaj nową **produktu** klasy do projektu
- Zastąp kod wygenerowany przez domyślną, używając następującego kodu:

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace WinFormswithEFSample
{
    public class Product
    {
        public int ProductId { get; set; }
        public string Name { get; set; }

        public int CategoryId { get; set; }
        public virtual Category Category { get; set; }
    }
}

```

- Dodaj **kategorii** klasy do projektu.
- Zastąp kod wygenerowany przez domyślną, używając następującego kodu:

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace WinFormswithEFSample
{
    public class Category
    {
        private readonly ObservableCollection<Product> _products =
            new ObservableCollection<Product>();

        public int CategoryId { get; set; }
        public string Name { get; set; }
        public virtual ObservableCollection<Product> Products { get { return _products; } }
    }
}

```

Oprócz Definiowanie jednostek, musisz zdefiniować klasę, która pochodzi od klasy **DbContext** i udostępnia **DbSet< TEntity >** właściwości. **DbSet** właściwości umożliwiają kontekstu wiedzieć, jakie typy, które mają zostać uwzględnione w modelu. **DbContext** i **DbSet** typy są definiowane w zestawie platformy EntityFramework.

Wystąpienie typu DbContext pochodzące zarządza obiektami jednostki w czasie wykonywania, zawierającą wypełnianie obiektów przy użyciu danych z bazy danych, zmień śledzenie i przechowywanie danych w bazie danych.

- Dodaj nową **ProductContext** klasy do projektu.
- Zastąp kod wygenerowany przez domyślną, używając następującego kodu:

```

using System;
using System.Collections.Generic;
using System.Data.Entity;
using System.Linq;
using System.Text;

namespace WinFormswithEFSample
{
    public class ProductContext : DbContext
    {
        public DbSet<Category> Categories { get; set; }
        public DbSet<Product> Products { get; set; }
    }
}

```

Skompiluj projekt.

Opcja 2: Definiowanie modelu przy użyciu pierwszej bazy danych

W tej sekcji pokazano, jak użyć pierwszej bazy danych do odtworzenia modelu z bazy danych za pomocą projektanta EF. Jeśli należy wykonać czynności opisane w poprzedniej sekcji (**opcja 1: Definiowanie modelu za pomocą funkcji Code First**), Pomiń tę sekcję i przejdź bezpośrednio do **powolne ładowanie** sekcji.

Utwórz istniejącej bazy danych

Zazwyczaj przeznaczonych do istniejącej bazy danych, zostanie już utworzony, ale w tym przewodniku należy utworzyć bazy danych w celu uzyskania dostępu.

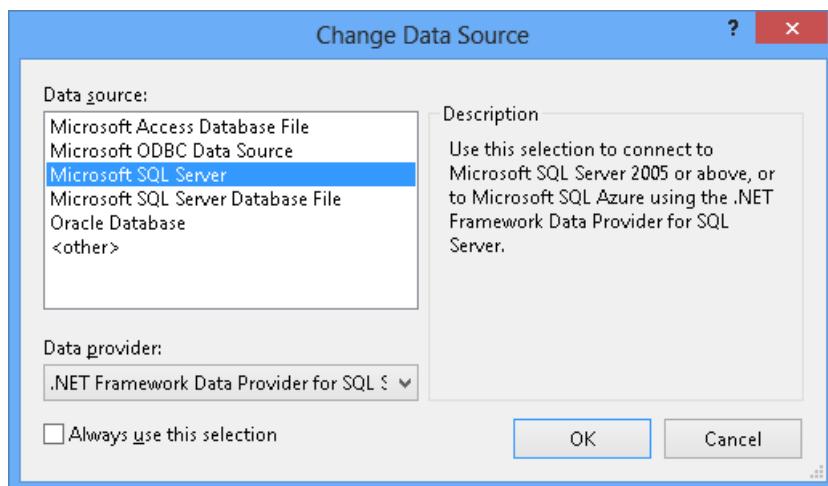
Serwer bazy danych, który został zainstalowany przy użyciu programu Visual Studio różni się zależnie od wersji programu Visual Studio zostały zainstalowane:

- Jeśli używasz programu Visual Studio 2010 zostanie utworzona baza danych programu SQL Express.
- Jeśli używasz programu Visual Studio 2012, a następnie zostanie utworzona [LocalDB](#) bazy danych.

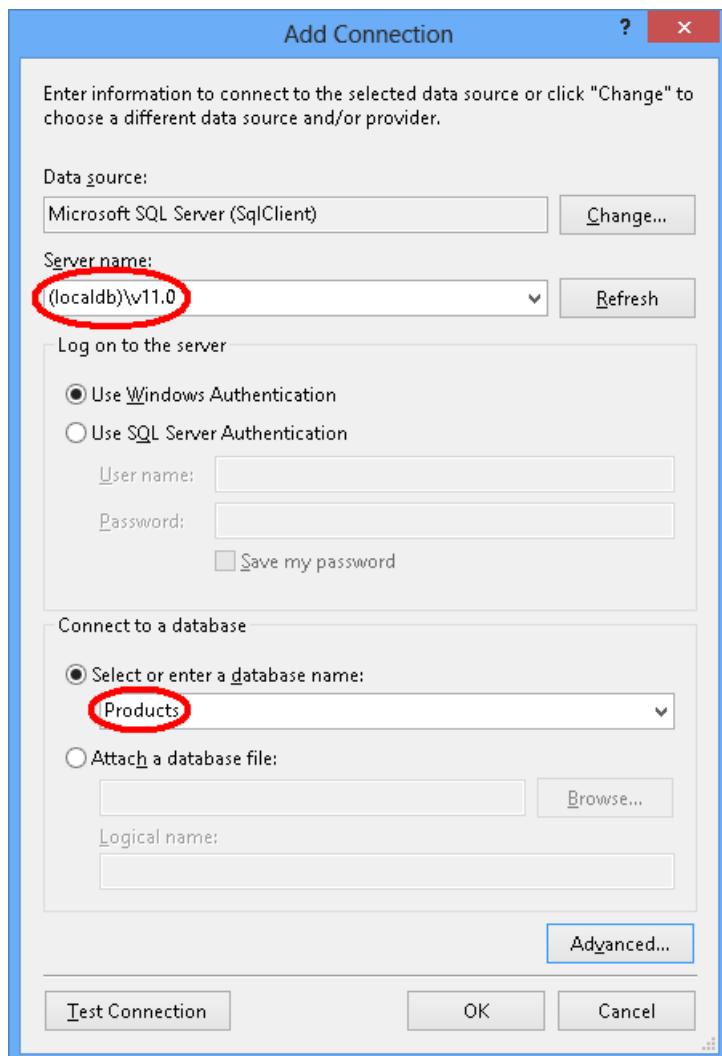
Rozpoczniemy i wygenerować bazę danych.

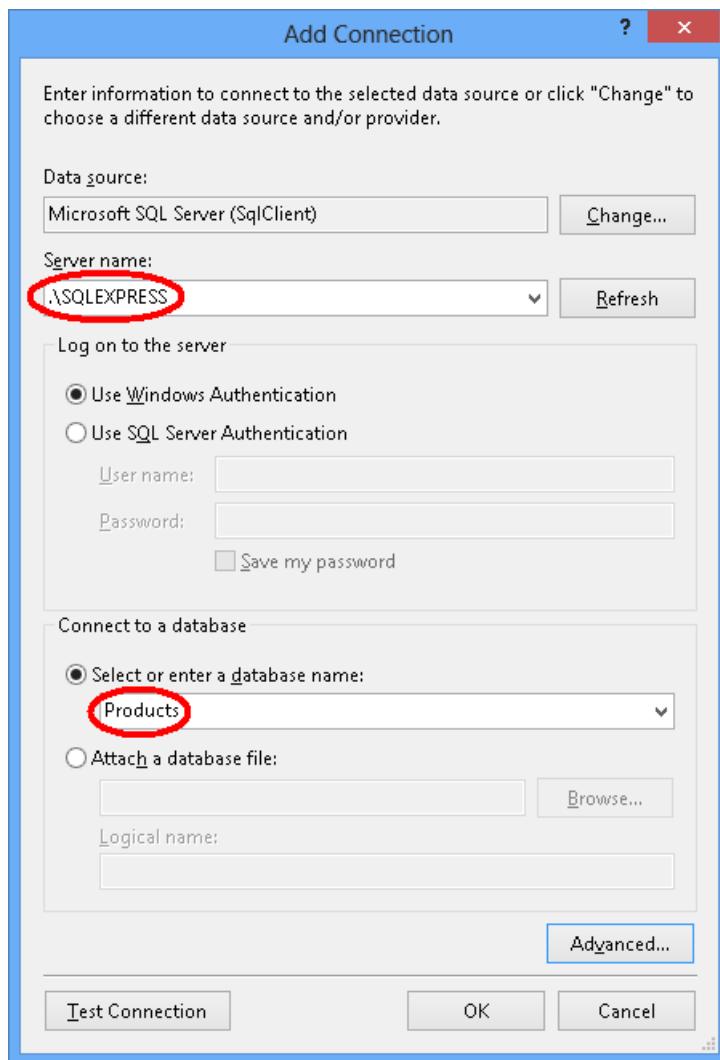
- **Widok —> Eksploratora serwera**

- Kliknij prawym przyciskiem myszy **połączenie danych -> Dodaj połączenie...**
- Jeśli nie jest połączona z bazą danych za pomocą Eksploratora serwera przed, musisz wybrać programu Microsoft SQL Server jako źródło danych

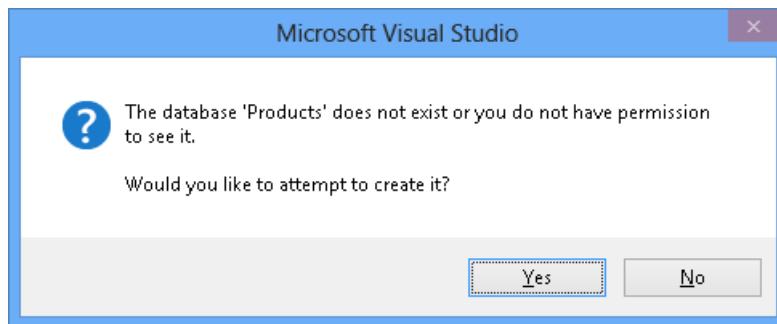


- Łączenie się z LocalDB lub SQL Express, w zależności od tego, który z nich został zainstalowany, a następnie wprowadź **produktów** jako nazwa bazy danych





- Wybierz **OK** i uzyskasz, jeśli chcesz utworzyć nową bazę danych, wybierz opcję **tak**



- Nowe bazy danych będą teraz wyświetlane w Eksploratorze serwera, kliknij prawym przyciskiem myszy na nim i wybierz **nowe zapytanie**
- Skopiuj następujące instrukcje SQL do nowego zapytania, a następnie kliknij prawym przyciskiem myszy, zapytania i wybierz pozycję **wykonania**

```

CREATE TABLE [dbo].[Categories] (
    [CategoryId] [int] NOT NULL IDENTITY,
    [Name] [nvarchar](max),
    CONSTRAINT [PK_dbo.Categories] PRIMARY KEY ([CategoryId])
)

CREATE TABLE [dbo].[Products] (
    [ProductId] [int] NOT NULL IDENTITY,
    [Name] [nvarchar](max),
    [CategoryId] [int] NOT NULL,
    CONSTRAINT [PK_dbo.Products] PRIMARY KEY ([ProductId])
)

CREATE INDEX [IX_CategoryId] ON [dbo].[Products]([CategoryId])

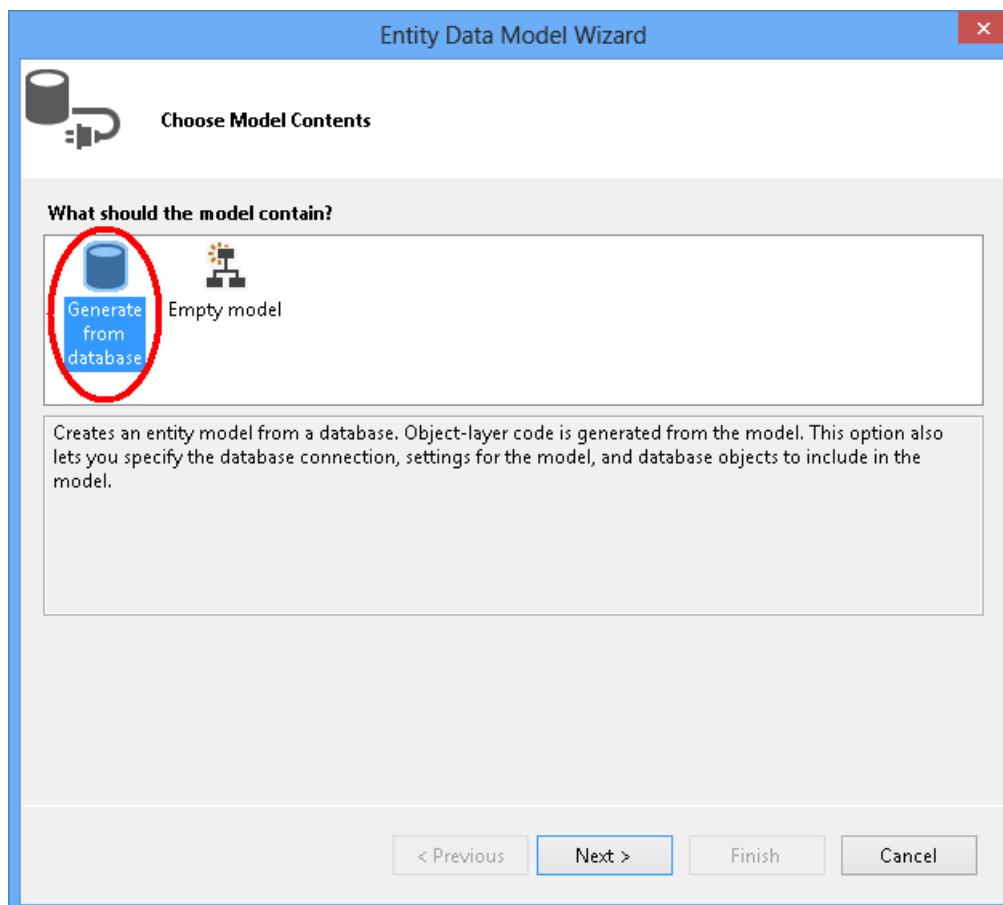
ALTER TABLE [dbo].[Products] ADD CONSTRAINT [FK_dbo.Products_dbo.Categories_CategoryId] FOREIGN KEY
([CategoryId]) REFERENCES [dbo].[Categories] ([CategoryId]) ON DELETE CASCADE

```

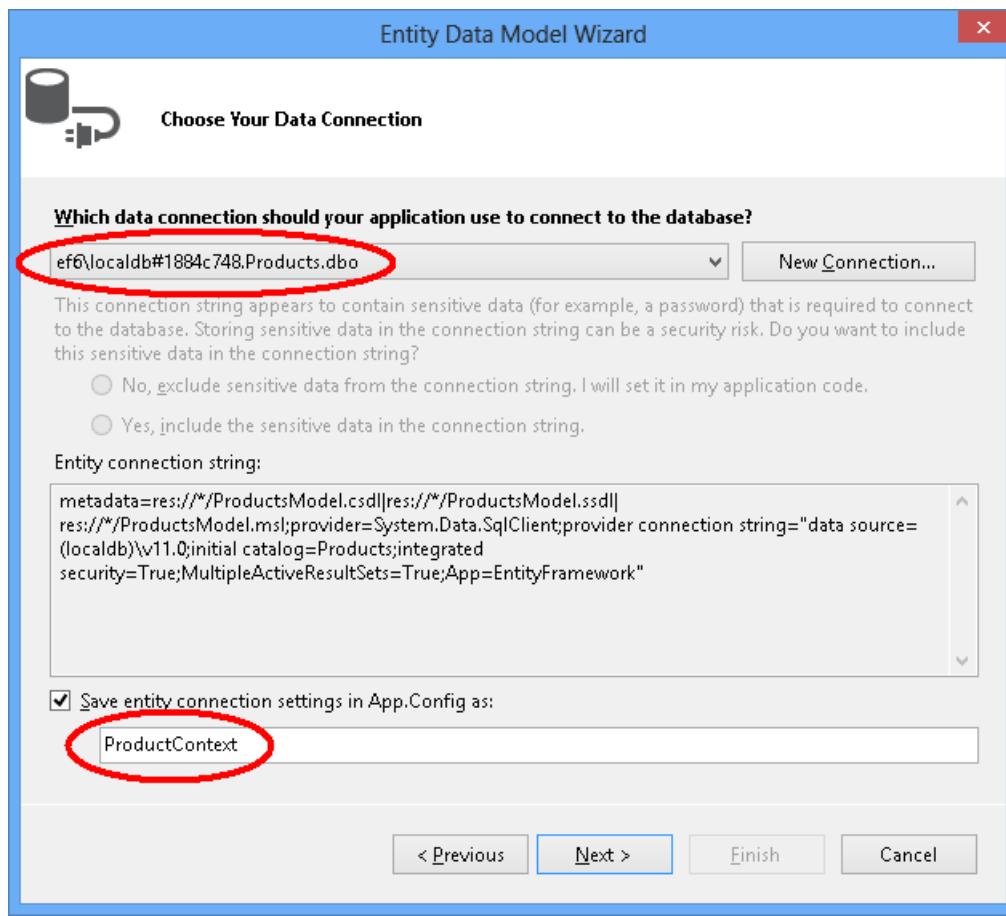
Model odtwarzania

Zamierzamy korzystania z programu Entity Framework Designer, który wchodzi w skład programu Visual Studio, aby utworzyć nasz model.

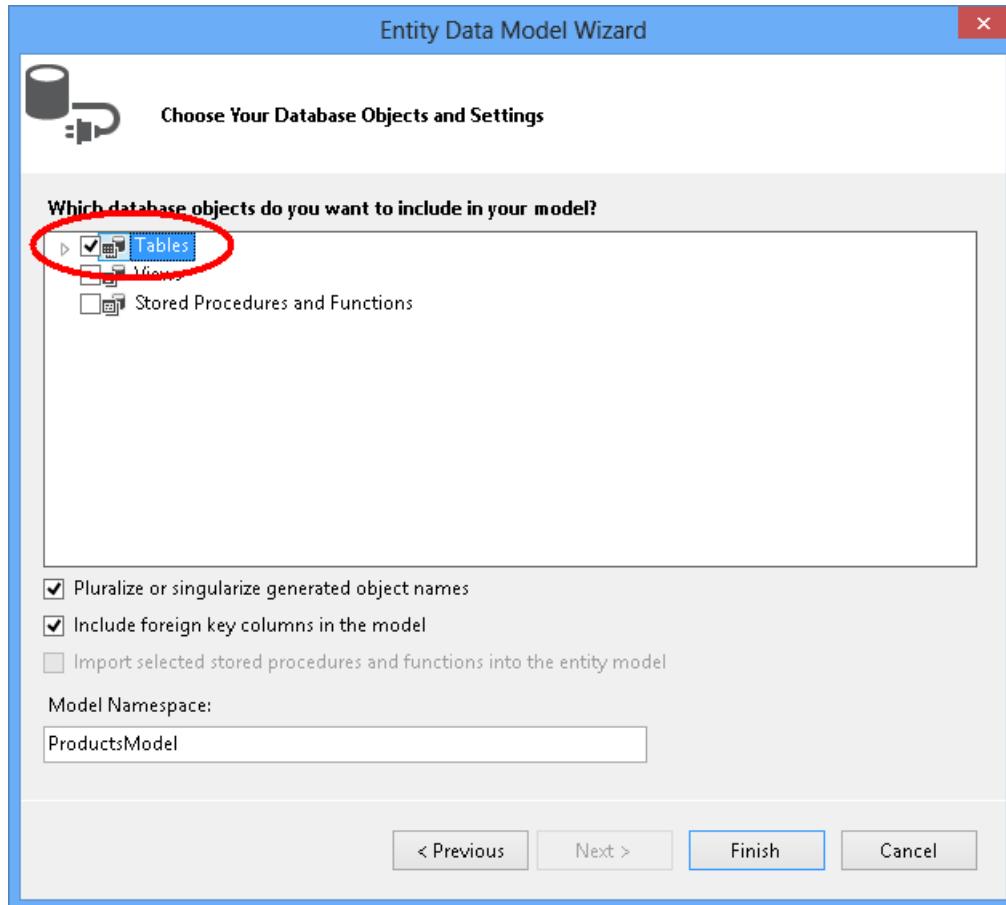
- Projekt —> Dodaj nowy element...
- Wybierz **danych** menu po lewej stronie i następnie **ADO.NET Entity Data Model**
- Wprowadź **ProductModel** jako nazwę i kliknij przycisk **OK**
- Spowoduje to uruchomienie **Kreator modelu Entity Data Model**
- Wybierz **Generuj z bazy danych** i kliknij przycisk **dalej**



- Wybierz połączenie do bazy danych utworzonej w pierwszej sekcji, wprowadź **ProductContext** jako nazwa parametrów połączenia i kliknij przycisk **dalej**



- Kliknij pole wyboru obok "Tabele", aby zaimportować wszystkie tabele i kliknij przycisk "Zakończ"



Po zakończeniu procesu odtwarzania nowy model jest dodawane do projektu i otworzona w celu wyświetlania w Projektancie Entity Framework. Dodano również pliku App.config do projektu przy użyciu szczegółów połączenia dla bazy danych.

Dodatkowe kroki w programie Visual Studio 2010

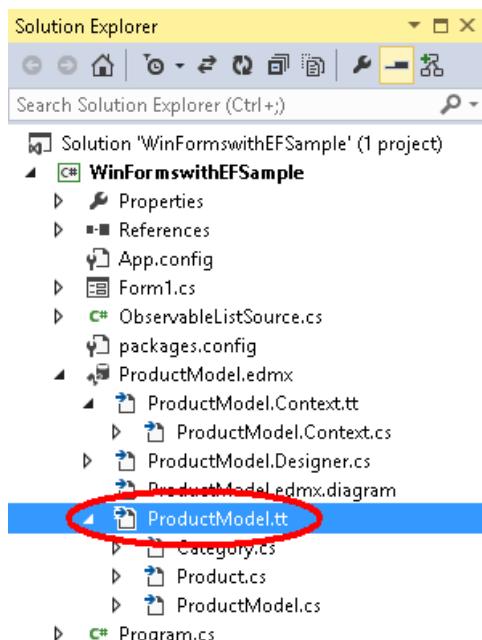
Jeśli pracujesz w programie Visual Studio 2010 należy zaktualizować projektancie platformy EF używać EF generowania kodu.

- Kliknij prawym przyciskiem myszy na puste miejsce modelu w Projektancie platformy EF, a następnie wybierz pozycję **Dodaj element generowanie kodu...**
- Wybierz **szablonów Online** z menu po lewej stronie i wyszukaj **DbContext**
- Wybierz **EF Generator DbContext dla języka C w wersji 6.x#**, wprowadź **ProductsModel** jako nazwę i kliknij przycisk Dodaj

Aktualizowanie generowania kodu dla powiązania danych

EF generuje kod z modelu przy użyciu szablonów T4. Szablonów dostarczanych z programem Visual Studio lub pobrać z galerii programu Visual Studio są przeznaczone do użytku ogólnego przeznaczenia. To oznacza, że jednostki generowane w te szablony mają prosty interfejs `ICollection<T>` właściwości. Jednak podczas wiązania z danymi jest wskazane właściwości kolekcji, które implementują `IListSource`. Dlatego utworzyliśmy klasy `ObservableListSource` powyżej i teraz zamierzamy zmodyfikować szablony Aby użyć tej klasy.

- Otwórz **Eksploratora rozwiązań** i Znajdź **ProductModel.edmx** pliku
- Znajdź **ProductModel.tt** pliku, który będzie można zagnieździć w pliku ProductModel.edmx



- Kliknij dwukrotnie plik ProductModel.tt, aby otworzyć go w edytorze programu Visual Studio
- Znajdź i Zamień dwa wystąpienia "**ICollection**" z "**ObservableListSource**". Te znajdują się w przybliżeniu wiersze 296 i 484.
- Znajdź i Zamień pierwsze wystąpienie ciągu "**hashset** —" z "**ObservableListSource**". To wystąpienie znajduje się w wierszu około 50. **Nie** zastąpić drugie wystąpienie hashset — znaleziono w dalszej części kodu.
- Zapisz plik ProductModel.tt. Powinno to spowodować kodu dla jednostek do ponownego wygenerowania. Jeśli kod nie jest generowany ponownie automatycznie, kliknij prawym przyciskiem myszy ProductModel.tt i wybierz polecenie "Uruchom narzędzie niestandardowe".

Jeśli teraz otworzysz plik Category.cs (która jest zagnieżdzony w ProductModel.tt), wówczas powinien zostać wyświetlony, że kolekcja produktów ma typ **ObservableListSource<produktu>**.

Skompiluj projekt.

Ładowanie z opóźnieniem

Produktów właściwość **kategorii** klasy i **kategorii** właściwość **produkту** klasy są właściwością nawigacji.

Platformy Entity Framework właściwości nawigacji Podaj sposób przechodzenia relację między dwoma typami encji.

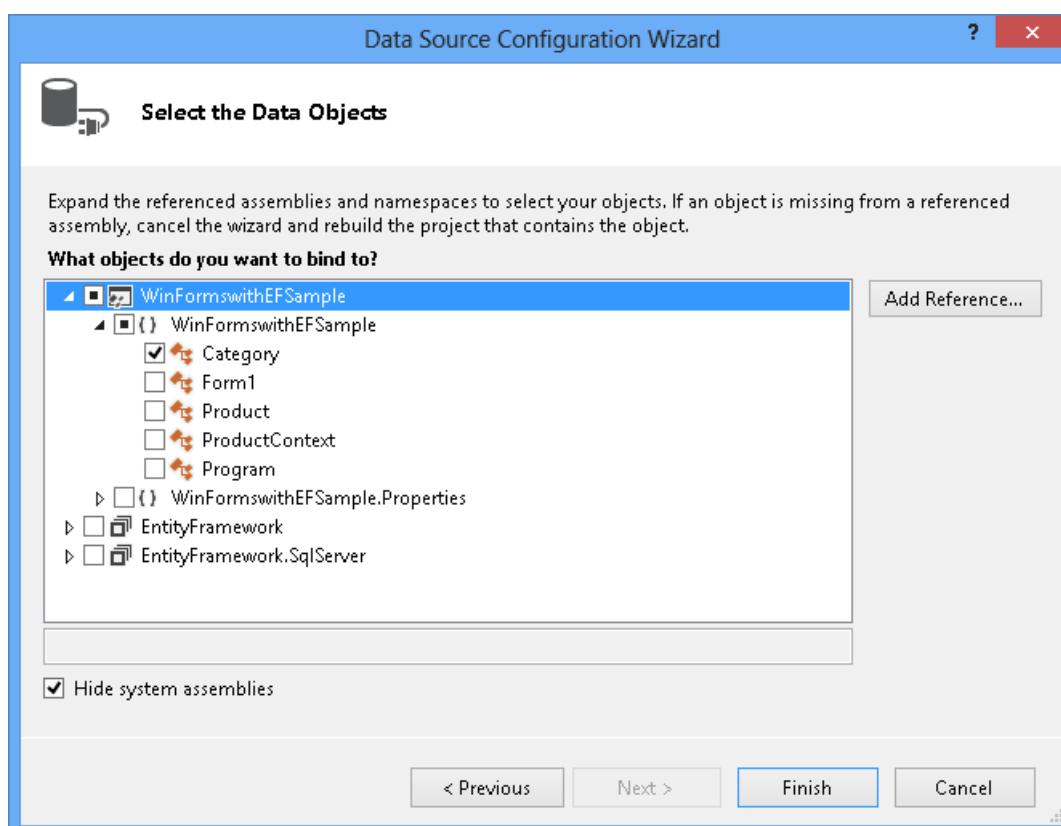
EF daje możliwość ładowanie powiązanych jednostek z bazy danych automatycznie po raz pierwszy możesz uzyskać dostęp do właściwości nawigacji. W przypadku tego typu (nazywane powolne ładowanie) podczas ładowania należy pamiętać, że po raz pierwszy możesz uzyskać dostęp do każdej właściwości nawigacji oddzielnego zapytania będą wykonywane względem bazy danych, jeśli zawartość nie jest już w kontekście.

Korzystając z typów jednostki POCO, EF realizuje powolne ładowanie przez tworzenie wystąpień typów pochodnych serwera proxy podczas wykonywania, a następnie zastępowanie właściwości wirtualnych w Twoich zajęciach, które można dodać punktów zaczepienia ładowania. Aby uzyskać powolne ładowanie powiązanych obiektów, należy zadeklarować nawigacji metody pobierające właściwości jako **publicznych i wirtualnego (Overridable** w języku Visual Basic), i możesz klasy nie może być **zapieczętowanego (NotOverridable** w języku Visual Basic). Przy użyciu bazy danych właściwości nawigacji pierwszy zostaną zastosowane automatycznie wirtualnego, aby umożliwić ładowanie z opóźnieniem. W sekcji Code First Wybrałyśmy Tworzenie właściwości nawigacji wirtualnego z tego samego powodu

Wiązanie obiektu z kontrolkami

Dodawanie klas, które są zdefiniowane w modelu jako źródła danych dla tej aplikacji formularzy WinForms.

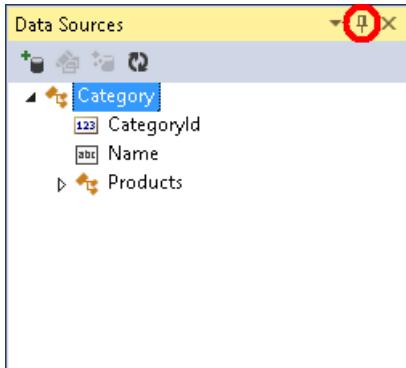
- W menu głównym wybierz **projekt —> Dodaj nowe źródło danych...** (w programie Visual Studio 2010, musisz wybrać **dane —> Dodaj nowe źródło danych...**)
- Wybierz typ źródła danych okna wybierz **obiektu** i kliknij przycisk **dalej**
- Wybierz obiekty danych okna dialogowego rozwijania może **WinFormswithEFSample** dwa razy i wybierz pozycję **kategorii** nie ma potrzeby wybrać źródło danych produktu, ponieważ można je uzyskać do niego w produkcie Właściwość w źródle danych kategorii.



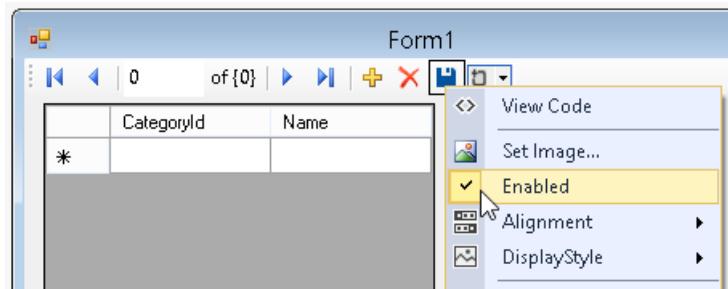
- Kliknij przycisk **Zakończ**. Okna źródła danych nie są wyświetlane, wybierz opcję *** Widok —> inne

Windows -> źródła danych*

- Naciśnij ikonę pinezki, więc okna źródła danych, automatycznie ukrywać. Może być konieczne przycisk Odśwież okno zostało już widoczne.



- W Eksploratorze rozwiązań kliknij dwukrotnie **Form1.cs** plik, aby otworzyć formularz główny w projektancie.
 - Wybierz **kategorii** źródła danych i przeciągnij je na formularzu. Domyślnie nowe DataGridView (**categoryDataGridView**) i formanty nawigacji na pasku narzędzi są dodawane do projektanta. Te kontrolki są powiązane z BindingSource (**categoryBindingSource**) i powiązanie Navigator (**categoryBindingNavigator**) składników, które są również tworzone.
 - Edytuj kolumny w **categoryDataGridView**. Chcemy ustawić **CategoryId** kolumny tylko do odczytu. Wartość **CategoryId** właściwości został wygenerowany przez bazę danych, po oszczędzamy danych.
 - Kliknij prawym przyciskiem myszy formant DataGridView i wybierz pozycję Edytuj kolumny...
 - Wybierz kolumnę CategoryId i wartość True tylko do odczytu
 - Naciśnij przycisk OK.
 - Wybierz produkty z kategorii źródła danych i przeciągnij go na formularzu. ProductDataGridView i productBindingSource są dodawane do formularza.
 - Edytuj kolumny na productDataGridView. Chcemy ukryć kolumny CategoryId i kategorii i ustawić ProductId tylko do odczytu. Wartość właściwości ProductId jest generowany przez bazę danych, po oszczędzamy danych.
 - Kliknij prawym przyciskiem myszy formant DataGridView i wybierz **Edytowanie kolumn...**
 - Wybierz **ProductId** kolumny oraz zestaw **tylko do odczytu** do **True**.
 - Wybierz **CategoryId** kolumny i naciśnij klawisz **Usuń** przycisku. Wykonaj te czynności z **kategorii** kolumny.
 - Naciśnij klawisz **OK**.
- Do tej pory firma Microsoft skojarzony naszych formantów DataGridView BindingSource składniki w projektancie. W następnej sekcji dodamy kod kodu można ustawić categoryBindingSource.DataSource do kolekcji jednostek, które obecnie są śledzone przez DbContext. Gdy przeciągnąć i porzucony produktów z kategorii, WinForms trwało Dbamy konfigurowania właściwości productsBindingSource.DataSource categoryBindingSource i productsBindingSource.DataMember właściwości produktów. Ze względu na to powiązanie tylko produkty należące do aktualnie wybranej kategorii będą wyświetlane w productDataGridView.
- Włącz **Zapisz** przycisk na pasku nawigacji, klikając prawym przyciskiem myszy i wybierając **włączone**.



- Dodaj program obsługi zdarzeń zapisywania przycisku przez dwukrotne kliknięcie przycisku. Spowoduje to dodanie obsługi zdarzeń i wyświetlić kod związany z formularza. Kod **categoryBindingNavigatorSaveItem_Click** programu obsługi zdarzeń zostanie dodana w następnej sekcji.

Dodaj kod, który obsługuje interakcji z danych

Teraz dodamy kod, aby użyć ProductContext przeprowadzić dostęp do danych. Aktualizowanie kodu dla okna głównego formularza, jak pokazano poniżej.

Ten kod deklaruje długotrwałych wystąpienie ProductContext. Obiekt ProductContext jest używany do wykonywania zapytań i zapisać dane w bazie danych. Metoda Dispose() w wystąpieniu ProductContext jest następnie wywoływana z przeciążonej OnClosing. Komentarze w kodzie zawierają szczegółowe informacje o dany kod realizuje.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Data.Entity;

namespace WinFormswithEFSample
{
    public partial class Form1 : Form
    {
        ProductContext _context;
        public Form1()
        {
            InitializeComponent();
        }

        protected override void OnLoad(EventArgs e)
        {
            base.OnLoad(e);
            _context = new ProductContext();

            // Call the Load method to get the data for the given DbSet
            // from the database.
            // The data is materialized as entities. The entities are managed by
            // the DbContext instance.
            _context.Categories.Load();

            // Bind the categoryBindingSource.DataSource to
            // all the Unchanged, Modified and Added Category objects that
            // are currently tracked by the DbContext.
            // Note that we need to call ToBindingList() on the
            // ObservableCollection< TEntity > returned by
            // the DbSet.Local property to get the BindingList< T >
            // in order to facilitate two-way binding in WinForms.
        }
    }
}
```

```

        this.categoryBindingSource.DataSource =
            _context.Categories.Local.ToBindingList();
    }

    private void categoryBindingNavigatorSaveItem_Click(object sender, EventArgs e)
    {
        this.Validate();

        // Currently, the Entity Framework doesn't mark the entities
        // that are removed from a navigation property (in our example the Products)
        // as deleted in the context.
        // The following code uses LINQ to Objects against the Local collection
        // to find all products and marks any that do not have
        // a Category reference as deleted.
        // The ToList call is required because otherwise
        // the collection will be modified
        // by the Remove call while it is being enumerated.
        // In most other situations you can do LINQ to Objects directly
        // against the Local property without using ToList first.
        foreach (var product in _context.Products.Local.ToList())
        {
            if (product.Category == null)
            {
                _context.Products.Remove(product);
            }
        }

        // Save the changes to the database.
        this._context.SaveChanges();

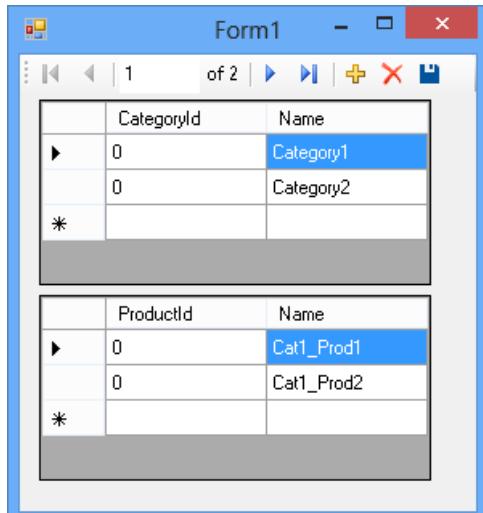
        // Refresh the controls to show the values
        // that were generated by the database.
        this.categoryDataGridView.Refresh();
        this.productsDataGridView.Refresh();
    }

    protected override void OnClosing(CancelEventArgs e)
    {
        base.OnClosing(e);
        this._context.Dispose();
    }
}
}

```

Testowanie aplikacji Windows Forms

- Kompiluj i uruchom aplikację i przetestowaj funkcji.

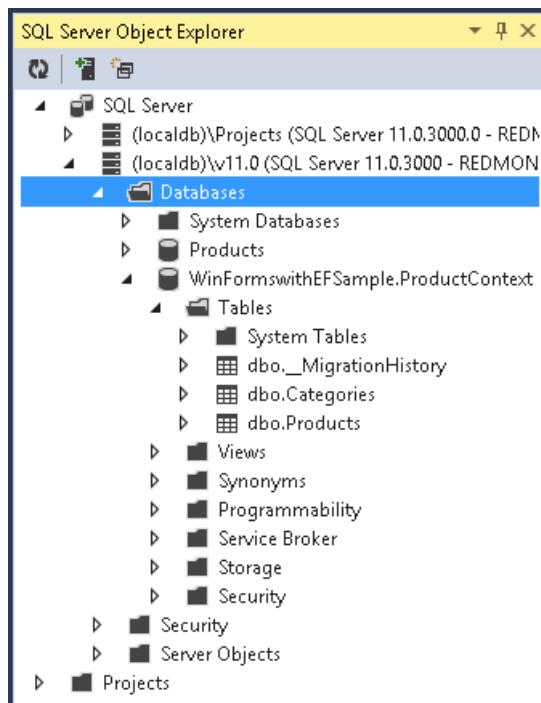


- Po zapisaniu wygenerowane klucze są wyświetlane na ekranie.

	CategoryId	Name
▶	1	Category1
	2	Category2
*		

	ProductId	Name
▶	1	Cat1_Prod1
	2	Cat1_Prod2
*		

- Jeśli została użyta Code First, to będzie również informacja, że **WinFormswithEFSample.ProductContext** baza danych została utworzona dla Ciebie.



Powiązanie danych przy użyciu platformy WPF

04.10.2018 • 23 minutes to read • [Edit Online](#)

Ten przewodnik krok po kroku pokazano, jak powiązać POCO typy kontrolek WPF w postaci "elementy główne szczegóły". Aplikacja używa interfejsów API programu Entity Framework do wypełniania obiekty z danymi z bazy danych, śledzenie zmian i utrwalanie danych w bazie danych.

Model definiuje dwa typy, które uczestniczą w relacji jeden do wielu: **kategorii** (jednostki\głównego) i **produkту** (zależne\szczegółów). Następnie narzędzia programu Visual Studio są używane do powiązania typy zdefiniowane w modelu, który ma kontrolkę WPF. Struktura powiązanie danych WPF umożliwia nawigacji między powiązane obiekty: Wybieranie wierszy w widoku głównego powoduje, że widok szczegółów aktualizacji za pomocą odpowiednich danych podanych.

Zrzuty ekranu i zamieszczone w tym przewodniku są pobierane z programu Visual Studio 2013, ale możesz ukończyć ten przewodnik przy użyciu programu Visual Studio 2012 lub Visual Studio 2010.

Użyj opcji "Object" do tworzenia źródeł danych w WPF

Za pomocą poprzedniej wersji programu Entity Framework użyliśmy zalecamy używanie **bazy danych** podczas tworzenia nowego źródła danych oparty na modelu utworzonych za pomocą projektancie platformy EF. Taka konfiguracja była ponieważ projektant wygeneruje kontekstu, który pochodzi od obiektu ObjectContext i klas jednostek, które pochodną EntityObject. Przy użyciu opcji bazy danych umożliwia pisanie kodu najlepsze do interakcji z tym powierzchni interfejsu API.

Projektant EF dla programu Visual Studio 2012 i Visual Studio 2013 generowanie kontekstu, który pochodzi od typu DbContext wraz z prostego klas obiektów POCO. Za pomocą programu Visual Studio 2010, zaleca się zamianę do szablonu generowania kodu, który używa typu DbContext, zgodnie z opisem w dalszej części tego przewodnika.

Korzystając z powierzchni interfejsu API typu DbContext należy użyć **obiektu** opcji podczas tworzenia nowego źródła danych, jak pokazano w tym przewodniku.

Jeśli to konieczne, możesz [powrócić do generowania kodu na podstawie obiektu ObjectContext](#) dla modeli utworzonych za pomocą projektanta EF.

Wymagania wstępne

Musisz mieć program Visual Studio 2013, Visual Studio 2012 lub Visual Studio 2010 należy zainstalować w celu przeprowadzenia tego instruktażu.

Jeśli używasz programu Visual Studio 2010 należy zainstalować NuGet. Aby uzyskać więcej informacji, zobacz [Instalowanie systemu NuGet](#).

Tworzenie aplikacji

- Otwórz program Visual Studio
- **Plik —> New -> projektu...**
- Wybierz **Windows** w okienku po lewej stronie i **WPFApplication** w okienku po prawej stronie
- Wprowadź **WPFwithEFSample** jako nazwę
- Wybierz **OK**

Instalowanie pakietu NuGet programu Entity Framework

- W Eksploratorze rozwiązań kliknij prawym przyciskiem myszy **WinFormswithEFSample** projektu
- Wybierz **Zarządzaj pakietami NuGet...**
- W oknie dialogowym pakiety zarządzania NuGet wybierz **Online** kartę i wybierz polecenie **EntityFramework** pakietu
- Kliknij przycisk **instalacji**

NOTE

Oprócz zestawu EntityFramework dodawany jest również odwołania do System.ComponentModel.DataAnnotations. Jeśli projekt zawiera odwołanie do System.Data.Entity, następnie zostanie on usunięty po zainstalowaniu pakietu platformy EntityFramework. Zestaw System.Data.Entity nie jest już używany w przypadku aplikacji platformy Entity Framework 6.

Zdefiniuj Model

W tym przewodniku, możesz wybrać do zaimplementowania model przy użyciu Code First i projektancie platformy EF. Wykonaj jedną z dwóch poniższych sekcji.

Opcja 1: Definiowanie modelu za pomocą funkcji Code First

W tej sekcji przedstawiono sposób tworzenia modelu i jego skojarzonej bazy danych za pomocą funkcji Code First. Przejdź do następnej sekcji (**opcja 2: Definiowanie modelu przy użyciu pierwszej bazy danych**) Jeśli zostanie wykorzystany raczej Database First odwrócić inżynier modelu z bazy danych za pomocą projektanta EF

Korzystając z rozwiązania deweloperskiego Code First zwykle Rozpocznij od pisania klas .NET Framework, które definiują model koncepcyjny (domena).

- Dodaj nową klasę do **WPFwithEFSample**:
 - Kliknij prawym przyciskiem myszy nazwę projektu
 - Wybierz **Dodaj**, następnie **nowy element**
 - Wybierz **klasy** i wprowadź **produkту** nazwy klasy
- Zastąp **produkту** klasy definicji z następującym kodem:

```

namespace WPFwithEFSample
{
    public class Product
    {
        public int ProductId { get; set; }
        public string Name { get; set; }

        public int CategoryId { get; set; }
        public virtual Category Category { get; set; }
    }
}

```

- Add a ****Category**** class with the following definition:

```

using System.Collections.ObjectModel;

namespace WPFwithEFSample
{
    public class Category
    {
        public Category()
        {
            this.Products = new ObservableCollection<Product>();
        }

        public int CategoryId { get; set; }
        public string Name { get; set; }

        public virtual ObservableCollection<Product> Products { get; private set; }
    }
}

```

Produktów właściwość **kategorii** klasy i **kategorii** właściwość **produkту** klasy są właściwości nawigacji.

Platformy Entity Framework właściwości nawigacji Podaj sposób przechodzenia relację między dwoma typami encji.

Oprócz definiowania jednostek, musisz zdefiniować klasę, która pochodzi od typu DbContext i udostępnia DbSet< TEntity > właściwości. DbSet< TEntity > właściwości umożliwiają kontekstu wiedzieć, jakie typy, które mają zostać uwzględnione w modelu.

Wystąpienie typu DbContext pochodzące zarządza obiektami jednostki w czasie wykonywania, zawierającą wypełnianie obiektów przy użyciu danych z bazy danych, zmień śledzenie i przechowywanie danych w bazie danych.

- Dodaj nową **ProductContext** klasy do projektu przy użyciu następujących definicji:

```

using System.Data.Entity;

namespace WPFwithEFSample
{
    public class ProductContext : DbContext
    {
        public DbSet<Category> Categories { get; set; }
        public DbSet<Product> Products { get; set; }
    }
}

```

Skompiluj projekt.

Opcja 2: Definiowanie modelu przy użyciu pierwszej bazy danych

W tej sekcji pokazano, jak użyć pierwszej bazy danych do odtworzenia modelu z bazy danych za pomocą projektanta EF. Jeśli należy wykonać czynności opisane w poprzedniej sekcji (**opcja 1: Definiowanie modelu za**

pomocą funkcji Code First), Pomiń tę sekcję i przejdź bezpośrednio do **powolne ładowanie** sekcji.

Utwórz istniejącej bazy danych

Zazwyczaj przeznaczonych do istniejącej bazy danych, zostanie już utworzony, ale w tym przewodniku należy utworzyć bazy danych w celu uzyskania dostępu.

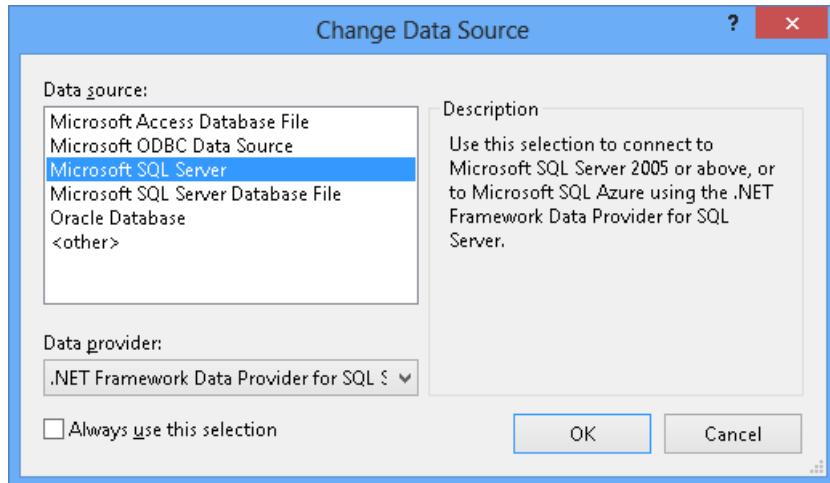
Serwer bazy danych, który został zainstalowany przy użyciu programu Visual Studio różni się zależnie od wersji programu Visual Studio zostały zainstalowane:

- Jeśli używasz programu Visual Studio 2010 zostanie utworzona baza danych programu SQL Express.
- Jeśli używasz programu Visual Studio 2012, a następnie zostanie utworzona [LocalDB](#) bazy danych.

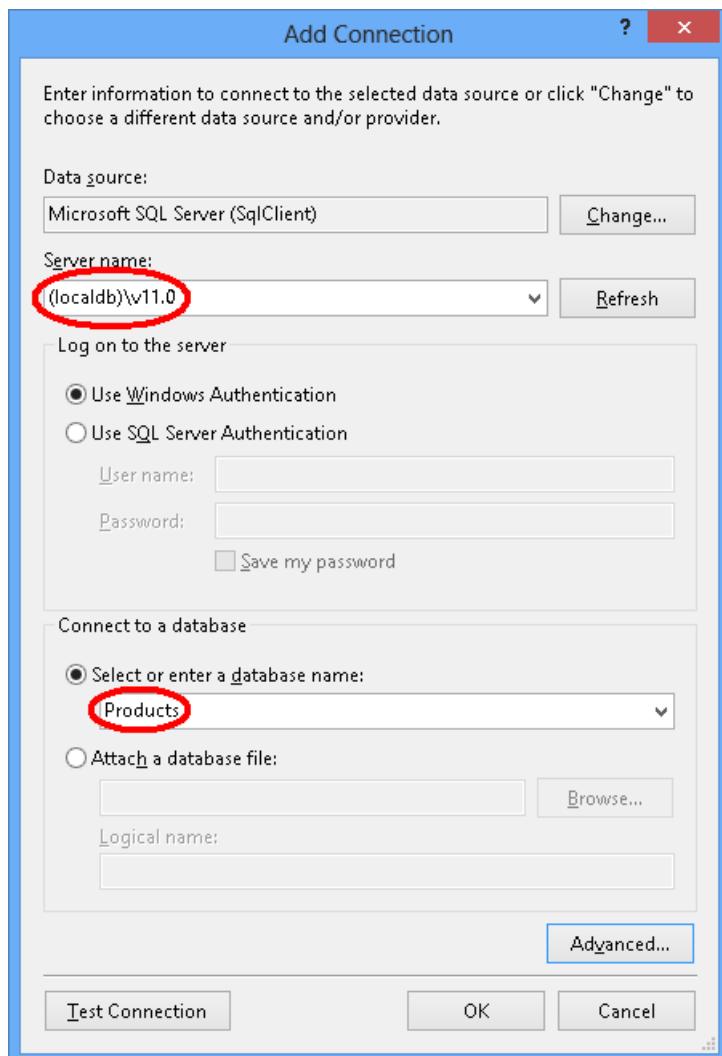
Rozpoczniemy i wygenerować bazę danych.

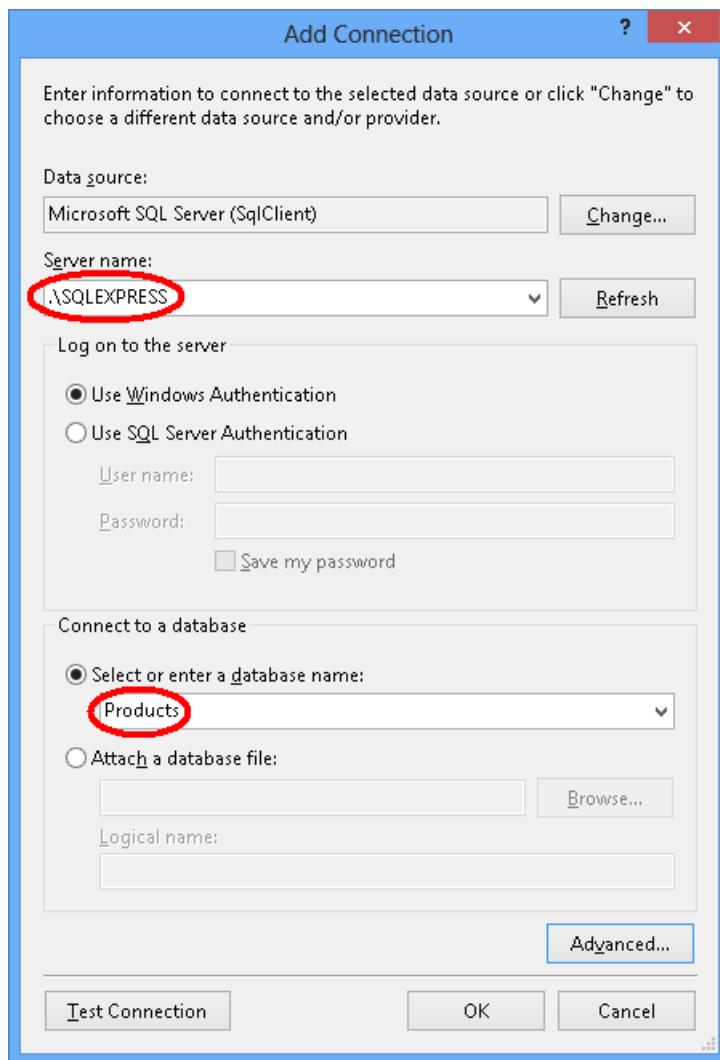
- **Widok —> Eksploratora serwera**

- Kliknij prawym przyciskiem myszy **połączeń danych** -> **Dodaj połaczenie...**
- Jeśli nie jest połączona z bazą danych za pomocą Eksploratora serwera przed, musisz wybrać programu Microsoft SQL Server jako źródło danych



- Łączenie się z LocalDB lub SQL Express, w zależności od tego, który z nich został zainstalowany, a następnie wprowadź **produktów** jako nazwa bazy danych





- Wybierz **OK** i uzyskasz, jeśli chcesz utworzyć nową bazę danych, wybierz opcję **tak**



- Nowe bazy danych będą teraz wyświetlane w Eksploratorze serwera, kliknij prawym przyciskiem myszy na nim i wybierz **nowe zapytanie**
- Skopiuj następujące instrukcje SQL do nowego zapytania, a następnie kliknij prawym przyciskiem myszy, zapytania i wybierz pozycję **wykonania**

```

CREATE TABLE [dbo].[Categories] (
    [CategoryId] [int] NOT NULL IDENTITY,
    [Name] [nvarchar](max),
    CONSTRAINT [PK_dbo.Categories] PRIMARY KEY ([CategoryId])
)

CREATE TABLE [dbo].[Products] (
    [ProductId] [int] NOT NULL IDENTITY,
    [Name] [nvarchar](max),
    [CategoryId] [int] NOT NULL,
    CONSTRAINT [PK_dbo.Products] PRIMARY KEY ([ProductId])
)

CREATE INDEX [IX_CategoryId] ON [dbo].[Products]([CategoryId])

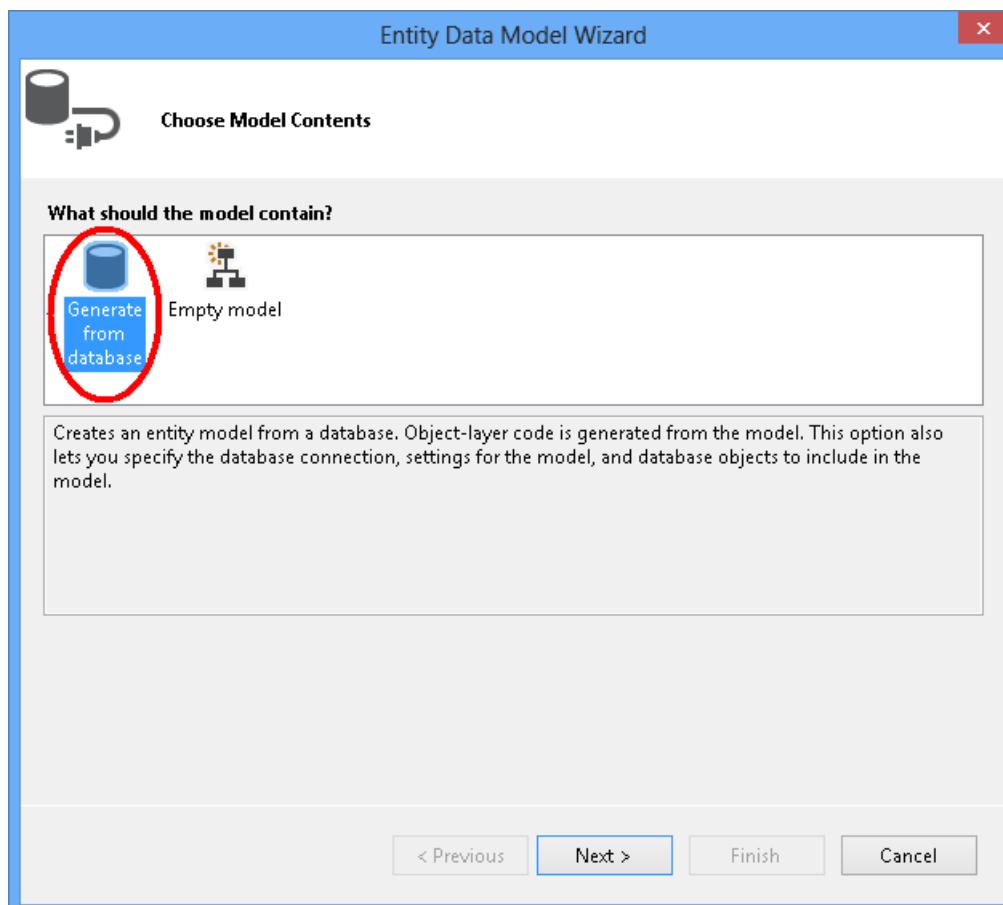
ALTER TABLE [dbo].[Products] ADD CONSTRAINT [FK_dbo.Products_dbo.Categories_CategoryId] FOREIGN KEY
([CategoryId]) REFERENCES [dbo].[Categories] ([CategoryId]) ON DELETE CASCADE

```

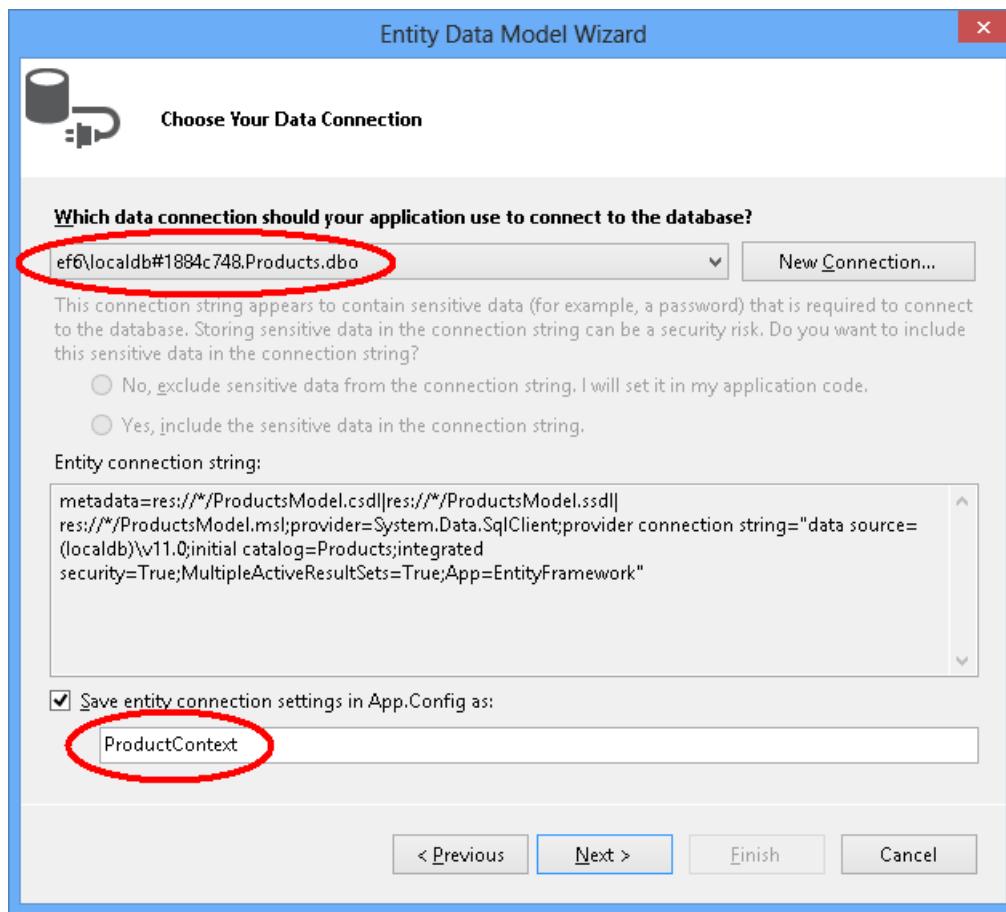
Model odtwarzania

Zamierzamy korzystania z programu Entity Framework Designer, który wchodzi w skład programu Visual Studio, aby utworzyć nasz model.

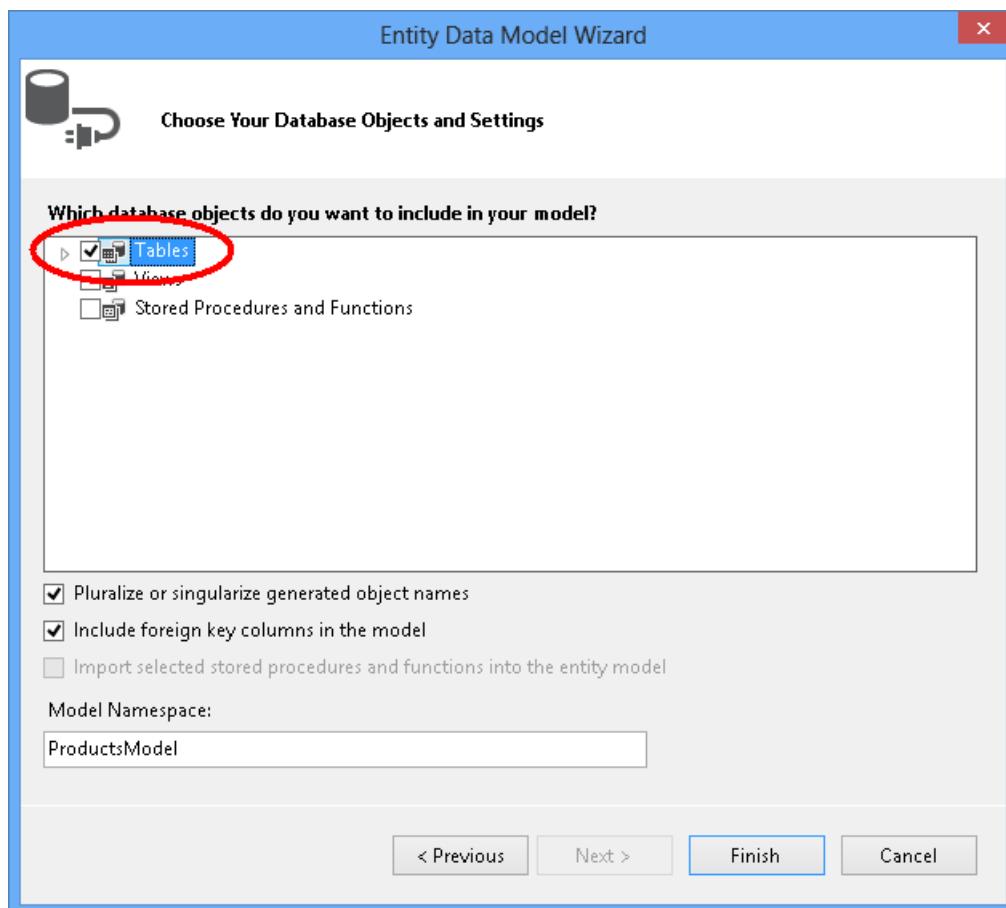
- Projekt —> Dodaj nowy element...
- Wybierz **danych** menu po lewej stronie i następnie **ADO.NET Entity Data Model**
- Wprowadź **ProductModel** jako nazwę i kliknij przycisk **OK**
- Spowoduje to uruchomienie **Kreator modelu Entity Data Model**
- Wybierz **Generuj z bazy danych** i kliknij przycisk **dalej**



- Wybierz połączenie do bazy danych utworzonej w pierwszej sekcji, wprowadź **ProductContext** jako nazwa parametrów połączenia i kliknij przycisk **dalej**



- Kliknij pole wyboru obok "Tabele", aby zaimportować wszystkie tabele i kliknij przycisk "Zakończ"



Po zakończeniu procesu odtwarzania nowy model jest dodawane do projektu i otworzona w celu wyświetlania w Projektancie Entity Framework. Dodano również pliku App.config do projektu przy użyciu szczegółów połączenia dla bazy danych.

Dodatkowe kroki w programie Visual Studio 2010

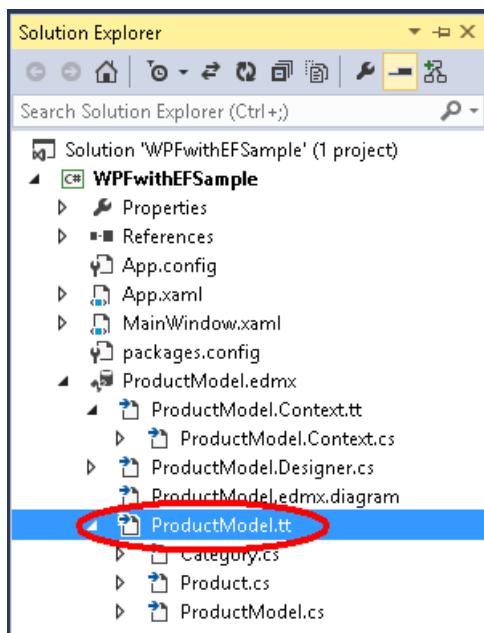
Jeśli pracujesz w programie Visual Studio 2010 należy zaktualizować projektancie platformy EF używać EF generowania kodu.

- Kliknij prawym przyciskiem myszy na puste miejsce modelu w Projektancie platformy EF, a następnie wybierz pozycję **Dodaj element generowanie kodu...**
- Wybierz **szablonów Online** z menu po lewej stronie i wyszukaj **DbContext**
- Wybierz **EF Generator DbContext dla języka C w wersji 6.x#**, wprowadź **ProductsModel** jako nazwę i kliknij przycisk Dodaj

Aktualizowanie generowania kodu dla powiązania danych

EF generuje kod z modelu przy użyciu szablonów T4. Szablonów dostarczanych z programem Visual Studio lub pobrać z galerii programu Visual Studio są przeznaczone do użytku ogólnego przeznaczenia. To oznacza, że jednostki generowane w te szablony mają prosty interfejs `ICollection<T>` właściwości. Jednak podczas ustalania danych powiązania, za pomocą WPF jest pożądane, aby użyć **observablecollection** — dla właściwości kolekcji, tak że WPF można śledzenie pracy do kolekcji. W tym celu firma Microsoft będzie modyfikować szablon do użycia z observablecollection —.

- Otwórz **Eksploratora rozwiązań** i Znajdź **ProductModel.edmx** pliku
- Znajdź **ProductModel.tt** pliku, który będzie można zagnieździć w pliku ProductModel.edmx



- Kliknij dwukrotnie plik ProductModel.tt, aby otworzyć go w edytorze programu Visual Studio
- Znajdź i Zamień dwa wystąpienia "**ICollection**" z "**observablecollection** —". Są to znajduje się mniej więcej w wierszach 296 i 484.
- Znajdź i Zamień pierwsze wystąpienie ciągu "**hashset** —" z "**observablecollection** —". To wystąpienie znajduje się mniej więcej wierszu 50. **Nie** zastąpić drugie wystąpienie hashset — znaleziono w dalszej części kodu.
- Znajdź i Zamień tylko wystąpienie "**System.Collections.Generic**" z "**System.Collections.ObjectModel**". To znajduje się mniej więcej wierszu 424.
- Zapisz plik ProductModel.tt. Powinno to spowodować kodu dla jednostek do ponownego wygenerowania. Jeśli kod nie jest generowany ponownie automatycznie, kliknij prawym przyciskiem myszy ProductModel.tt i wybierz polecenie "Uruchom narzędzie niestandardowe".

Jeśli teraz otworzysz plik Category.cs (która jest zagnieżdzony w ProductModel.tt), wówczas powinien zostać wyświetlony, że kolekcja produktów ma typ **observablecollection —<produktu>**.

Skompiluj projekt.

Ładowanie z opóźnieniem

Produktów właściwość **kategorii** klasy i **kategorii** właściwość **produkту** klasy są właściwości nawigacji. Platformy Entity Framework właściwości nawigacji Podaj sposób przechodzenia relację między dwoma typami encji.

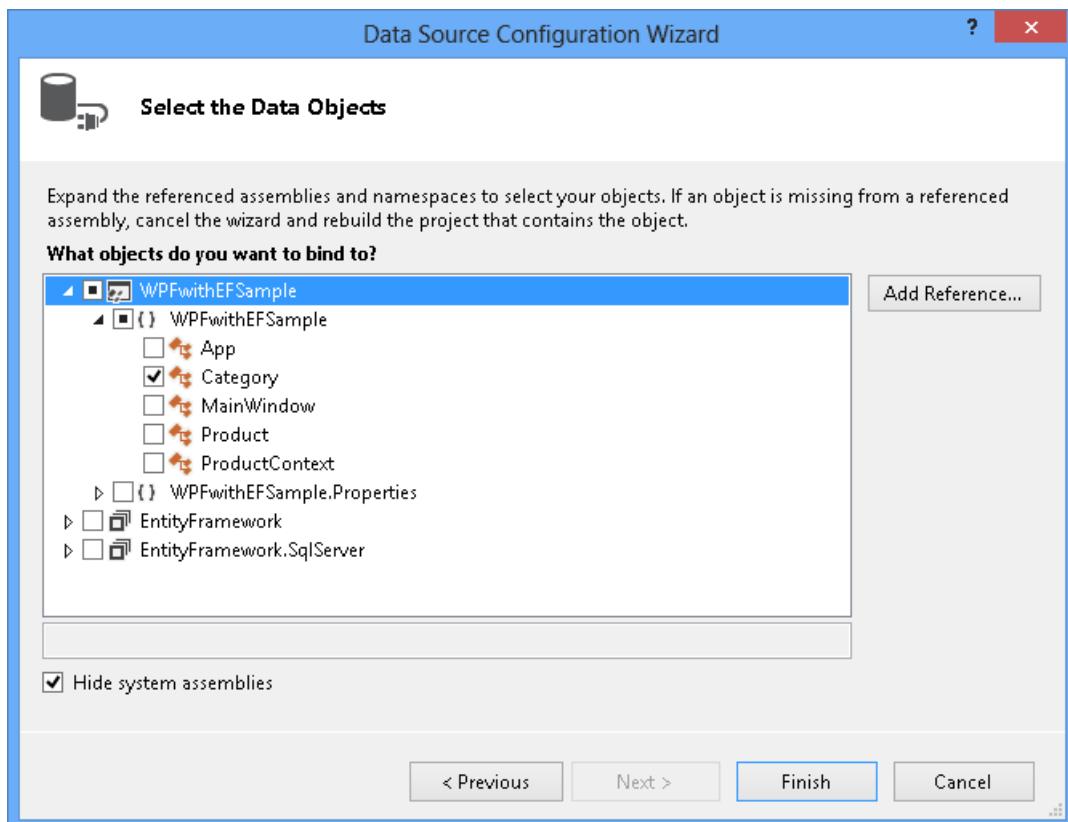
EF daje możliwość ładowanie powiązanych jednostek z bazy danych automatycznie po raz pierwszy możesz uzyskać dostęp do właściwości nawigacji. W przypadku tego typu (nazywane powolne ładowanie) podczas ładowania należy pamiętać, że po raz pierwszy możesz uzyskać dostęp do każdej właściwości nawigacji oddzielnego zapytania będą wykonywane względem bazy danych, jeśli zawartość nie jest już w kontekście.

Korzystając z typów jednostki POCO, EF realizuje powolne ładowanie przez tworzenie wystąpień typów pochodnych serwera proxy podczas wykonywania, a następnie zastępowanie właściwości wirtualnych w Twoich zajęciach, które można dodać punktów zaczepienia ładowania. Aby uzyskać powolne ładowanie powiązanych obiektów, należy zadeklarować nawigacji metody pobierające właściwości jako **publicznych i wirtualnego (Overridable** w języku Visual Basic), i możesz klasy nie może być **zapieczętowanego (NotOverridable** w języku Visual Basic). Przy użyciu bazy danych właściwości nawigacji pierwszy zostaną zastosowane automatycznie wirtualnego, aby umożliwić ładowanie z opóźnieniem. W sekcji Code First Wybraliśmy Tworzenie właściwości nawigacji wirtualnego z tego samego powodu

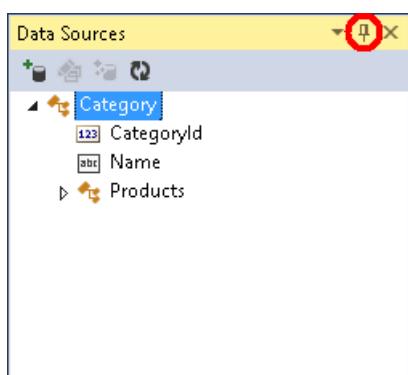
Wiązanie obiektu z kontrolkami

Dodawanie klas, które są zdefiniowane w modelu jako źródła danych dla tej aplikacji WPF.

- Kliknij dwukrotnie **MainWindow.xaml** w Eksploratorze rozwiązań, aby otworzyć formularz główny
- W menu głównym wybierz **projekt —> Dodaj nowe źródło danych...** (w programie Visual Studio 2010, musisz wybrać **dane —> Dodaj nowe źródło danych...**)
- W polu Wybierz Typewindow źródła danych wybierz **obiektu** i kliknij przycisk **dalej**
- Wybierz obiekty danych okna dialogowego rozwijania może **WPFwithEFSample** dwa razy i wybierz pozycję **kategorii**
*Nie ma potrzeby wybrać **produkту** źródła danych, ponieważ można je uzyskać do niego za pośrednictwem **produkту** przez właściwość **kategorii** źródła danych*



- Kliknij przycisk **Zakończ**.
- Okno źródeł danych zostanie otwarty obok okna MainWindow.xaml *okna źródła danych nie są wyświetlane, wybierz opcję widok —> inne Windows -> źródło danych*
- Naciśnij ikonę pinezki, więc okna źródeł danych, automatycznie ukrywać. Może być konieczne przycisk Odśwież okno zostało już widoczne.



- Wybierz **kategorię** źródła danych i przeciągnij je na formularzu.

Następujące wystąpiły podczas możemy przeciągnąć tego źródła:

- **CategoryViewSource** zasobów i **categoryDataGrid** kontroli zostały dodane do XAML
- Właściwość DataContext nadzawanego elementu siatki "{staticresource — **categoryViewSource** }".
CategoryViewSource zasobu służy jako źródło powiązania zewnętrzne\nnadzawanego elementu siatki.
Wewnętrzny elementów siatki następnie dziedziczyć wartość DataContext nadzawanego siatki (właściwości ItemsSource categoryDataGrid jest ustawiiona na "{powiązania}")

```

<Window.Resources>
    <CollectionViewSource x:Key="categoryViewSource"
        d:DesignSource="{d:DesignInstance {x:Type local:Category}, CreateList=True}"/>
</Window.Resources>
<Grid DataContext="{StaticResource categoryViewSource}">
    <DataGrid x:Name="categoryDataGrid" AutoGenerateColumns="False" EnableRowVirtualization="True"
        ItemsSource="{Binding}" Margin="13,13,43,191"
        RowDetailsVisibilityMode="VisibleWhenSelected">
        <DataGrid.Columns>
            <DataGridTextColumn x:Name="categoryIdColumn" Binding="{Binding CategoryId}"
                Header="Category Id" Width="SizeToHeader"/>
            <DataGridTextColumn x:Name="nameColumn" Binding="{Binding Name}"
                Header="Name" Width="SizeToHeader"/>
        </DataGrid.Columns>
    </DataGrid>
</Grid>

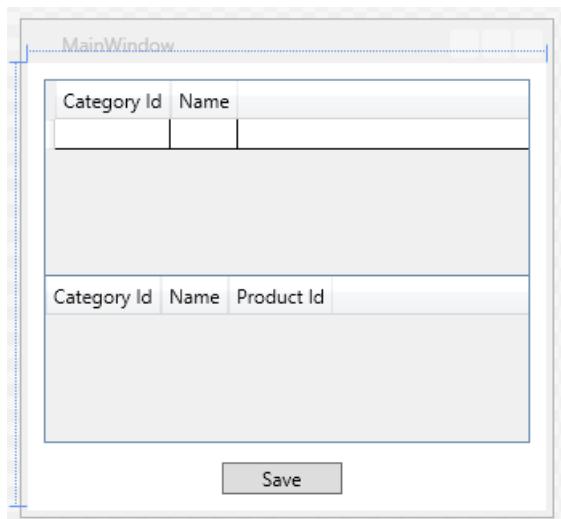
```

Dodawanie szczegółów siatki

Skoro mamy już siatki, aby wyświetlić kategorie Przyjrzyjmy dodać siatkę szczegóły, aby wyświetlić skojarzone produkty.

- Wybierz **produktów** właściwości z **kategorii** źródła danych i przeciągnij je na formularzu.
 - **CategoryProductsViewSource** zasobów i **productDataGrid** siatki są dodawane do XAML
 - Ścieżka powiązania dla tego zasobu jest równa produktów
 - Framework powiązanie danych WPF zapewnia tylko produktów związanych z wybraną kategorię wyświetlane w **productDataGrid**
- Z przybornika przeciągnij **przycisk** się do formularza. Ustaw **nazwa** właściwości **buttonSave** i **zawartości** właściwości **Zapisz**.

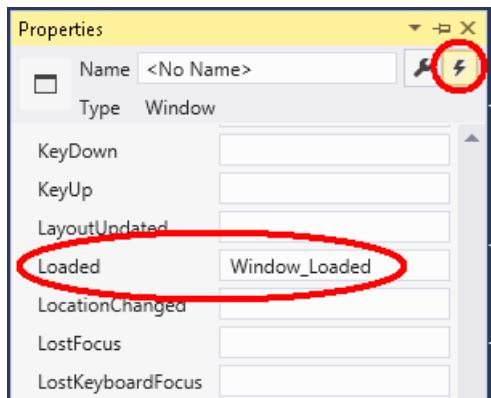
Formularz powinien wyglądać mniej więcej tak:



Dodaj kod, który obsługuje interakcji z danych

Nadszedł czas, aby dodać niektóre procedury obsługi zdarzeń do głównego okna.

- W oknie XAML kliknij <bokna> elementu, po wybraniu tej opcji okna głównego
- W **właściwości** wybierz okno **zdarzenia** w prawym górnym rogu, następnie kliknij dwukrotnie przycisk po prawej stronie pola tekstowego **Loaded** etykiety



- Również dodać **kliknij** zdarzenie **Zapisz** przycisku przez dwukrotne kliknięcie przycisku Zapisz w projektancie.

Wybranie tej opcji powoduje możesz kodu formularza, firma Microsoft będzie teraz edytować kod, aby użyć **ProductContext** przeprowadzić dostępu do danych. Zaktualizuj kod dla okna głównego, jak pokazano poniżej.

Ten kod deklaruje wystąpienie długotrwałych **ProductContext**. **ProductContext** obiekt jest używany do wykonywania zapytań i zapisać dane w bazie danych. **Dispose()** na **ProductContext** wystąpienia jest następnie wywoływana zastąpione **OnClosing** metody. Komentarze w kodzie zawierają szczegółowe informacje o dany kod realizuje.

```

using System.Data.Entity;
using System.Linq;
using System.Windows;

namespace WPFwithEFSample
{
    public partial class MainWindow : Window
    {
        private ProductContext _context = new ProductContext();
        public MainWindow()
        {
            InitializeComponent();
        }

        private void Window_Loaded(object sender, RoutedEventArgs e)
        {
            System.Windows.Data.CollectionViewSource categoryViewSource =
                ((System.Windows.Data.CollectionViewSource)(this.FindResource("categoryViewSource")));

            // Load is an extension method on IQueryable,
            // defined in the System.Data.Entity namespace.
            // This method enumerates the results of the query,
            // similar to ToList but without creating a list.
            // When used with Linq to Entities this method
            // creates entity objects and adds them to the context.
            _context.Categories.Load();

            // After the data is loaded call the DbSet<T>.Local property
            // to use the DbSet<T> as a binding source.
            categoryViewSource.Source = _context.Categories.Local;
        }

        private void buttonSave_Click(object sender, RoutedEventArgs e)
        {
            // When you delete an object from the related entities collection
            // (in this case Products), the Entity Framework doesn't mark
            // these child entities as deleted.
            // Instead, it removes the relationship between the parent and the child
            // by setting the parent reference to null.
            // So we manually have to delete the products
            // that have a Category reference set to null.
        }
    }
}

```

```

// The following code uses LINQ to Objects
// against the Local collection of Products.
// The ToList call is required because otherwise the collection will be modified
// by the Remove call while it is being enumerated.
// In most other situations you can use LINQ to Objects directly
// against the Local property without using ToList first.
foreach (var product in _context.Products.Local.ToList())
{
    if (product.Category == null)
    {
        _context.Products.Remove(product);
    }
}

_context.SaveChanges();
// Refresh the grids so the database generated values show up.
this.categoryDataGrid.Items.Refresh();
this.productsDataGrid.Items.Refresh();
}

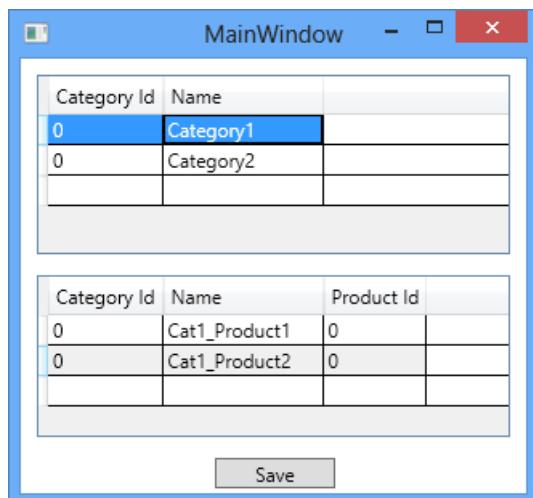
protected override void OnClosing(System.ComponentModel.CancelEventArgs e)
{
    base.OnClosing(e);
    this._context.Dispose();
}
}

}

```

Testowanie aplikacji WPF

- Skompilować i uruchomić aplikację. Jeśli użyto Code First, a następnie zostanie wyświetlony **WPFWithEFSample.ProductContext** baza danych została utworzona dla Ciebie.
- Wprowadź nazwę kategorii w górnym siatki produktu nazwy i w dół siatki *wypełniać kolumny Identyfikatora, ponieważ klucz podstawowy jest generowany przez bazę danych*



- Naciśnij klawisz **Zapisz** przycisk, aby zapisać dane w bazie danych

Po wywołaniu przez DbContext **SaveChanges()**, identyfikatory są wypełniane przy użyciu wartości z bazy danych, wygenerowane. Ponieważ firma Microsoft o nazwie **Refresh()** po **SaveChanges()** DataGrid formanty są aktualizowane na nowe wartości.

The screenshot shows a Windows application window titled "MainWindow". Inside the window, there are two data grids and a "Save" button.

The first data grid has columns "Category Id" and "Name". It contains three rows:

Category Id	Name
3	Category1
4	Category2

The second data grid has columns "Category Id", "Name", and "Product Id". It contains three rows:

Category Id	Name	Product Id
4	Cat2_Product1	5
4	Cat2_Product2	6

Below the second data grid is a "Save" button.

Dodatkowe zasoby

Aby dowiedzieć się, jak wiązanie danych do kolekcji przy użyciu programu WPF, zobacz [w tym temacie](#) w dokumentacji programu WPF.

Praca z odłączone jednostki

25.10.2018 • 3 minutes to read • [Edit Online](#)

W aplikacji Entity Framework, na podstawie klasy kontekstu jest odpowiedzialny za wykrywanie zmian zastosowanych do śledzonych jednostek. Wywołanie metody `SaveChanges` utrzymuje zmian śledzonych przez kontekst do bazy danych. Podczas pracy z aplikacjami n warstwowej, obiekty jednostki są zazwyczaj modyfikowane przy braku połączenia z kontekstem i należy zdecydować, jak śledzić zmiany i raportu te zmiany do kontekstu. W tym temacie omówiono różne opcje, które są dostępne, gdy używający narzędzia Entity Framework o odłączony jednostek.

Struktury usługi sieci Web

Technologie usług sieci Web zwykle obsługują wzorce, które mogą służyć do utrwalenia zmian na poszczególne obiekty odłączonych. Na przykład ASP.NET Web API umożliwia akcji kontrolera kodu, które może obejmować wywołania do EF, aby zachować zmiany wprowadzone do obiektu w bazie danych. W rzeczywistości internetowy interfejs API, narzędzi w programie Visual Studio ułatwia tworzenia szkieletu kontrolera interfejsu API sieci Web z modelem Entity Framework 6. Aby uzyskać więcej informacji, zobacz [przy użyciu interfejsu API sieci Web za pomocą platformy Entity Framework 6](#).

W przeszłości były kilka innych technologii sieci Web usługi, które oferowane integracji z programem Entity Framework, takie jak [usług danych WCF](#) i [RIA Services](#).

Interfejsy API niskiego poziomu EF

Jeśli nie chcesz użyć istniejącego rozwiązania n warstwowa lub jeśli chcesz dostosować, co się dzieje w akcji kontrolera, w usługach interfejsu API sieci Web, platformy Entity Framework udostępnia interfejsy API, które umożliwiają zastosowanie zmian wprowadzonych w warstwie odłączonych. Aby uzyskać więcej informacji, zobacz [stanu Dodaj, Dołącz i jednostki](#).

Samodzielnie śledzenie jednostek

Śledzenie zmian na wykresach dowolnego jednostek przy braku połączenia z kontekstem EF jest twardych problem. Jeden z próbuje rozwiązać problem był szablonu generacji kodu Self-Tracking jednostek. Ten szablon generuje klasy jednostek, które zawierają logikę do śledzenia zmian wprowadzanych w warstwie odłączonego jako stan w samych jednostkach. Zestaw metod rozszerzenia zostanie również wygenerowany tak, aby zastosować te zmiany do kontekstu.

Ten szablon może być używany z modeli utworzonych za pomocą projektanta EF, ale nie można używać w modelach Code First. Aby uzyskać więcej informacji, zobacz [jednostek Self-Tracking](#).

IMPORTANT

Nie zaleca się przy użyciu szablonu samoobsługowego tracking jednostek. Tylko będą dostępne do obsługi istniejących aplikacji. Jeśli aplikacja wymaga pracy z wykresami odłączonych jednostek, należy wziąć pod uwagę inne alternatywy dla takich jak [słupkowych jednostek](#), która jest podobna do samoobsługowego-Tracking-jednostek, które jest bardziej aktywnie rozwijany przez technologię Społeczność lub pisanie kodu niestandardowego za pomocą śledzenia interfejsów API zmian niskiego poziomu.

Samodzielnie śledzenia jednostek

27.09.2018 • 9 minutes to read • [Edit Online](#)

IMPORTANT

Nie zaleca się przy użyciu szablonu samoobsługowego tracking jednostek. Tylko będą dostępne do obsługi istniejących aplikacji. Jeśli aplikacja wymaga pracy z wykresami odłączonych jednostek, należy wziąć pod uwagę inne alternatywy dla takich jak [słupkowych jednostek](#), która jest podobna do samoobsługowego-Tracking-jednostek, które jest bardziej aktywnie rozwijany przez technologię Społeczność lub pisanie kodu niestandardowego za pomocą śledzenia interfejsów API zmian niskiego poziomu.

W aplikacji Entity Framework, na podstawie kontekstu jest odpowiedzialny za śledzenie zmian w obiekty. Możesz następnie użyć metody `SaveChanges` aby utrważyć zmiany w bazie danych. Podczas pracy z aplikacjami N-warstwowej, obiekty jednostki zwykle jest odłączony od kontekstu i należy zdecydować, jak śledzić zmiany i raportu te zmiany do kontekstu. Samodzielnie śledzenia jednostek (`ste`) może pomóc śledzić zmiany w dowolnej warstwy, a następnie odtworzenia te zmiany do kontekstu do zapisania.

Użyj `ste` tylko wtedy, gdy kontekst nie jest dostępna w ramach warstwy gdzie zostały wprowadzone zmiany do obiektu wykresu. Jeśli kontekst jest dostępny, nie ma potrzeby używania `ste`, ponieważ zajmie się kontekst śledzenia zmian.

Ten element szablon generuje dwa `.TT` — pliki (szablon tekstowy):

- **<Nazwę modelu>.tt** generuje plik typów jednostek i klasa pomocnika, która zawiera logikę śledzenia zmian, która jest używana przez własny śledzenia jednostek i metody rozszerzenia, które umożliwiają ustawianie stanu na własnym śledzenie jednostki.
- **<Nazwę modelu>. Context.TT** generuje plik pochodnej kontekstu i klasa rozszerzenia, która zawiera `applychanges` — metody **ObjectContext** i **obiektu ObjectSet** klasy. Te metody zbadania informacji śledzenia zmian, które znajduje się na wykresie własnym śledzenia jednostek wywnioskowania zestaw operacji, które należy wykonać, aby zapisać zmiany w bazie danych.

Rozpocznij

Aby rozpocząć pracę, odwiedź stronę [Self-Tracking jednostek wskazówki](#) strony.

Uwagi dotyczące funkcjonalności podczas pracy z własnym śledzenia jednostek

IMPORTANT

Nie zaleca się przy użyciu szablonu samoobsługowego tracking jednostek. Tylko będą dostępne do obsługi istniejących aplikacji. Jeśli aplikacja wymaga pracy z wykresami odłączonych jednostek, należy wziąć pod uwagę inne alternatywy dla takich jak [słupkowych jednostek](#), która jest podobna do samoobsługowego-Tracking-jednostek, które jest bardziej aktywnie rozwijany przez technologię Społeczność lub pisanie kodu niestandardowego za pomocą śledzenia interfejsów API zmian niskiego poziomu.

Podczas pracy z własnym śledzenie jednostek, należy wziąć pod uwagę następujące czynności:

- Upewnij się, że projekt klienta ma odwołanie do zestawu zawierającego typów jednostek. Jeśli dodasz tylko odwołanie do usługi do projektu klienta, projekt klienta użyje typy serwerów proxy usługi WCF i nie

rzeczywiste własnym śledzenie typów jednostek. Oznacza to, że nie będzie można uzyskać funkcji automatyczne powiadomienie o zarządzających śledzenia jednostki na kliencie. Jeśli nie chcesz jej celowo i obejmuje dodatkowe typy jednostek, należy ręcznie ustawić informacji śledzenia zmian na klienta, aby zmiany wysyłane z powrotem do usługi.

- Wywołania operacji usługi powinny być bezstanowe i Utwórz nowe wystąpienie obiektu kontekstu. Zalecamy również utworzenie obiektu kontekstu w **przy użyciu** bloku.
- Kiedy wykres, który został zmodyfikowany na kliencie z usługą i wysyła następnie zamierzasz kontynuować pracę z tym samym wykresie na komputerze klienckim, należy ręcznie wykonać iterację wykresu i wywołania **AcceptChanges** metody dla każdego obiektu do Zresetuj śledzenie zmian.

Jeśli obiekty w grafie zawierają właściwości wartościami wygenerowanych w bazie danych (na przykład wartości tożsamości lub współbieżności), platformy Entity Framework spowoduje zastąpienie wartości tych właściwości z wartościami bazy danych, wygenerowane po **SaveChanges** metoda jest wywoływana. Możesz zaimplementować operację usługi do zwrócenia zapisanych obiektów lub Podaj listę wartości wygenerowanej właściwości dla obiektów do klienta. Klient musiałby Zamień wystąpienie obiektów lub wartości właściwości obiektu do obiektów lub wartości właściwości zwrócony przez operację usługi.

- Scalanie wykresów z wielu żądań usługi może stanowić obiekty ze zduplikowanymi wartościami klucza w wynikowym wykresie. Entity Framework nie powoduje usunięcia obiektów z zduplikowane klucze po wywołaniu **applychanges** — metody, ale zamiast tego zgłasza wyjątek. Aby uniknąć wykresy ze zduplikowanymi wartościami klucza, postępuj zgodnie z jednym z wzorców opisanego w następujący wpis w blogu: [jednostek Self-Tracking: applychanges — i zduplikowanych podmiotów](#).
- Po zmianie relacji między obiektami, ustawiając właściwość klucza obcego referencyjna właściwość nawigacji jest ustawiona na wartość null i nie są zsynchronizowane do odpowiedniej jednostki podmiotu zabezpieczeń na komputerze klienckim. Po dołączeniu do kontekstu obiektów programu graph (na przykład, po wywołaniu metody **applychanges** — metoda), właściwości klucza obcego i właściwości nawigacji są synchronizowane.

Nie ma właściwości nawigacji odwołania synchronizowane z odpowiedniego obiektu podmiotu zabezpieczeń może być problem, jeśli określono usuwanie kaskadowe w relacji klucza obcego. Jeśli usuniesz główny, Usuń nie będą przekazywane do obiektów zależnych. Jeśli masz kaskadowo określony, należy użyć właściwości nawigacji, aby zmienić relacji zamiast ustawiać właściwości klucza obcego.

- Samodzielnie śledzenie jednostek nie są włączone do wykonywania ładowania z opóźnieniem.
- Serializacja binarna i serializacji obiektów zarządzania stan programu ASP.NET nie jest obsługiwane przez własny śledzenie jednostek. Można jednak dostosować szablon, aby dodać obsługę serializacji binarnej. Aby uzyskać więcej informacji, zobacz [za pomocą serializacji binarnej i ViewState z jednostkami Self-Tracking](#).

Zagadnienia dotyczące zabezpieczeń

Następujące zagadnienia dotyczące zabezpieczeń powinny brane pod uwagę podczas pracy z własnym śledzenie jednostki:

- Usługa nie należy ufać żądań kierowanych do pobrania lub aktualizowanie danych w kliencie niezaufanej lub za pośrednictwem niezaufanej kanału. Klient musi zostać uwierzytelny: bezpieczne koperty kanału lub komunikat powinien być używany. Aby upewnić się, że są one zgodne z oczekiwaniami i jest uzasadnione zmiany dla danego scenariusza, można sprawdzić poprawności żądania klientów do aktualizacji lub odbierać

dane.

- Należy unikać poufnych informacji jako klucze jednostek (na przykład numery ubezpieczenia społecznego). Zmniejsza to możliwość przypadkowo szeregowania informacje poufne na własnym śledzenie wykresy jednostki do klienta, który nie jest w pełni zaufany. Za pomocą skojarzeń niezależnie od oryginalnego klucza podmiotu, który jest powiązany z tą, która jest deserializowana mogą zostać wysłane do klienta, jak również.
- Aby uniknąć propagowanie komunikaty o wyjątkach, zawierających poufne dane w warstwie klienta wywołania **applychanges** — i **SaveChanges** na serwerze warstwy musi być ujęte w kodzie obsługi wyjątków.

Śledzenie własnym wskazówkami jednostek

13.09.2018 • 20 minutes to read • [Edit Online](#)

IMPORTANT

Nie zaleca się przy użyciu szablonu samoobsługowego tracking jednostek. Tylko będą dostępne do obsługi istniejących aplikacji. Jeśli aplikacja wymaga pracy z wykresami odłączonych jednostek, należy wziąć pod uwagę inne alternatywy dla takich jak [słupkowych jednostek](#), która jest podobna do samoobsługowego-Tracking-jednostek, które jest bardziej aktywnie rozwijany przez technologię Społeczność lub pisanie kodu niestandardowego za pomocą śledzenia interfejsów API zmian niskiego poziomu.

W tym instruktażu przedstawiono scenariusz, w którym usługi Windows Communication Foundation (WCF) udostępniają operację, która zwraca grafikę jednostki. Następnie aplikacja kliencka manipuluje tą grafiką i przesyła zmiany do operacji usługi, która weryfikuje i zapisuje aktualizacji do bazy danych przy użyciu platformy Entity Framework.

Przed wykonaniem tego przewodnika, upewnij się, możesz przeczytać [jednostek Self-Tracking](#) strony.

W tym przewodniku wykona następujące czynności:

- Tworzy bazę danych w celu uzyskania dostępu do.
- Tworzy bibliotekę klas, który zawiera model.
- Zamiany w szablonie Self-Tracking jednostki Generator.
- Przenosi klas jednostek do oddzielnego projektu.
- Tworzy usługę WCF, która udostępnia operacje do wykonywania zapytań i zapisywania jednostki.
- Tworzy klienta aplikacji (konsoli i WPF), które mogą korzystać z usługi.

Użyjemy pierwszej bazy danych w ramach tego przewodnika, ale te same techniki stosuje się jednakowo do pierwszego modelu.

Wymagania wstępne

Do przeprowadzenia tego instruktażu potrzebujesz najnowszej wersji programu Visual Studio.

Tworzenie bazy danych

Serwer bazy danych, który został zainstalowany przy użyciu programu Visual Studio różni się zależnie od wersji programu Visual Studio zostały zainstalowane:

- Jeśli używasz programu Visual Studio 2012, a następnie będzie utworzenie bazy danych LocalDB.
- Jeśli używasz programu Visual Studio 2010 zostanie utworzona baza danych programu SQL Express.

Rozpoczniemy i wygenerować bazę danych.

- Otwórz program Visual Studio
- **Widok —> Eksploratora serwera**
- Kliknij prawym przyciskiem myszy **połączeń danych** -> **Dodaj połączenie...**
- Jeśli nie jest połączona z bazą danych za pomocą Eksploratora serwera, zanim będzie konieczne wybranie **programu Microsoft SQL Server** jako źródło danych
- Łączenie się z LocalDB lub SQL Express, w zależności od tego, który z nich został zainstalowany

- Wprowadź **STESample** jako nazwa bazy danych
- Wybierz **OK** i uzyskasz, jeśli chcesz utworzyć nową bazę danych, wybierz opcję **tak**
- Nowa baza danych będzie teraz wyświetlany w Eksploratorze serwera
- Jeśli używasz programu Visual Studio 2012
 - Kliknij prawym przyciskiem myszy w bazie danych w Eksploratorze serwera i wybierz **nowe zapytanie**
 - Skopiuj następujące instrukcje SQL do nowego zapytania, a następnie kliknij prawym przyciskiem myszy, zapytania i wybierz pozycję **wykonania**
- Jeśli używasz programu Visual Studio 2010
 - Wybierz **dane —> języka Transact SQL edytor —> nowego połączenia zapytania...**
 - Wprowadź **.\ jako nazwę serwera i kliknij przycisk **OK****
 - Wybierz **STESample** bazy danych z listy rozwijanej w górnej części edytora zapytań
 - Skopiuj następujące instrukcje SQL do nowego zapytania, a następnie kliknij prawym przyciskiem myszy, zapytania i wybierz pozycję **wykonaj instrukcję SQL**

```

CREATE TABLE [dbo].[Blogs] (
    [BlogId] INT IDENTITY (1, 1) NOT NULL,
    [Name] NVARCHAR (200) NULL,
    [Url] NVARCHAR (200) NULL,
    CONSTRAINT [PK_dbo.Blogs] PRIMARY KEY CLUSTERED ([BlogId] ASC)
);

CREATE TABLE [dbo].[Posts] (
    [PostId] INT IDENTITY (1, 1) NOT NULL,
    [Title] NVARCHAR (200) NULL,
    [Content] NTEXT NULL,
    [BlogId] INT NOT NULL,
    CONSTRAINT [PK_dbo.Posts] PRIMARY KEY CLUSTERED ([PostId] ASC),
    CONSTRAINT [FK_dbo.Posts_dbo.Blogs_BlogId] FOREIGN KEY ([BlogId]) REFERENCES [dbo].[Blogs] ([BlogId])
ON DELETE CASCADE
);

SET IDENTITY_INSERT [dbo].[Blogs] ON
INSERT INTO [dbo].[Blogs] ([BlogId], [Name], [Url]) VALUES (1, N'ADO.NET Blog', N'blogs.msdn.com/adonet')
SET IDENTITY_INSERT [dbo].[Blogs] OFF
INSERT INTO [dbo].[Posts] ([Title], [Content], [BlogId]) VALUES (N'Intro to EF', N'Interesting stuff...', 1)
INSERT INTO [dbo].[Posts] ([Title], [Content], [BlogId]) VALUES (N'What is New', N'More interesting stuff...', 1)

```

Tworzenie modelu

Up, potrzebujemy projektu, aby przełączyć modelu.

- **Plik —> New -> projektu...**
- Wybierz **Visual C#** z okienka po lewej stronie i następnie **biblioteki klas**
- Wprowadź **STESample** jako nazwę i kliknij przycisk **OK**

Teraz utworzymy prosty model w Projektancie platformy EF dostępu do naszej bazy danych:

- **Projekt —> Dodaj nowy element...**
- Wybierz **danych** z okienka po lewej stronie i następnie **ADO.NET Entity Data Model**
- Wprowadź **BloggingModel** jako nazwę i kliknij przycisk **OK**
- Wybierz **Generuj z bazy danych** i kliknij przycisk **dalej**
- Wprowadź informacje o połączeniu dla bazy danych, który został utworzony w poprzedniej sekcji
- Wprowadź **BloggingContext** jako nazwy parametrów połączenia, a następnie kliknij przycisk **dalej**
- Zaznacz pole wyboru obok pozycji **tabel** i kliknij przycisk **Zakończ**

Przechodź do generowania kodu WKLEJ

Teraz należy wyłączyć domyślne generowanie kodu i wymiany Self-Tracking jednostek.

Jeśli używasz programu Visual Studio 2012

- Rozwiń **BloggingModel.edmx** w **Eksploratora rozwiązań** i Usuń **BloggingModel.tt** i **BloggingModel.Context.tt** Spowoduje to wyłączenie generowania kodu domyślnego
- Kliknij prawym przyciskiem myszy pusty obszar w Projektancie platformy EF powierzchni i wybierz **Dodaj element generowanie kodu...**
- Wybierz **Online** w okienku po lewej stronie i wyszukaj **Generator WKLEJ**
- Wybierz **Generator Wklej kod c#** szablonu, wprowadź **STETemplate** jako nazwę i kliknij przycisk **Dodaj**
- **STETemplate.tt** i **STETemplate.Context.tt** pliki zostaną dodane zagnieźdzony w ramach pliku **BloggingModel.edmx**

Jeśli używasz programu Visual Studio 2010

- Kliknij prawym przyciskiem myszy pusty obszar w Projektancie platformy EF powierzchni i wybierz **Dodaj element generowanie kodu...**
- Wybierz **kodu** z okienka po lewej stronie i następnie **Generator jednostki Self-Tracking ADO.NET**
- Wprowadź **STETemplate** jako nazwę i kliknij przycisk **Dodaj**
- **STETemplate.tt** i **STETemplate.Context.tt** pliki zostaną dodane bezpośrednio do projektu

Przenieś typów jednostek do oddzielnego projektu

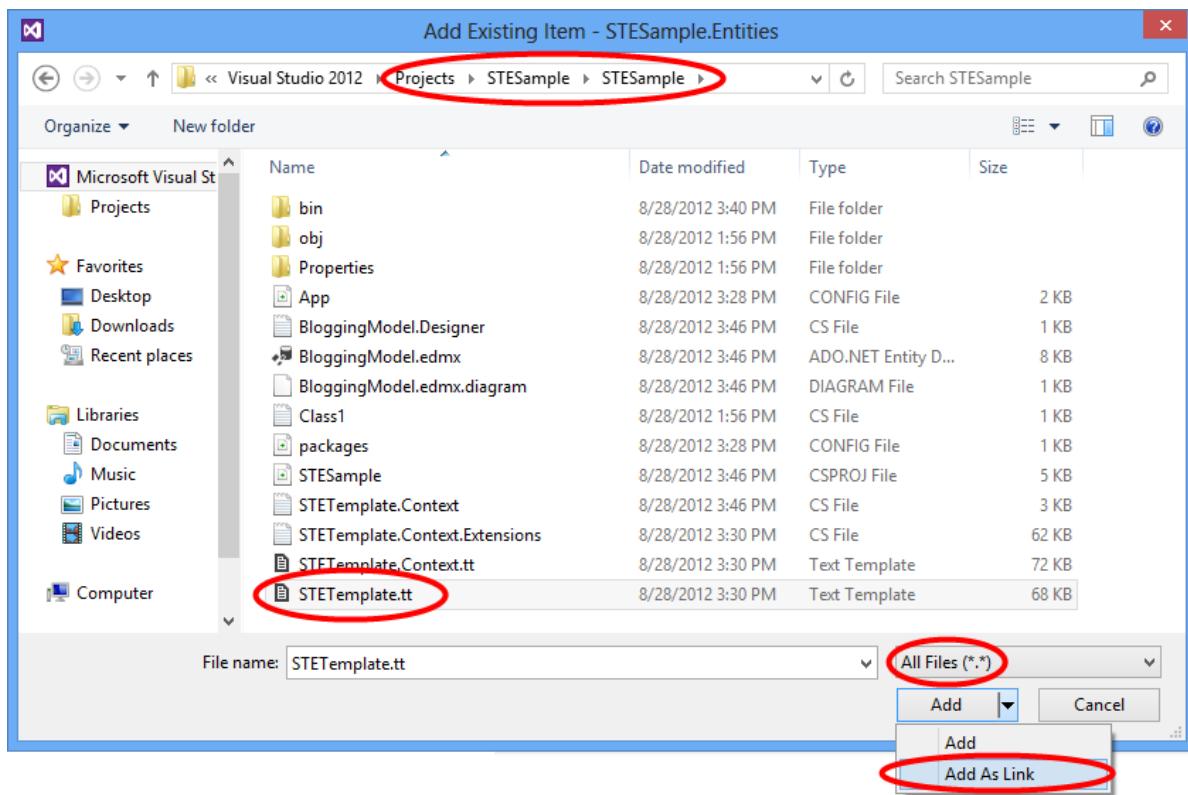
Aby korzystanie z jednostek Self-Tracking naszej aplikacji klienta musi mieć dostęp do klas jednostek wygenerowana przez nasz model. Ponieważ nie chcemy ujawniać całego modelu do aplikacji klienckiej będziemy przenosić klas jednostek do oddzielnego projektu.

Pierwszym krokiem jest zatrzymać generowanie klas jednostek w istniejący projekt:

- Kliknij prawym przyciskiem myszy **STETemplate.tt** w **Eksploratora rozwiązań** i wybierz **właściwości**
- W **właściwości** Wyczyść okno **TextTemplatingFileGenerator** z **CustomTool** właściwości
- Rozwiń **STETemplate.tt** w **Eksploratora rozwiązań** i usunąć wszystkie pliki zagnieźdzony w nim

Następnie użyjemy Dodaj nowy projekt i wygenerowania klas obiektów w nim

- **Plik —> Add -> projektu...**
- Wybierz **Visual C#** z okienka po lewej stronie i następnie **biblioteki klas**
- Wprowadź **STESample.Entities** jako nazwę i kliknij przycisk **OK**
- **Projekt —> Dodaj istniejący element...**
- Przejdź do **STESample** folder projektu
- Zaznacz, aby wyświetlić **wszystkie pliki (*.*)**
- Wybierz **STETemplate.tt** pliku
- Kliknij strzałkę listy rozwijanej obok **Dodaj** i wybrać **Dodaj jako łącze**



Przedstawimy również upewnić się, że wygenerowanie klas obiektów w tej samej przestrzeni nazw jako kontekst. Zmniejsza to po prostu liczbę za pomocą instrukcji, które musimy dodać w naszej aplikacji.

- Kliknij prawym przyciskiem myszy na połączonym **STETemplate.tt** w **Eksploratora rozwiązań** i wybierz **właściwości**
- W **właściwości** zestaw okna **niestandardowe narzędzie Namespace** do **STESample**

Kod wygenerowany przez szablon WKLEJ należy odwołanie do **System.Runtime.Serialization** w celu komplikacji. Ta biblioteka jest wymagany w przypadku WCF **DataContract** i **DataMember** atrybuty, które są używane dla typów możliwych do serializacji jednostek.

- Kliknij prawym przyciskiem myszy **STESample.Entities** projektu w **Eksploratora rozwiązań** i wybierz **Dodaj odwołanie...**
 - W programie Visual Studio 2012 — zaznacz pole wyboru obok pozycji **System.Runtime.Serialization** i kliknij przycisk **OK**
 - W programie Visual Studio 2010 — wybierz **System.Runtime.Serialization** i kliknij przycisk **OK**

Na koniec projektu z nasz kontekst w nim należy odwołania do typów jednostek.

- Kliknij prawym przyciskiem myszy **STESample** projektu w **Eksploratora rozwiązań** i wybierz **Dodaj odwołanie...**
 - W programie Visual Studio 2012 — wybierz **rozwiązania** z okienka po lewej stronie, zaznacz pole wyboru obok pozycji **STESample.Entities** i kliknij przycisk **OK**
 - W programie Visual Studio 2010 — wybierz **projektów** zaznacz **STESample.Entities** i kliknij przycisk **OK**

NOTE

Inną opcją przechodzenia typów jednostek do oddzielnego projektu jest można przenieść pliku szablonu, a nie połączenie go z jego domyślnej lokalizacji. Jeśli to zrobisz, musisz zaktualizować **wejściowy** zmiennej szablonu, aby podać ścieżkę względną do pliku edmx (w tym przykładzie, która byłaby ... \BloggingModel.edmx).

Tworzenie usługi WCF

Teraz nadszedł czas, aby dodać usługi WCF w celu udostępnienia danych. Zaczniemy od utworzenia projektu.

- Plik —> Add -> projektu...
- Wybierz **Visual C#** z okienka po lewej stronie i następnie **aplikacja usługi WCF**
- Wprowadź **STESample.Service** jako nazwę i kliknij przycisk **OK**
- Dodaj odwołanie do **System.Data.Entity** zestawu
- Dodaj odwołanie do **STESample** i **STESample.Entities** projektów

Potrzebujemy skopiować parametry połączenia platformy EF do tego projektu, dzięki czemu znajduje się w czasie wykonywania.

- Otwórz **App.Config** plik ** STESample ** projektu i skopiuj **connectionStrings** — element
- Wklej **connectionStrings** element jako element podzialek **konfiguracji** elementu **Web.Config** w pliku **STESample.Service** projektu

Teraz nadszedł czas, aby zaimplementować rzeczywistej usługi.

- Otwórz **IService1.cs** i zastąp jego zawartość następującym kodem

```
using System.Collections.Generic;
using System.ServiceModel;

namespace STESample.Service
{
    [ServiceContract]
    public interface IService1
    {
        [OperationContract]
        List<Blog> GetBlogs();

        [OperationContract]
        void UpdateBlog(Blog blog);
    }
}
```

- Otwórz **plik Service1.svc** i zastąp jego zawartość następującym kodem

```

using System;
using System.Collections.Generic;
using System.Data;
using System.Linq;

namespace STESample.Service
{
    public class Service1 : IService1
    {
        /// <summary>
        /// Gets all the Blogs and related Posts.
        /// </summary>
        public List<Blog> GetBlogs()
        {
            using (BloggingContext context = new BloggingContext())
            {
                return context.Blogs.Include("Posts").ToList();
            }
        }

        /// <summary>
        /// Updates Blog and its related Posts.
        /// </summary>
        public void UpdateBlog(Blog blog)
        {
            using (BloggingContext context = new BloggingContext())
            {
                try
                {
                    // TODO: Perform validation on the updated order before applying the changes.

                    // The ApplyChanges method examines the change tracking information
                    // contained in the graph of self-tracking entities to infer the set of operations
                    // that need to be performed to reflect the changes in the database.
                    context.Blogs.ApplyChanges(blog);
                    context.SaveChanges();

                }
                catch (UpdateException)
                {
                    // To avoid propagating exception messages that contain sensitive data to the client
                    tier
                    // calls to ApplyChanges and SaveChanges should be wrapped in exception handling code.
                    throw new InvalidOperationException("Failed to update. Try your request again.");
                }
            }
        }
    }
}

```

Korzystanie z usługi z aplikacji konsoli

Utwórz aplikację konsolową, która korzysta z naszych usług.

- **Plik —> New -> projektu...**
- Wybierz **Visual C#** z okienka po lewej stronie i następnie **aplikacji konsoli**
- Wprowadź **STESample.ConsoleTest** jako nazwę i kliknij przycisk **OK**
- Dodaj odwołanie do **STESample.Entities** projektu

Potrzebujemy odwołanie do usługi dla usługi WCF

- Kliknij prawym przyciskiem myszy **STESample.ConsoleTest** projektu w **Eksploratora rozwiązań** i wybierz **Dodaj odwołanie do usługi...**

- Kliknij przycisk **odnajdywania**
- Wprowadź **BloggingService** jako przestrzeń nazw i kliknij przycisk **OK**

Obecnie firma Microsoft może pisanie kodu w celu korzystania z usługi.

- Otwórz **Program.cs** i zastąp jego zawartość następującym kodem.

```

using STESample.ConsoleTest.BloggingService;
using System;
using System.Linq;

namespace STESample.ConsoleTest
{
    class Program
    {
        static void Main(string[] args)
        {
            // Print out the data before we change anything
            Console.WriteLine("Initial Data:");
            DisplayBlogsAndPosts();

            // Add a new Blog and some Posts
            AddBlogAndPost();
            Console.WriteLine("After Adding:");
            DisplayBlogsAndPosts();

            // Modify the Blog and one of its Posts
            UpdateBlogAndPost();
            Console.WriteLine("After Update:");
            DisplayBlogsAndPosts();

            // Delete the Blog and its Posts
            DeleteBlogAndPost();
            Console.WriteLine("After Delete:");
            DisplayBlogsAndPosts();

            Console.WriteLine("Press any key to exit...");
            Console.ReadKey();
        }

        static void DisplayBlogsAndPosts()
        {
            using (var service = new Service1Client())
            {
                // Get all Blogs (and Posts) from the service
                // and print them to the console
                var blogs = service.GetBlogs();
                foreach (var blog in blogs)
                {
                    Console.WriteLine(blog.Name);
                    foreach (var post in blog.Posts)
                    {
                        Console.WriteLine(" - {0}", post.Title);
                    }
                }
            }

            Console.WriteLine();
            Console.WriteLine();
        }

        static void AddBlogAndPost()
        {
            using (var service = new Service1Client())
            {
                // Create a new Blog with a couple of Posts
                var newBlog = new Blog

```

```

    {
        Name = "The New Blog",
        Posts =
        {
            new Post { Title = "Welcome to the new blog"},
            new Post { Title = "What's new on the new blog"}
        }
    };

    // Save the changes using the service
    service.UpdateBlog(newBlog);
}

static void UpdateBlogAndPost()
{
    using (var service = new Service1Client())
    {
        // Get all the Blogs
        var blogs = service.GetBlogs();

        // Use LINQ to Objects to find The New Blog
        var blog = blogs.First(b => b.Name == "The New Blog");

        // Update the Blogs name
        blog.Name = "The Not-So-New Blog";

        // Update one of the related posts
        blog.Posts.First().Content = "Some interesting content...";

        // Save the changes using the service
        service.UpdateBlog(blog);
    }
}

static void DeleteBlogAndPost()
{
    using (var service = new Service1Client())
    {
        // Get all the Blogs
        var blogs = service.GetBlogs();

        // Use LINQ to Objects to find The Not-So-New Blog
        var blog = blogs.First(b => b.Name == "The Not-So-New Blog");

        // Mark all related Posts for deletion
        // We need to call ToList because each Post will be removed from the
        // Posts collection when we call MarkAsDeleted
        foreach (var post in blog.Posts.ToList())
        {
            post.MarkAsDeleted();
        }

        // Mark the Blog for deletion
        blog.MarkAsDeleted();

        // Save the changes using the service
        service.UpdateBlog(blog);
    }
}
}

```

Teraz można uruchomić aplikacji, aby zobaczyć go w działaniu.

- Kliknij prawym przyciskiem myszy **STESample.ConsoleTest** projektu w **Eksploratora rozwiązań** i wybierz **debugowanie -> Uruchom nowe wystąpienie**

Zostaną wyświetcone następujące dane wyjściowe, gdy aplikacja wykonuje.

```
Initial Data:  
ADO.NET Blog  
- Intro to EF  
- What is New  
  
After Adding:  
ADO.NET Blog  
- Intro to EF  
- What is New  
The New Blog  
- Welcome to the new blog  
- What's new on the new blog  
  
After Update:  
ADO.NET Blog  
- Intro to EF  
- What is New  
The Not-So-New Blog  
- Welcome to the new blog  
- What's new on the new blog  
  
After Delete:  
ADO.NET Blog  
- Intro to EF  
- What is New  
  
Press any key to exit...
```

Korzystanie z usługi z aplikacji WPF

Utworzymy aplikację WPF, która korzysta z naszych usług.

- **Plik —> New -> projektu...**
- Wybierz **Visual C#** z okienka po lewej stronie i następnie **aplikacji WPF**
- Wprowadź **STESample.WPFTest** jako nazwę i kliknij przycisk **OK**
- Dodaj odwołanie do **STESample.Entities** projektu

Potrzebujemy odwołanie do usługi dla usługi WCF

- Kliknij prawym przyciskiem myszy **STESample.WPFTest** projektu w **Eksploratora rozwiązań** i wybierz **Dodaj odwołanie do usługi...**
- Kliknij przycisk **odnajdywania**
- Wprowadź **BloggingService** jako przestrzeń nazw i kliknij przycisk **OK**

Obecnie firma Microsoft może pisanie kodu w celu korzystania z usługi.

- Otwórz **MainWindow.xaml** i zastąp jego zawartość następującym kodem.

```

<Window
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    xmlns:STESample="clr-namespace:STESample;assembly=STESample.Entities"
    mc:Ignorable="d" x:Class="STESample.WPFTest.MainWindow"
    Title="MainWindow" Height="350" Width="525" Loaded="Window_Loaded">

    <Window.Resources>
        <CollectionViewSource
            x:Key="blogViewSource"
            d:DesignSource="{d:DesignInstance {x:Type STESample:Blog}, CreateList=True}"/>
        <CollectionViewSource
            x:Key="blogPostsViewSource"
            Source="{Binding Posts, Source={StaticResource blogViewSource}}"/>
    </Window.Resources>

    <Grid DataContext="{StaticResource blogViewSource}">
        <DataGrid AutoGenerateColumns="False" EnableRowVirtualization="True"
                  ItemsSource="{Binding}" Margin="10,10,10,179">
            <DataGrid.Columns>
                <DataGridTextColumn Binding="{Binding BlogId}" Header="Id" Width="Auto" IsReadOnly="True" />
                <DataGridTextColumn Binding="{Binding Name}" Header="Name" Width="Auto"/>
                <DataGridTextColumn Binding="{Binding Url}" Header="Url" Width="Auto"/>
            </DataGrid.Columns>
        </DataGrid>
        <DataGrid AutoGenerateColumns="False" EnableRowVirtualization="True"
                  ItemsSource="{Binding Source={StaticResource blogPostsViewSource}}"
                  Margin="10,145,10,38">
            <DataGrid.Columns>
                <DataGridTextColumn Binding="{Binding PostId}" Header="Id" Width="Auto" IsReadOnly="True"/>
                <DataGridTextColumn Binding="{Binding Title}" Header="Title" Width="Auto"/>
                <DataGridTextColumn Binding="{Binding Content}" Header="Content" Width="Auto"/>
            </DataGrid.Columns>
        </DataGrid>
        <Button Width="68" Height="23" HorizontalAlignment="Right" VerticalAlignment="Bottom"
               Margin="0,0,10,10" Click="buttonSave_Click">Save</Button>
    </Grid>
</Window>

```

- Otwórz kod związanego z dla głównego (**MainWindow.xaml.cs**) i zastąp jego zawartość następującym kodem

```

using STESample.WPFTest.BloggingService;
using System.Collections.Generic;
using System.Linq;
using System.Windows;
using System.Windows.Data;

namespace STESample.WPFTest
{
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
        }

        private void Window_Loaded(object sender, RoutedEventArgs e)
        {
            using (var service = new Service1Client())
            {
                // Find the view source for Blogs and populate it with all Blogs (and related Posts)
                // from the Service. The default editing functionality of WPF will allow the objects
                // to be manipulated on the screen.
                var blogsViewSource = (CollectionViewSource)this.FindResource("blogViewSource");
                blogsViewSource.Source = service.GetBlogs().ToList();
            }
        }

        private void buttonSave_Click(object sender, RoutedEventArgs e)
        {
            using (var service = new Service1Client())
            {
                // Get the blogs that are bound to the screen
                var blogsViewSource = (CollectionViewSource)this.FindResource("blogViewSource");
                var blogs = (List<Blog>)blogsViewSource.Source;

                // Save all Blogs and related Posts
                foreach (var blog in blogs)
                {
                    service.UpdateBlog(blog);
                }

                // Re-query for data to get database-generated keys etc.
                blogsViewSource.Source = service.GetBlogs().ToList();
            }
        }
    }
}

```

Teraz można uruchomić aplikacji, aby zobaczyć go w działaniu.

- Kliknij prawym przyciskiem myszy **STESample.WPFTest** projektu w **Eksploratora rozwiązań** i wybierz **debugowanie -> Uruchom nowe wystąpienie**
- Można manipulować danymi na ekranie i zapisać go za pośrednictwem usługi przy użyciu **Zapisz** przycisku

Blog Id	Name	Url
1	ADO.NET Blog	blogs.msdn.com/adonet
3	My Blog	

Post Id	Title	Content
5	Welcome	This is my first post...

Rejestrowanie i przechwytyuje operacji bazy danych

13.09.2018 • 22 minutes to read • [Edit Online](#)

NOTE

EF6 poczawszy tylko — funkcje, interfejsów API itp. z opisem na tej stronie zostały wprowadzone w programie Entity Framework 6. Jeśli używasz starszej wersji, niektóre lub wszystkie informacje, nie ma zastosowania.

Począwszy od platformy Entity Framework 6, w dowolnym momencie Entity Framework wysyła polecenie do bazy danych to polecenie może zostać przechwycone przez kod aplikacji. Najczęściej jest używana do logowania SQL, ale może również służyć do modyfikowania lub Przerwij polecenie.

W szczególności EF obejmuje:

- Właściwości rejestracji w kontekście podobnie jak `DataContext.Log` w składniku LINQ to SQL
- Mechanizm, aby dostosować zawartość i formatowania danych wyjściowych wysyłane do dziennika
- Niskiego poziomu bloków konstrukcyjnych dla przejmowanie, dzięki czemu większa kontrola/elastycznie

Właściwości rejestracji w kontekście

Można ustawić właściwości `DbContext.Database.Log` delegata dla dowolnej metody, która przyjmuje ciąg. Najczęściej jest używany z dowolnym `TextWriter`, ustawiając jej do metody "Zapisywanie" tego `TextWriter`. Moduł zapisujący zostaną zarejestrowane SQL wszystkich generowanych przez bieżącego kontekstu. Na przykład poniższy kod zostanie dziennika SQL do konsoli:

```
using (var context = new BlogContext())
{
    context.Database.Log = Console.Write;

    // Your code here...
}
```

Należy zauważyć, że kontekst `Console.Write` — ustawiono `Database.Log`. To wszystko, co jest potrzebne do logowania SQL do konsoli.

Dodajmy kilka prostych kodów zapytania/`insert/update`, dzięki czemu możemy zobaczyć pewne dane wyjściowe:

```
using (var context = new BlogContext())
{
    context.Database.Log = Console.Write;

    var blog = context.Blogs.First(b => b.Title == "One Unicorn");

    blog.Posts.First().Title = "Green Eggs and Ham";

    blog.Posts.Add(new Post { Title = "I do not like them!" });

    context.SaveChangesAsync().Wait();
}
```

Spowoduje to wygenerowanie następujące dane wyjściowe:

```

SELECT TOP (1)
    [Extent1].[Id] AS [Id],
    [Extent1].[Title] AS [Title]
    FROM [dbo].[Blogs] AS [Extent1]
    WHERE (N'One Unicorn' = [Extent1].[Title]) AND ([Extent1].[Title] IS NOT NULL)
-- Executing at 10/8/2013 10:55:41 AM -07:00
-- Completed in 4 ms with result: SqlDataReader

SELECT
    [Extent1].[Id] AS [Id],
    [Extent1].[Title] AS [Title],
    [Extent1].[BlogId] AS [BlogId]
    FROM [dbo].[Posts] AS [Extent1]
    WHERE [Extent1].[BlogId] = @EntityKeyValue1
-- EntityKeyValue1: '1' (Type = Int32)
-- Executing at 10/8/2013 10:55:41 AM -07:00
-- Completed in 2 ms with result: SqlDataReader

UPDATE [dbo].[Posts]
SET [Title] = @0
WHERE ([Id] = @1)
-- @0: 'Green Eggs and Ham' (Type = String, Size = -1)
-- @1: '1' (Type = Int32)
-- Executing asynchronously at 10/8/2013 10:55:41 AM -07:00
-- Completed in 12 ms with result: 1

INSERT [dbo].[Posts]([Title], [BlogId])
VALUES (@0, @1)
SELECT [Id]
FROM [dbo].[Posts]
WHERE @@ROWCOUNT > 0 AND [Id] = scope_identity()
-- @0: 'I do not like them!' (Type = String, Size = -1)
-- @1: '1' (Type = Int32)
-- Executing asynchronously at 10/8/2013 10:55:41 AM -07:00
-- Completed in 2 ms with result: SqlDataReader

```

(Należy pamiętać, że to dane wyjściowe przy założeniu, że już stało się inicjowanie bazy danych. Jeśli inicjowanie bazy danych nie ma już wystąpił, a następnie będzie można o wiele więcej danych wyjściowych, przedstawiający wszystkie prace migracji nie dzieje się w tle, aby sprawdzić lub Utwórz nową bazę danych.)

Co pobiera zarejestrowane?

Gdy właściwość dziennika ma wartość wszystkie poniższe będą rejestrowane:

- SQL dla różnych rodzajów poleceń. Na przykład:
 - Zapytania, łącznie z normalnym zapytań LINQ, eSQL kwerend i pierwotne zapytania z metody takie jak `SqlQuery`
 - Operacje wstawiania, aktualizacji i usuwania, wygenerowane jako część `SaveChanges`
 - Podczas ładowania zapytania, np. tych generowanych przez powolne ładowanie relacji
- Parametry
- Określa, czy polecenie jest wykonywane asynchronicznie
- Sygnatura czasowa wskazująca, podczas uruchamiania polecenia wykonywania
- Czy polecenie zostało wykonane pomyślnie, nie powiodło się, zgłaszając wyjątek lub asynchroniczne, zostało anulowane
- Niektóre wskazania wartość wyniku
- Kwota przybliżony czas potrzebny do wykonania polecenia. Należy pamiętać, że jest to czas wysyłanie polecenia do pobierania obiektu wyników, ponownie. Nie ma czasu można odczytać wyników.

Patrząc powyższych danych wyjściowych przykładu, każdy z czterech poleceń rejestrowane są:

- Zapytanie wynikające z wywołania do kontekstu. Blogs.First
 - Należy zauważyc, że metoda ToString pobierania SQL nie będzie działać dla tego zapytania, ponieważ "First" nie zawiera element IQueryble, na którym może być wywoływana ToString
- Zapytanie wynikające z powolne ładowanie blogu. Wpisy
 - Zwróć uwagę, dzieje się szczegóły parametrów dla wartości klucza, dla którego opóźnieniem ładowania
 - Rejestrowane są tylko te właściwości parametru, które są ustawione na wartości innych niż domyślne. Na przykład właściwość rozmiaru jest wyświetlna tylko jeśli jest różna od zera.
- Dwa polecenia wynikające z SaveChangesAsync; jeden dla aktualizacji zmienić tytuł wpisu, drugie dla instrukcji insert dodać nowy wpis
 - Zwróć uwagę, szczegóły parametrów właściwości klucza Obcego i tytułu
 - Należy zauważyc, że te polecenia są wykonywane asynchronicznie

Rejestrowanie w różnych miejscach

Jak wspomniano powyżej logowanie do konsoli jest bardzo proste. Jest również łatwe logowanie do pamięci, plików, itp. przy użyciu różnych rodzajów z [TextWriter](#).

Jeśli znasz za pomocą LINQ to SQL, można zauważyc, że w składniku LINQ to SQL ustawiono właściwość dziennika do rzeczywistego TextWriter obiektu (na przykład Console.Out) podczas w programie EF ustawiono właściwość dziennika do metody, która akceptuje ciąg (na przykład Console.WriteLine — lub Console.Out.WriteLine). Przyczyną jest oddzielenie EF z TextWriter, akceptując dowolnym delegatem, który może działać jako obiekt sink dla ciągów. Założmy, że masz już pewne struktury rejestrowania i definiuje metodę rejestrowania w następujący sposób:

```
public class MyLogger
{
    public void Log(string component, string message)
    {
        Console.WriteLine("Component: {0} Message: {1} ", component, message);
    }
}
```

To może podłączany do właściwości rejestrowania EF następująco:

```
var logger = new MyLogger();
context.Database.Log = s => logger.Log("EFApp", s);
```

Rejestrowanie wyników

Rejestrator domyślny tekst polecenia (SQL), parametry i dzienników wiersza "Executing" z sygnaturą czasową przed wysłaniem polecenia do bazy danych. "Ukończzone" wiersz zawierający czas, który upłynął są rejestrowane następujące wykonanie polecenia.

Należy pamiętać, że asynchroniczne "ukończony" wiersz polecenia nie jest rejestrowane, dopóki zadanie asynchroniczne faktycznie kończy, kończy się niepowodzeniem lub zostanie anulowane.

"Ukończzone" wiersz zawiera różne informacje w zależności od typu polecenia i określa, czy wykonywanie zakończyło się pomyślnie.

Pomyślne wykonanie

W przypadku poleceń, które się to odbyć danych wyjściowych jest "wykonane w x ms z wynikiem:" następuje niektóre wskazania wynik był. Dla poleceń, które zwracają wynik czytnik danych oznaczenie jest typ [obiekt DbDataReader](#) zwracane. Dla poleceń, które zwracają wartość całkowitą, takich jak aktualizacja przedstawionego

powyżej wyniku, przedstawione polecenia jest tym liczbą całkowitą.

Wykonywanie nie powiodło się

Dla poleceń, które się nie powieś, zostanie zgłoszony wyjątek dane wyjściowe zawierają komunikat z wyjątku. Na przykład przy użyciu `SqlQuery` zapytania względem tabeli, która istnieje będzie wynik w dzienniku danych wyjściowych podobny do poniższego:

```
SELECT * from ThisTableIsMissing
-- Executing at 5/13/2013 10:19:05 AM
-- Failed in 1 ms with error: Invalid object name 'ThisTableIsMissing'.
```

Anulowane wykonywanie

Dla poleceń asynchronicznych, gdy zadanie zostanie anulowane może się zdarzyć, Niepowodzenie z powodu wyjątku, ponieważ jest to, jakie źródłowy dostawca ADO.NET często wykonuje, gdy podejmowana jest próba anulowania. Jeśli to nie jest realizowane, a zadanie zostanie anulowane nie pozostawia żadnych śladów dane wyjściowe będą wyglądać mniej więcej tak:

```
update Blogs set Title = 'No' where Id = -1
-- Executing asynchronously at 5/13/2013 10:21:10 AM
-- Canceled in 1 ms
```

Zmiana zawartości dziennika i formatowanie

Dzieje się w tle `Database.Log` sprawia, że właściwość użyć obiektu `DatabaseLogFormatter`. Ten obiekt skutecznie wiąże implementację `IDbCommandInterceptor` (patrz poniżej) do delegata, który akceptuje ciągi i typu `DbContext`. Oznacza to, że metody `DatabaseLogFormatter` są wywoływane przed i po wykonaniu polecenia przez EF. Te metody `DatabaseLogFormatter` zbierania i sformatować dane wyjściowe dziennika i wysyłać je do obiektu delegowanego.

Dostosowywanie `DatabaseLogFormatter`

Zmiana, co jest rejestrowane i sposobu formatowania można osiągnąć, tworząc nową klasę pochodną `DatabaseLogFormatter`, która zastępuje metody zgodnie z potrzebami. Najbardziej typowe metody do przesłonięcia to:

- `LogCommand` — to zmienić, aby zmienić sposób polecenia są rejestrowane, przed wykonaniem. Domyślnie `LogCommand` wywołuje `LogParameter` dla każdego parametru; można może się tak samo w przesłonięcia lub obsługuje parametrów inaczej zamiast tego.
- `LogResult` — to zmienić, aby zmienić sposób wynik wykonania polecenia jest rejestrowane.
- `LogParameter` — to zmienić, aby zmienić formatowanie i zawartość parametru rejestrowania.

Na przykład założymy, że Chcieliśmy się jednym wierszu, przed wysłaniem każdego polecenia w bazie danych. Można to zrobić za pomocą dwóch zastępień:

- Zastąp `LogCommand` do formatu i zapisu w jednym wierszu programu SQL Server
- Zastąpienie `LogResult`, aby nic nie rób.

Kod powinien wyglądać mniej więcej tak:

```

public class OneLineFormatter : DatabaseLogFormatter
{
    public OneLineFormatter(DbContext context, Action<string> writeAction)
        : base(context, writeAction)
    {
    }

    public override void LogCommand<TResult>(
        DbCommand command, DbCommandInterceptionContext<TResult> interceptionContext)
    {
        Write(string.Format(
            "Context '{0}' is executing command '{1}'{2}",
            Context.GetType().Name,
            command.CommandText.Replace(Environment.NewLine, ""),
            Environment.NewLine));
    }

    public override void LogResult<TResult>(
        DbCommand command, DbCommandInterceptionContext<TResult> interceptionContext)
    {
    }
}

```

Do rejestrowania danych wyjściowych po prostu wywołanie metody zapisu, która będzie wysyłać dane wyjściowe do delegata skonfigurowanego zapisu.

(Zwróć uwagę, że ten kod jest uproszczony usuwania podziałów tylko jako przykład. Jego prawdopodobnie nie będą działać dobrze w przypadku wyświetlania złożonych SQL.)

Ustawienie DatabaseLogFormatter

Nowa klasa DatabaseLogFormatter od momentu utworzenia go musi być zarejestrowane przy użyciu programu EF. Odbywa się przy użyciu konfiguracji na podstawie kodu. Mówiąc oznacza to, tworzenie nowej klasy, która wynika z DbConfiguration z tego samego zestawu jako swojej klasy DbContext, a następnie wywołując SetDatabaseLogFormatter w Konstruktorze tej nowej klasy. Na przykład:

```

public class MyDbConfiguration : DbConfiguration
{
    public MyDbConfiguration()
    {
        SetDatabaseLogFormatter(
            (context, writeAction) => new OneLineFormatter(context, writeAction));
    }
}

```

Za pomocą nowego DatabaseLogFormatter

Ten nowy DatabaseLogFormatter będzie używany w dowolnym momencie Database.Log jest ustawiona. Tak uruchamiając kod z części 1 teraz spowoduje następujące dane wyjściowe:

```

Context 'BlogContext' is executing command 'SELECT TOP (1) [Extent1].[Id] AS [Id], [Extent1].[Title] AS [Title]FROM [dbo].[Blogs] AS [Extent1]WHERE (N'One Unicorn' = [Extent1].[Title]) AND ([Extent1].[Title] IS NOT NULL)'
Context 'BlogContext' is executing command 'SELECT [Extent1].[Id] AS [Id], [Extent1].[Title] AS [Title], [Extent1].[BlogId] AS [BlogId]FROM [dbo].[Posts] AS [Extent1]WHERE [Extent1].[BlogId] = @EntityKeyValue1'
Context 'BlogContext' is executing command 'update [dbo].[Posts]set [Title] = @0where ([Id] = @1)'
Context 'BlogContext' is executing command 'insert [dbo].[Posts]([Title], [BlogId])values (@0, @1)select [Id]from [dbo].[Posts]where @@rowcount > 0 and [Id] = scope_identity()'

```

Bloki konstrukcyjne przejmowanie

Do tej pory mają zobaczyliśmy, jak używa DbContext.Database.Log logowania SQL generowane przez EF. Ale ten kod jest faktycznie fasada stosunkowo alokowania elastycznego za pośrednictwem niektórych niskiego poziomu bloków konstrukcyjnych dla bardziej ogólnych przejmowanie.

Przejmowanie interfejsów

Kod zatrzymania opiera się na koncepcji przejmowanie interfejsów. Te interfejsy dziedziczą IDbInterceptor i definiowania metod, które są wywoływane, gdy EF wykonuje jakąś akcję. Celem jest zapewnienie jednego interfejsu na typ obiektu, które są przechwytywane. Na przykład interfejs IDbCommandInterceptor definiuje metody, które są wywoływane przed EF wywoływa ExecuteNonQuery ExecuteScalar, ExecuteReader oraz powiązanych metod. Podobnie interfejs definiuje metody, które są wywoływane po zakończeniu każdej z tych operacji. Klasa DatabaseLogFormatter, która przyrzeliśmy się powyżej implementuje ten interfejs logowania poleceń.

Kontekst przejmowanie

Patrząc metod w interfejsach interceptor wobec jego okaże się, że każde wywołanie podanego obiektu typu DbInterceptionContext lub typu pochodzi od klasy to przykład DbCommandInterceptionContext<>. Ten obiekt zawiera informacje kontekstowe o akcji, która zajmuje EF. Na przykład jeśli akcja jest wykonywana w imieniu typu DbContext, kontekstu DbContext jest uwzględnione w DbInterceptionContext. Podobnie do poleceń, które są wykonywane asynchronicznie, IsAsync flaga jest ustawiona na DbCommandInterceptionContext.

Obsługa wyników

DbCommandInterceptionContext<> klasa zawiera właściwości o nazwie wynik, OriginalResult, wyjątków i oryginalny wyjątek. Te właściwości są ustawione na wartość null/zero dla wywołania metod przejmowanie, które są wywoływane przed wykonaniem operacji — czyli dla... Wykonywanie metod. Jeśli operacja jest wykonywana, a zakończy się pomyślnie, następnie wynik i OriginalResult są ustawione na wynik operacji. Następnie można zaobserwować te wartości w metodach przejmowanie, które są wywoływane po wykonaniu operacji — czyli na... Wykonane metody. Podobnie jeśli operacja zgłosi, następnie właściwości wyjątku i oryginalny wyjątek zostaną ustawione.

Pomijanie wykonania

Jeśli ta opcja interceptor ustawia właściwości wyniku, zanim to polecenie zostało wykonane (w jednym z... Wykonywanie metod) następnie EF nie podejmie próby rzeczywistego wykonania polecenia, ale zamiast tego użycie po prostu zestaw wyników. Innymi słowy interceptor można pominąć wykonywanie polecenia, ale ma EF kontynuowane tak, jakby było zostały wykonane polecenie.

Przykład sposobu użycia tego może być jest polecenia przetwarzania wsadowego, która była tradycyjnie wykonywana przy użyciu dostawcy zawiązania. Interceptor będzie przechowywać polecenia do późniejszego wykonania jako zadania wsadowego, ale będzie "poudawać" do programu EF, że polecenie było wykonywane w zwykły. Należy pamiętać, że wymaga więcej niż to do zaimplementowania przetwarzania wsadowego, ale jest to przykład jak zmiana wyniku przejmowanie mogą być używane.

Wykonanie również można pominąć, ustawiając właściwość wyjątku w jednej z... Wykonywanie metod. Powoduje to EF kontynuować, tak, jakby wykonanie operacji miał nie powiodło się, zwracając dany wyjątek. Oczywiście, może to spowodować aplikacji awarię, ale może to być także wyjątek przejściowy lub innych wyjątków, który jest obsługiwany przez EF. Na przykład to może służyć w środowiskach testowych do testowania zachowania aplikacji, gdy wykonywanie polecenia nie powiodło się.

Zmiana wyniku Po wykonaniu

Jeśli ta opcja ustawia interceptor właściwości wyniku Po wykonaniu polecenia (w jednym z... Wykonywanie metody), a następnie platforma EF użycie wynik zmieniono zamiast wynik, który faktycznie został zwrócony przez operację. Podobnie jeśli interceptor ustawia właściwość wyjątku po wykonaniu polecenia, następnie EF spowoduje zgłoszenie wyjątku zestaw tak, jakby operacji miał wyjątek.

Interceptor można również ustawić właściwość wyjątku na wartość null, aby wskazać, że nie należy zgłosić wyjątku. Może to być przydatne, jeśli nie można wykonać operacji, ale interceptor chce EF, aby kontynuować, tak,

jakby zakończyło się operacji. Zwykle obejmuje to również ustawienie wynik tak, aby EF jakąś wartość wynik, aby pracować, jak dugo.

OriginalResult i oryginalny wyjątek

Po wykonaniu operacji EF zostanie ustawiony wynik i OriginalResult właściwości, jeśli wykonanie nie zakończyła się niepowodzeniem lub właściwości wyjątku i oryginalny wyjątek, jeśli wykonanie nie powiodło się z powodu wyjątku.

Właściwości OriginalResult i oryginalny wyjątek są przeznaczone tylko do odczytu i ustawionych tylko przez EF po rzeczywistego wykonania operacji. Te właściwości nie może ustawić interceptory. Oznacza to, że wszelkie interceptor może rozróżnić wystąpi wyjątek lub wynik, która została ustawiona przez niektóre interceptor, w przeciwieństwie do rzeczywistego wyjątek lub wynik, który wystąpił podczas operacji został wykonany.

Rejestrowanie interceptory

Po utworzeniu klasy, która implementuje co najmniej jeden z interfejsów przejmowanie mogą być rejestrowane struktury jednostek przy użyciu klasy `DblInterception`. Na przykład:

```
DbInterception.Add(new NLogCommandInterceptor());
```

Interceptors można zarejestrować w taki sposób, na poziomie domeny aplikacji przy użyciu mechanizmu `DbConfiguration` konfiguracja na podstawie kodu.

Przykład: Rejestrowanie NLog

Teraz umieść to wszystko ze sobą przykładowi, przy użyciu `IDbCommandInterceptor` i [NLog](#) do:

- Ostrzeżenie dla dowolnego polecenia, który jest wykonywany bez asynchronicznie dziennika
- Błąd dla dowolnego polecenia, który zgłasza wyjątek podczas wykonywania

Poniżej przedstawiono klasy, która wykonuje rejestrowanie, które powinny być rejestrowane, jak pokazano powyżej:

```

public class NLogCommandInterceptor : IDbCommandInterceptor
{
    private static readonly Logger Logger = LogManager.GetCurrentClassLogger();

    public void NonQueryExecuting(
        SqlCommand command, SqlCommandInterceptionContext<int> interceptionContext)
    {
        LogIfNonAsync(command, interceptionContext);
    }

    public void NonQueryExecuted(
        SqlCommand command, SqlCommandInterceptionContext<int> interceptionContext)
    {
        LogIfError(command, interceptionContext);
    }

    public void ReaderExecuting(
        SqlCommand command, SqlCommandInterceptionContext<DbDataReader> interceptionContext)
    {
        LogIfNonAsync(command, interceptionContext);
    }

    public void ReaderExecuted(
        SqlCommand command, SqlCommandInterceptionContext<DbDataReader> interceptionContext)
    {
        LogIfError(command, interceptionContext);
    }

    public void ScalarExecuting(
        SqlCommand command, SqlCommandInterceptionContext<object> interceptionContext)
    {
        LogIfNonAsync(command, interceptionContext);
    }

    public void ScalarExecuted(
        SqlCommand command, SqlCommandInterceptionContext<object> interceptionContext)
    {
        LogIfError(command, interceptionContext);
    }

    private void LogIfNonAsync<TResult>(
        SqlCommand command, SqlCommandInterceptionContext<TResult> interceptionContext)
    {
        if (!interceptionContext.IsAsync)
        {
            Logger.Warn("Non-async command used: {0}", command.CommandText);
        }
    }

    private void LogIfError<TResult>(
        SqlCommand command, SqlCommandInterceptionContext<TResult> interceptionContext)
    {
        if (interceptionContext.Exception != null)
        {
            Logger.Error("Command {0} failed with exception {1}",
                command.CommandText, interceptionContext.Exception);
        }
    }
}

```

Zwróć uwagę, jak ten kod używa kontekstu przejmowania, aby dowiedzieć się, gdy polecenie jest wykonywane innych niż asynchronicznie, a także do wykrywania, gdy wystąpił błąd podczas wykonywania polecenia.

Zagadnienia dotyczące wydajności na platformie EF, 4, 5 i 6

25.10.2018 • 124 minutes to read • [Edit Online](#)

David Obando, Eric Dettinger i inne osoby

Data publikacji: Kwietnia 2012

Ostatnia aktualizacja: maj 2014

1. Wprowadzenie

Mapowania obiektowo-relacyjny struktury to wygodny sposób zapewnienia klasą abstrakcyjną dla dostępu do danych w aplikacji, zorientowane obiektowo. W przypadku aplikacji .NET zalecaną przez firmę Microsoft jest Obiektowo Entity Framework. Wszelkie abstrakcje jednak wydajności mogą stać się problemem.

Ten oficjalny dokument został zapisany do wyświetlenia zagadnienia związane z wydajnością podczas tworzenia aplikacji przy użyciu platformy Entity Framework, aby zaoferować deweloperom pomysł wewnętrzne algorytmy Entity Framework, które mogą wpływać na wydajność i zapewnienie porady dotyczące badania i poprawy wydajności w aplikacjach korzystających z programu Entity Framework. Istnieje wiele dobrych tematy dotyczące wydajności już dostępne w sieci web, a także staraliśmy, wskazując do tych zasobów, jeśli jest to możliwe.

Wydajność jest trudne tematu. Ten oficjalny dokument jest przeznaczony jako zasób ułatwiające wprowadzeniu wydajności związane z decyzje dotyczące aplikacji korzystających z programu Entity Framework. Wprowadzono niektóre metryki testu, aby zademonstrować wydajność, ale te metryki nie są przeznaczone jako bezwzględne wskaźników wydajności, które będą wyświetlane w aplikacji.

Ze względów praktycznych w tym dokumencie przyjęto założenie, Entity Framework 4 jest uruchamiany program .NET 4.0 i Entity Framework 5 i 6 są uruchamiane w ramach platformy .NET 4.5. Wiele ulepszeń wydajności dla programu Entity Framework 5 znajdują się w podstawowe składniki, które są dostarczane za pomocą platformy .NET 4.5.

Entity Framework 6 jest poza pasmem wersji i nie zależy od składników platformy Entity Framework, które są dostarczane za pomocą platformy .NET. Entity Framework 6 działają zarówno w przypadku programu .NET 4.0, jak i .NET 4.5 i zaoferować korzyść duża wydajność dla użytkowników, którzy jeszcze nie uaktualniono z programu .NET 4.0, ale ma najnowsze elementy platformy Entity Framework w aplikacjach. Gdy ten dokument jest wspomniany Entity Framework 6, odwołuje się do najnowszej wersji, dostępnym w momencie pisania tego dokumentu: wersja 6.1.0.

2. Zimnych programu vs. Wykonywanie zapytania bez wyłączenia zasilania

Podczas pierwszego dowolne zapytanie jest wykonywane względem danego modelu Entity Framework jest dużo pracy w tle, aby załadować i sprawdzania poprawności modelu. Często nazywamy to pierwsze zapytanie jako zapytanie "zimnymi". Dodatkowe zapytania względem modelu już załadowana są określone jako "cieplych" zapytań i jest znacznie szybsze.

Teraz możesz ogólny widok, w której jest zużywany czas podczas wykonywania zapytania za pomocą programu Entity Framework i zobacz, gdzie poprawiamy rzeczy w Entity Framework 6.

Pierwsze wykonanie zapytania — zimnych zapytania

ZAPISY UŻYTKOWNIKA KODU	AKCJA	EF4 WPŁYW NA WYDAJNOŚĆ	EF5 WPŁYW NA WYDAJNOŚĆ	EF6 WPŁYW NA WYDAJNOŚĆ
<pre>using(var db = new MyContext()) {</pre>	Tworzenie kontekstu	Średni	Średni	Niska
<pre>var q1 = from c in db.Customers where c.Id == id1 select c;</pre>	Tworzenie wyrażenia kwerendy	Niska	Niska	Niska
<pre>var c1 = q1.First();</pre>	Wykonywanie zapytania LINQ	<ul style="list-style-type: none"> -Ładowanie metadanych: wysoki, ale pamięci podrycznej — Wyświetlanie generacji: potencjalnie bardzo wysoka, ale pamięci podrycznej -Parametr oceny: średni -Zapytania tłumaczenia: średni -Generowanie materializer: średnie, ale pamięci podrycznej — Wykonywanie kwerend bazy danych: potencjalnie dużego + Connection.Open + Command.ExecuteReader + DataReader.Read Materializacja obiektu: średni -Wyszukiwanie identity: średni 	<ul style="list-style-type: none"> -Ładowanie metadanych: wysoki, ale pamięci podrycznej — Wyświetlanie generacji: potencjalnie bardzo wysoka, ale pamięci podrycznej -Parametr oceny: Niski -Zapytania tłumaczenia: średnie, ale pamięci podrycznej -Generowanie materializer: średnie, ale pamięci podrycznej — Wykonywanie kwerend bazy danych: potencjalnie dużego (lepsze zapytania w niektórych sytuacjach) + Connection.Open + Command.ExecuteReader + DataReader.Read Materializacja obiektu: średni -Wyszukiwanie identity: średni 	<ul style="list-style-type: none"> -Ładowanie metadanych: wysoki, ale pamięci podrycznej — Wyświetlanie generacji: średnie, ale pamięci podrycznej -Parametr oceny: Niski -Zapytania tłumaczenia: średnie, ale pamięci podrycznej -Generowanie materializer: średnie, ale pamięci podrycznej — Wykonywanie kwerend bazy danych: potencjalnie dużego (lepsze zapytania w niektórych sytuacjach) + Connection.Open + Command.ExecuteReader + DataReader.Read Materializacja obiektu: średni (szycie niż EF5) -Wyszukiwanie identity: średni
}	Connection.Close	Niska	Niska	Niska

Drugiego wykonywania zapytania — zapytania bez wyłączenia zasilania

ZAPISY UŻYTKOWNIKA KODU	AKCJA	EF4 WPŁYW NA WYDAJNOŚĆ	EF5 WPŁYW NA WYDAJNOŚĆ	EF6 WPŁYW NA WYDAJNOŚĆ
<pre>using(var db = new MyContext()) {</pre>	Tworzenie kontekstu	Średni	Średni	Niska

ZAPISY UŻYTKOWNIKA KODU	AKCJA	EF4 WPŁYW NA WYDAJNOŚĆ	EF5 WPŁYW NA WYDAJNOŚĆ	EF6 WPŁYW NA WYDAJNOŚĆ
<pre>var q1 = from c in db.Customers where c.Id == id1 select c;</pre>	Tworzenie wyrażenia kwerendy	Niska	Niska	Niska
<pre>var c1 = q1.First();</pre>	Wykonywanie zapytania LINQ	<ul style="list-style-type: none"> -Metadanych ładowania wyszukiwania: wysoka, ale pamięci podręcznej niski — Wyświetlanie generowania wyszukiwania: potencjalnie bardzo wysoka, lecz buforowane niski -Parametr oceny: średni -Zapytania tłumaczenia wyszukiwania: średni -Materializer generowania wyszukiwania: średnie, ale pamięci podręcznej niski — Wykonywanie kwerend bazy danych: potencjalnie dużego + Connection.Open + Command.ExecuteReader + DataReader.Read Materializacja obiektu: średni -Wyszukiwanie identity: średni 	<ul style="list-style-type: none"> -Metadanych ładowania wyszukiwania: wysoka, ale pamięci podręcznej niski — Wyświetlanie generowania wyszukiwania: potencjalnie bardzo wysoka, lecz buforowane niski -Parametr oceny: Niski -Zapytania tłumaczenia wyszukiwania: średnie, ale pamięci podręcznej niski -Materializer generowania wyszukiwania: średnie, ale pamięci podręcznej niski — Wykonywanie kwerend bazy danych: potencjalnie dużego (lepsze zapytania w niektórych sytuacjach) + Connection.Open + Command.ExecuteReader + DataReader.Read Materializacja obiektu: średni (szycie niż EF5) -Wyszukiwanie identity: średni 	<ul style="list-style-type: none"> -Metadanych ładowania wyszukiwania: wysoka, ale pamięci podręcznej niski — Wyświetlanie generowania wyszukiwania: średnie, ale pamięci podręcznej niski -Parametr oceny: Niski -Zapytania tłumaczenia wyszukiwania: średnie, ale pamięci podręcznej niski -Materializer generowania wyszukiwania: średnie, ale pamięci podręcznej niski — Wykonywanie kwerend bazy danych: potencjalnie dużego (lepsze zapytania w niektórych sytuacjach) + Connection.Open + Command.ExecuteReader + DataReader.Read Materializacja obiektu: średni (szycie niż EF5) -Wyszukiwanie identity: średni
}	Connection.Close	Niska	Niska	Niska

Istnieje kilka sposobów, aby zmniejszyć koszt wydajności zapytań, ciepło i zimno, a firma Microsoft będzie Przyjrzysią się one w poniższej sekcji. W szczególności Zapoznamy się obniżyć koszty ładowania zimnych zapytań przy użyciu wstępnie wygenerowanych widoków, które należy zmniejszyć wydajność problemy doświadczenie podczas generowania widoku modelu. Dla zapytań bez wyłączania zasilania omówimy, buforowanie planu zapytania, nie śledzenia zapytań i opcje wykonanie innego zapytania.

2.1 Generowanie widoku co to jest?

Aby zrozumieć, jakie widoku Generowanie jest, firma Microsoft musisz najpierw zrozumieć, co to są "Mapowanie widoków". Widoki mapowania są reprezentacji pliku wykonywalnego przekształcenia określony w mapowaniu dla każdego zestawu jednostek i skojarzenia. Wewnętrznie te widoki mapowania przybrać CQTs (canonical zapytania drzewa). Istnieją dwa rodzaje widokach mapowania:

- Widoki kwerendę: reprezentują one to konieczne, można przejść od schematu bazy danych do modelu koncepcyjnego transformacji.
- Aktualizowanie widoków: reprezentuje przekształcenie konieczne przechodzenie z modelu koncepcyjnego do schematu bazy danych.

Należy pamiętać, że model koncepcyjny różnić się od schematu bazy danych na różne sposoby. Na przykład jeden pojedynczej tabeli mogą służyć do przechowywania danych dla dwóch typów jednostek innej. Dziedziczenie i mapowania nietrywialnymi pełnić rolę, w złożoność widokach mapowania.

Proces przetwarzania tych widoków, na podstawie specyfikacji mapowania to tak zwany generowania widoku. Generowanie widoku albo korzystać z miejsca dynamicznie podczas ładowania modelu lub w czasie komplikacji, używając "wstępnie wygenerowanych widoków"; te ostatnie są serializowane w postaci instrukcji języka SQL jednostki, C# lub plik VB.

Po wygenerowaniu widoków, one również są weryfikowane. Z punktu widzenia wydajności większość koszt generowania widoku jest faktycznie weryfikacji widoków, który zapewnia, że połączenia między jednostkami sens i mają Kardynalność prawidłowe dla wszystkich obsługiwanych operacji.

Podczas wykonywania zapytania za pośrednictwem zestawu jednostek zapytania jest połączony z odpowiedniego widoku zapytania, a wynik tej kompozycji jest uruchamiany przez kompilator planu, aby utworzyć reprezentację zapytanie, które może zrozumieć magazyn zapasowy. Dla programu SQL Server ostateczny wynik tej komplikacji będzie instrukcję języka T-SQL ZAZNACZYĆ. Po raz pierwszy wykonać aktualizację za pośrednictwem zestawu jednostek, widok aktualizacji jest uruchamiane za pomocą podobnej do przekształcania go w instrukcji DML dla docelowej bazy danych.

2.2 czynniki, które mają wpływ na wydajność generowania widoku

Wydajność krok generowania widoku zależy nie tylko rozmiar modelu, ale także na połączonych jak model jest. Jeśli dwie jednostki są połączone za pośrednictwem łańcuchów dziedziczenia lub skojarzenia, są one określone jako podłączone. Podobnie jeśli dwie tabele są połączone za pomocą klucza obcego, są one połączone. Jak zwiększyć liczbę połączonych jednostek, jak i tabele w swoje schematy, generowanie widoku koszt zwiększa się.

Algorytmu, używanego do generowania i zweryfikować widoków jest wykładniczego w najgorszym przypadku, chociaż używamy niektóre optymalizacje tego. Największych czynników, które wydaje się, że negatywnie wpływać na wydajność są następujące:

- Rozmiar modelu odnoszące się do liczby jednostek i ilość skojarzenia między tymi jednostkami.
- Złożoność modelu, szczególnie dziedziczenia obejmujące wiele typów.
- Użycie niezależnych skojarzeń, zamiast skojarzeń klucza obcego.

W przypadku małych, prostych modeli kosztów może być wystarczająco mała, aby nie odblokowane za pomocą wstępnie wygenerowanych widoków. Jak zwiększyć rozmiar modelu i złożoność, dostępnych jest kilka opcji, które można zmniejszyć koszt generowania widoku i sprawdzania poprawności.

2.3 widoków Pre-Generated przy użyciu modelu zmniejszy czas ładowania

Aby uzyskać szczegółowe informacje dotyczące sposobu używania wstępnie wygenerowanych widoków w programie Entity Framework 6 odwiedź [Pre-Generated mapowanie widoków](#)

2.3.1 wstępnie wygenerowanych widoków za pomocą programu Entity Framework Power Tools Community Edition

Możesz użyć [Entity Framework 6 Power Tools Community Edition](#) można wygenerować widoków plików EDMX i Code First modeli kliknij prawym przyciskiem myszy plik klasy modelu, a następnie wybierz pozycję "Generuj widoki" przy użyciu menu platformy Entity Framework. Entity Framework Power Tools Community Edition działa tylko w kontekstach pochodzących od typu DbContext.

2.3.2 sposób używania wstępnie wygenerowanych widoków z modelem, który został utworzony przez EDMGen

EDMGen to narzędzie jest dostarczany za pomocą platformy .NET, która działa z programu Entity Framework 4 i 5, ale nie z programu Entity Framework 6. EDMGen umożliwia generowanie pliku modelu warstwy obiektu i

widoków z wiersza polecenia. Jedną z danych wyjściowych będzie plik widoków w języku wybranym języku VB lub C#. Jest to plik kodu zawierający fragmenty jednostki SQL dla każdego zestawu jednostek. Aby włączyć wstępnie wygenerowanych widoków, po prostu Dołącz plik w projekcie.

Jeśli ręcznie wprowadzić zmiany plików schematów dla modelu, należy ponownie wygenerować plik widoków. Można to zrobić, uruchamiając EDMGen z **/mode:ViewGeneration** flagi.

2.3.3 jak widoków Pre-Generated za pomocą pliku EDMX

Można również użyć EDMGen można wygenerować widoków dla pliku EDMX — wcześniej odwołania temacie w witrynie MSDN zawiera opis sposobu dodawania Zdarzenie sprzed komplikacji, w tym -, ale jest to skomplikowane i istnieją przypadki, gdy nie jest możliwe. Ogólnie łatwiej szablon T4 umożliwia generowanie widoków, gdy model znajduje się w pliku edmx.

Blog zespołu programu ADO.NET ma wpis, który opisuje sposób używania szablon T4 do generowania widoku (<http://blogs.msdn.com/b/adonet/archive/2008/06/20/how-to-use-a-t4-template-for-view-generation.aspx>). Ten wpis zawiera szablon, który może być pobrane i dodane do projektu. Szablon został napisany dla pierwszej wersji programu Entity Framework, więc one nie są gwarantowane najnowsze wersje platformy Entity Framework. Jednakże można pobrać bardziej aktualny zestaw szablonów generowania widoku Entity Framework 4 i 5 from galerii Visual Studio:

- VB.NET: <<http://visualstudiogallery.msdn.microsoft.com/118b44f2-1b91-4de2-a584-7a680418941d>>
- C#: <<http://visualstudiogallery.msdn.microsoft.com/ae7730ce-ddab-470f-8456-1b313cd2c44d>>

Jeśli używasz platformy Entity Framework 6 można uzyskać widok szablon T4 generacji z galerii Visual Studio na <<http://visualstudiogallery.msdn.microsoft.com/18a7db90-6705-4d19-9dd1-0a6c23d0751f>>.

2.4 obniżyć koszty generowania widoku

Z pomocą wstępnie wygenerowanych widoków przenosi koszt generowania widoku z modelu, załadowanie (czas wykonywania) do czasu projektowania. Gdy poprawia to wydajność uruchamiania w środowisku uruchomieniowym, będzie nadal występować ból generowania widoku podczas programowania. Istnieje kilka dodatkowe wskazówki, które mogą pomóc zmniejszyć koszt generowania widoku, zarówno w czasie komplikacji i w czasie wykonywania.

2.4.1 za pomocą skojarzeń klucza obcego, aby zmniejszyć koszt generowania widoku

Zaobserwowałyśmy liczbę przypadków, w której przełączanie skojarzeń w modelu z niezależnym skojarzeniem obcego skojarzenia klucza znacznie ulepszona czas potrzebny do generowania widoku.

Aby zademonstrować to ulepszenie, firma Microsoft generowane dwie wersje modelu Navision przy użyciu EDMGen. *Uwaga: see appendix C for opis modelu Navision.* Navision model jest interesujące dla tego ćwiczenia z powodu ich dużej ilości jednostek i relacji między nimi.

Jedną wersję tego modelu bardzo dużych został wygenerowany z użyciem obcego skojarzenia kluczy i innych został wygenerowany z użyciem niezależnych skojarzenia. Następnie timed się, jak długo można wygenerować widoków dla każdego modelu. Jednostki Framework5 test używał GenerateViews() metody z klasy EntityViewGenerator, można wygenerować widoków, podczas testu Entity Framework 6 GenerateViews() metody z klasy obiekt StorageMappingItemCollection. To z powodu restrukturyzacji kod, który wystąpił w bazie kodu podlegającej Entity Framework 6.

Z pomocą programu Entity Framework 5, widok generacji dla modelu przy użyciu kluczy obcych trwało 65 minut w komputerze laboratoryjnym. Wiadomo jak długo zajęłyby do generowania widoków dla modelu, który używane niezależnie od skojarzenia. Pozostawiliśmy test uruchomiony w ciągu miesiąca, zanim komputer został ponownie uruchomiony w nasze laboratorium, aby zainstalować comiesięcznych aktualizacji.

Widok generacji dla modelu przy użyciu kluczy obcych przy użyciu platformy Entity Framework 6, zajęło 28 sekundach w tym samym komputerze laboratoryjnym. Widok generacji dla modelu, który używa niezależnych skojarzeń zajęło s 58. Ulepszenia jej kodu generowania widoku poświęconej Entity Framework 6 oznaczają, że w przypadku wielu projektów nie będzie już konieczne wstępnie wygenerowanych widoków, aby uzyskać krótszy

czas uruchamiania.

Jest to ważne uwagi, który wstępnie generowanie widoków w programie Entity Framework 4 i 5 może odbywać się przy użyciu EDMGen lub narzędzi Entity Framework Power Tools. Entity Framework 6 widoku generowania może odbywać się za pomocą narzędzi Entity Framework Power Tools lub programowo, zgodnie z opisem w [Pre-Generated mapowanie widoków](#).

2.4.1.1 jak do używania kluczy obcych zamiast niezależnych skojarzenia

Korzystając z EDMGen lub Projektant jednostki w programie Visual Studio, otrzymasz FKs domyślnie, ale zajmuje tylko OK pojedynczej flagi pola wyboru lub wiersza polecenia można przełączać się między FKs i IAs.

W przypadku dużych model Code First przy użyciu niezależnych skojarzenia mają ten sam efekt na generowania widoku. Możesz uniknąć wpływu przez dołączenie klucza obcego właściwości klasy dla obiektów zależnych, chociaż niektórzy deweloperzy będą należeć wziąć pod uwagę ten element, aby być zanieczyszczanie ich modelu obiektów. Można znaleźć więcej informacji na ten temat w <<http://blog.oneunicorn.com/2011/12/11/whats-the-deal-with-mapping-foreign-keys-using-the-entity-framework/>>.

KORZYSTAJĄC Z	ZRÓB TO
Projektant ekranu	Po dodaniu skojarzenia między dwiema jednostkami, upewnij się, że masz ograniczenia referencyjnego. Ograniczenia referencyjne Poinformuj Entity Framework do używania kluczy obcych zamiast niezależnych skojarzenia. Aby uzyskać więcej informacji, odwiedź stronę < http://blogs.msdn.com/b/efdesign/archive/2009/03/16/foreign-keys-in-the-entity-framework.aspx >.
EDMGen	Generowanie plików z bazy danych za pomocą EDMGen, klucze obce będą przestrzegane i dodana do modelu, w związku z tym. Aby uzyskać więcej informacji na temat różnych opcji udostępnianych przez EDMGen odwiedź stronę http://msdn.microsoft.com/library/bb387165.aspx .
Najpierw kod	Zobacz sekcję "Konwencji relacji" pierwszy konwencje związane z zawiera informacje dotyczące sposobu uwzględniania właściwości klucza obcego na obiekty zależne, gdy za pomocą funkcji Code First.

2.4.2 przenoszenie modelu w osobnym zestawie

Gdy model znajduje się bezpośrednio w projekcie aplikacji i generowanie widoków przez zdarzenie sprzed komplikacji lub szablon T4, generowania widoku i sprawdzanie poprawności nastąpi zawsze wtedy, gdy projekt zostanie ponownie skompilowany, nawet jeśli nie można zmienić modelu. Jeśli przenoszenie modelu w osobnym zestawie i odwoływać się do niego z projektu aplikacji, można zapisać wprowadzić inne zmiany aplikacji bez konieczności ponownie skompiluj projekt zawierający modelu.

Uwaga: podczas przenoszenia modelu do oddzielnych zestawów Pamiętaj o skopiowaniu parametrów połączenia dla modelu w pliku konfiguracyjnym aplikacji projektu klienta.

2.4.3 wyłączyć sprawdzanie poprawności modelu opartego na edmx

Modele EDMX są weryfikowane w czasie komplikacji, nawet jeśli model jest bez zmian. Jeśli model została już zweryfikowana, można pominąć sprawdzanie poprawności w czasie komplikacji przez ustawienie właściwości "Sprawdzanie poprawności w komplikacji" na wartość false w oknie dialogowym właściwości. Po zmianie z mapowania lub modelu, można tymczasowo ponownie włączyć sprawdzanie poprawności, aby zweryfikować zmiany.

Należy pamiętać, że wprowadzono ulepszenia wydajności do programu Entity Framework Designer dla programu Entity Framework 6 i koszt "Sprawdzanie poprawności w komplikacji" jest znacznie niższa niż w poprzednich wersjach projektanta.

3 buforowania w programie Entity Framework

Entity Framework zawiera następujące rodzaje wbudowanej w pamięci podręcznej:

1. Obiekt z pamięci podręcznej — obiekt ObjectStateManager wbudowaną wystąpienie obiektu ObjectContext śledzi w pamięci obiektów, które zostały pobrane przy użyciu tego wystąpienia. To jest również nazywany pierwszego poziomu w pamięci podręcznej.
2. Buforowanie planu zapytania — ponowne użycie polecenia magazynu wygenerowany, gdy zapytanie jest wykonywane więcej niż jeden raz.
3. Metadane w pamięci podręcznej — Udostępnianie metadanych dla modelu w różnych połączeń do tego samego modelu.

Oprócz pamięci podręczne, które EF zapewnia gotowych specjalny rodzaj dostawcy danych ADO.NET, znane jako dostawcy opakowujące aplikacje można również rozszerzyć Entity Framework z pamięcią podręczną zawiera wyniki pobrane z bazy danych, nazywany również pamięć podręczna drugiego poziomu.

3.1 obiektu pamięci podręcznej

Domyślnie gdy jednostka jest zwracany w wynikach zapytania, przed EF materializuje, Obiekt ObjectContext sprawdzi, jeśli jednostki z tym samym kluczu został już załadowany w jego obiekcie ObjectStateManager. Jeśli jednostki z tych samych kluczów znajduje się już EF będzie dołączyć wyniki zapytania. Mimo że EF nadal będzie wysyłać zapytań w bazie danych, to zachowanie można pominąć większość koszt materializowania jednostki wiele razy.

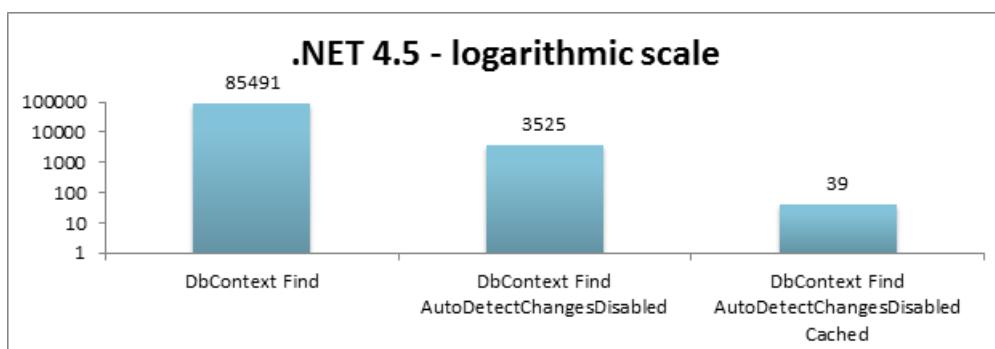
3.1.1 pobieranie jednostek z pamięci podręcznej obiektów korzystania z funkcji znajdowania typu DbContext

W przeciwieństwie do regularnych zapytania metody Find w DbSet (interfejsy API uwzględnione po raz pierwszy w EF 4.1) będzie wykonywać wyszukiwania w pamięci przed wystawieniem nawet zapytanie w bazie danych.

Należy zauważać, że dwa różne wystąpienia obiektu ObjectContext dwóch różnych wystąpień obiektu ObjectStateManager, co oznacza, do których mają oddzielny obiekt w pamięci podręcznej.

Znajdź używa wartość klucza podstawowego do podejmą próbę odnalezienia śledzone przez kontekst jednostki. Jeśli jednostki nie znajduje się w kontekście następnie wykonywane i oceniane w bazie danych zapytania i zwarcana jest wartość null, jeśli jednostka nie zostanie odnaleziona w kontekście lub w bazie danych. Należy pamiętać, że znajdowanie zwraca również wartość jednostek, które zostały dodane do kontekstu, ale nie zostały zapisane w bazie danych.

Brak jest brany pod uwagę wydajności do wykonania podczas korzystania z funkcji znajdowania. Wywołania do tej metody, domyślnie wyzwoli weryfikacji obiektu pamięci podręcznej w celu wykrycia zmian, które wciąż oczekują na zatwierdzenie w bazie danych. Ten proces może zająć bardzo kosztowny w przypadku bardzo dużej liczby obiektów w pamięci podręcznej obiektów lub wykresie dużego obiektu dodawane do pamięci podręcznej obiektów, ale można również zostać wyłączone. W niektórych przypadkach mogą postrzegać przez rzad wielkości różnicy podczas wywoływanego Znajdź metodę po wyłączeniu automatycznego wykrywania zmian. Jeszcze drugi rzad wielkości jest traktowany, gdy obiekt jest rzeczywiście w pamięci podręcznej, a gdy obiekt ma być pobierane z bazy danych. Oto przykładowy Graf za pomocą pomiarów dokonanych przy użyciu niektórych z naszych microbenchmarks wyrażony w milisekundach, wynosi 5000 jednostek:



Przykład Znajdź ze zmianami auto-detect wyłączone:

```
context.Configuration.AutoDetectChangesEnabled = false;
var product = context.Products.Find(productId);
context.Configuration.AutoDetectChangesEnabled = true;
...
```

Co należy wziąć pod uwagę podczas korzystania z metody Znajdź jest:

1. Jeśli obiekt nie jest w pamięci podręcznej z zalet wyszukiwania jest ujemna, ale składnia jest nadal prostsza niż zapytania według klucza.
2. Jeśli automatyczne wykrywanie zmian jest włączona może zwiększyć koszt metody Find, co o rząd wielkości i jeszcze bardziej w zależności od złożoności modelu i ilość jednostek w pamięci podręcznej obiektu.

Ponadto należy pamiętać, że znaleźć tylko zwraca obiekt, do którego szukasz, i go nie automatycznie ładowania jego skojarzone jednostki, jeśli nie są jeszcze w pamięci podręcznej obiektów. Jeśli musisz pobrać skojarzone jednostki można użyć zapytania według klucza przy użyciu wcześnie ładowanie. Aby uzyskać więcej informacji, zobacz **8.1 powolne ładowanie programu vs. Wczesne ładowanie**.

3.1.2 problemy z wydajnością, gdy pamięć podręczna obiekt zawiera wiele jednostek

Obiektu pamięci podręcznej pomaga zwiększyć ogólną szybkość reakcji Entity Framework. Jednak po pamięci podręcznej obiektów zawiera bardzo dużą ilość jednostek załadowane, który może wpływać na niektórych operacji, takich jak dodawanie, usuwanie, znajdź wpis, SaveChanges i wiele innych. W szczególności operacje, które wyzwala wywołanie metody DetectChanges będzie negatywny wpływ bardzo dużych obiektów w pamięci podręcznej. Metody DetectChanges synchronizuje wykres obiektu z obiektu state manager i spowoduje jego wydajności, określany bezpośrednio przez rozmiar wykresu obiektu. Aby uzyskać więcej informacji na temat metody DetectChanges zobacz [śledzenie zmian w jednostkach obiektów POCO](#).

Korzystając z platformy Entity Framework 6, deweloperzy mają możliwość wywoływania wywoływanie metody AddRange i RemoveRange bezpośrednio na DbSet, zamiast Iterowanie w kolekcji i wywoływanie Dodaj jeden raz dla każdego wystąpienia. Zaletą używania metody range jest koszt metody DetectChanges tylko raz płatnych dla całego zestawu jednostek, a nie raz dla każdej jednostki dodano.

3.2 buforowanie planu zapytania za pomocą

Zapytanie jest wykonywane, po raz pierwszy, go przechodzi przez kompilator wewnętrzny plan do tłumaczenia koncepcyjny zapytanie na polecenia magazynu (na przykład T-SQL, który jest wykonywany po uruchomieniu testów programu SQL Server). Jeśli włączone jest buforowanie planu zapytania, przy następnym zapytanie jest wykonywane sklepu polecenia są pobierane bezpośrednio z pamięci podręcznej planu zapytania do wykonania, z pominięciem kompilatora planu.

Pamięci podręcznej planu zapytania jest współużytkowany przez obiekt ObjectContext wystąpienia w ramach tej samej domen aplikacji. Nie należy przechowywać na wystąpienie obiektu ObjectContext do korzystania z buforowanie planu zapytania.

3.2.1 kilka uwag dotyczących buforowanie planu zapytania

- Pamięci podręcznej planu zapytania jest udostępniany dla wszystkie typy zapytań: Entity SQL, składnik LINQ to Entities i CompiledQuery obiektów.
- Domyślnie buforowanie planu zapytania jest włączona dla zapytań jednostki SQL, czy wykonywane za pośrednictwem EntityCommand lub ObjectQuery. Jego jest również domyślnie włączone dla programu LINQ do zapytań jednostki w Entity Framework w .NET 4.5 i Entity Framework 6
 - Buforowanie planu zapytania, może być wyłączone przez ustawienie wartości false dla właściwości EnablePlanCaching (na EntityCommand lub ObjectQuery). Na przykład:

```

var query = from customer in context.Customer
            where customer.CustomerId == id
            select new
            {
                customer.CustomerId,
                customer.Name
            };
ObjectQuery oQuery = query as ObjectQuery;
oQuery.EnablePlanCaching = false;

```

- W zapytaniach parametrycznych zmiana wartości parametru nadal będzie trafiać w pamięci podręcznej zapytań. Jednak zmiana parametru aspektami (na przykład rozmiaru, dokładności lub skali) spowoduje osiągnięcie innego wpisu w pamięci podręcznej.
- Podczas korzystania z jednostki SQL, ciąg zapytania jest częścią klucza. Zmiana zapytanie na wszystkich spowoduje wpisy w pamięci podręcznej różne, nawet jeśli zapytania są funkcjonalnie równoważne. Obejmuje to zmiany wielkości liter lub była białym znakiem.
- Podczas korzystania z LINQ, zapytania są przetwarzane do generowania część klucza. Zmiana wyrażenia LINQ w związku z tym wygeneruje inny klucz.
- Inne ograniczenia techniczne mogą zastosować; Aby uzyskać więcej informacji, zobacz Autocompiled zapytania.

3.2.2 algorytm eksmisji pamięci podręcznej

Zrozumienie, jak działa wewnętrznego algorytmu pomoże Ci zorientować się, aby włączyć lub wyłączyć buforowanie planu zapytania. Algorytm oczyszczania jest w następujący sposób:

1. Gdy pamięć podręczna zawiera określona liczba wpisów (800), na początek czasomierz okresowo (jeden raz na minutę) wrzucając pamięć podręcznej.
2. W trakcie symulacji pamięci podręcznej wpisy są usuwane z pamięci podręcznej na LFRU (najmniej często — ostatnio używane) podstawy. Ten algorytm uwzględnia liczbę trafeń i wieku przy podejmowaniu decyzji, które wpisy są odrzucane.
3. Na koniec każdego czyszczenia pamięci podręcznej pamięć podręczna zawiera ponownie 800 wpisów.

Wszystkie wpisy pamięci podręcznej są traktowani jednakowo podczas ustalania, które wpisy do wykluczenia. Oznacza to, że polecenie magazynu dla CompiledQuery ma ten sam prawdopodobieństwo eksmisji jako polecenie magazynu zapytania SQL jednostki.

Należy zauważyć, że czasomierza eksmisji pamięci podręcznej rozpocznie się w przypadku 800 jednostkami w pamięci podręcznej, ale w pamięci podręcznej tylko przechwytywana 60 sekund, po uruchomieniu tego czasomierza. Oznacza to, że do 60 sekund pamięci podręcznej może rosnąć dość duży.

3.2.3 test metryki ukazujące planu zapytania, buforowanie wydajności

Aby zaprezentować efekt planu zapytania, buforowanie na wydajność aplikacji, wykonane testu których firma Microsoft wykonywane liczby zapytań SQL jednostki w modelu Navision. Zobacz dodatek opis modelu Navision i typów kwerend, które zostały wykonane. W tym teście możemy najpierw iteracji przez listę zapytań i wykonywane każdego z nich raz, aby dodać je do pamięci podręcznej (jeśli jest włączone buforowanie). Ten krok jest untimed. Następnie możemy uśpienia wątku głównego ponad 60 sekund umożliwić buforowanie sprawdzaniu została wykonana; na koniec możemy wykonać iterację czasu listy 2 do wykonywania zapytań pamięci podręcznej. Ponadto on pamięci podręcznej planu programu SQL Server jest opróżniany przed wykonaniem każdego zestawu zapytań, aby przypadków, gdy uzyskany dokładnie odzwierciedla korzyści przez pamięć podręczną planu zapytań.

3.2.3.1 wyniki testu

TEST	EF5 BRAK PAMIĘCI PODRĘCZNEJ	EF5 PAMIĘCI PODRĘCZNEJ	EF6 BRAK PAMIĘCI PODRĘCZNEJ	EF6 PAMIĘCI PODRĘCZNEJ
Wyliczanie wszystkich zapytań 18723	124	125.4	124.3	125.3

TEST	EF5 BRAK PAMIĘCI PODRĘCZNEJ	EF5 PAMIĘCI PODRĘCZNEJ	EF6 BRAK PAMIĘCI PODRĘCZNEJ	EF6 PAMIĘCI PODRĘCZNEJ
Unikanie odchylenia (tylko pierwszy 800 zapytań, niezależnie od tego, co do złożoności)	41.7	5.5	40.5	5.4
Po prostu zapytania AggregatingSubtotals (178 razem — w celu uniknięcia odchylenia)	39.5	4.5	38.1	4.6

Cały czas w sekundach.

Autorskie — w przypadku wykonywania wiele różnych zapytań (na przykład tworzone dynamicznie zapytań), buforowania nie pomoże, a wynikowy opróżniania pamięci podręcznej można zachować zapytań, które używającym najbardziej buforowanie planu faktyczne korzystanie z niego.

Zapytania AggregatingSubtotals są najbardziej złożonych zapytań, które przetestowaliśmy za pomocą. Zgodnie z oczekiwaniami, tym bardziej złożone jest zapytanie, więcej korzyści, zobaczysz ze buforowanie planu zapytania.

Ponieważ CompiledQuery jest naprawdę zapytania LINQ z jego planem pamięci podręcznej, porównanie CompiledQuery a równoważne zapytań jednostki SQL powinna mieć podobne wyniki. W rzeczywistości Jeśli aplikacja ma wiele zapytań jednostki SQL dynamicznych, wypełnienie pamięci podręcznej za pomocą zapytań również skutecznie spowoduje CompiledQueries "dekompilować", gdy są one opróżniane z pamięci podręcznej. W tym scenariuszu można poprawić wydajność, wyłączenie buforowania na zapytania dynamiczne, aby określić priorytety CompiledQueries. Jeszcze lepiej oczywiście, abyśmy ponownego zapisywania aplikacji Używanie zapytań sparametryzowanych zamiast zapytań dynamicznych.

3.3 za pomocą CompiledQuery poprawianie wydajności za pomocą zapytań LINQ

Nasze testy wykażą, że za pomocą CompiledQuery może przynieść korzyści % 7 za pośrednictwem autocompiled zapytań LINQ; oznacza to, że musisz wykonać pewne czynności 7% mniej czasu na wykonywanie kodu ze stosu Entity Framework; nie oznacza to, że Twój aplikacja będzie 7% szybciej. Ogólnie rzecz biorąc koszt napisaniem i obsługą CompiledQuery obiektów w programie EF 5.0 może nie być warte problemy w porównaniu do korzyści. Przebieg może się różnić w, więc wykonuje tę opcję, jeśli Twój projekt wymaga dodatkowego wypychania. Należy pamiętać, że CompiledQueries tylko są zgodne z klasy pochodnej ObjectContext modeli i nie jest zgodna z modelami pochodzi od typu DbContext.

Aby uzyskać więcej informacji na temat tworzenia i wywoływania CompiledQuery, zobacz [zapytania skompilowane \(LINQ to Entities\)](#).

Istnieją dwie kwestie, które należy wykonać, korzystając z CompiledQuery, a mianowicie wymóg dotyczący używania statycznych wystąpień oraz problemy, że zawierają dzięki możliwości tworzenia. W tym miejscu poniżej szczegółowe wyjaśnienie tych dwóch zagadnień.

3.3.1 używać statycznych wystąpień CompiledQuery

Ponieważ kompilowanie zapytania LINQ jest czasochłonne, nie chcemy zrobić to za każdym razem, gdy będziemy musieli pobierać dane z bazy danych. Wystąpienia CompiledQuery pozwalają na raz skompilować i uruchomić wiele razy, ale trzeba należy zachować ostrożność i nabywania ponownego używania tego samego wystąpienia CompiledQuery za każdym razem, zamiast wielokrotnie zestawiania. Konieczność użycia statyczne elementy członkowskie do przechowywania wystąpień CompiledQuery; w przeciwnym razie nie będziesz widzieć żadnych korzyści.

Na przykład założymy, że Twoja strona zawiera następujące treści metody do obsługi wyświetlania produktów dla wybranej kategorii:

```

// Warning: this is the wrong way of using CompiledQuery
using (NorthwindEntities context = new NorthwindEntities())
{
    string selectedCategory = this.categoriesList.SelectedValue;

    var productsForCategory = CompiledQuery.Compile<NorthwindEntities, string, IQueryable<Product>>(
        (NorthwindEntities nwnd, string category) =>
        nwnd.Products.Where(p => p.Category.CategoryName == category)
    );

    this.productsGrid.DataSource = productsForCategory.Invoke(context, selectedCategory).ToList();
    this.productsGrid.DataBind();
}

this.productsGrid.Visible = true;

```

W takim przypadku utworzysz nowe wystąpienie CompiledQuery na bieżąco za każdym razem, gdy metoda jest wywoływana. Zamiast zobaczyć korzyści wydajności, pobierając polecenie magazynu z pamięci podręcznej planu zapytania, CompiledQuery będzie przejście przez kompilator planu, za każdym razem, gdy tworzone jest nowe wystąpienie. W rzeczywistości można będzie mieć zanieczyszczenie pamięci podręcznej planu zapytania z nowym wpisem CompiledQuery za każdym razem, gdy metoda jest wywoływana.

Zamiast tego chcesz utworzyć wystąpienie statyczne kompilowanym zapytaniu, więc wywoływanie tego samego zapytania skompilowane za każdym razem, gdy metoda jest wywoywana. Jednym ze sposobów to przez dodanie wystąpienia CompiledQuery jako członek kontekstu obiektów. Następnie można wprowadzić rzeczy nieco testu czyszczenia po zalogowaniu się za pośrednictwem metody pomocnika do CompiledQuery:

```

public partial class NorthwindEntities : ObjectContext
{
    private static readonly Func<NorthwindEntities, string, IEnumerable<Product>> productsForCategoryCQ =
        CompiledQuery.Compile(
            (NorthwindEntities context, string categoryName) =>
            context.Products.Where(p => p.Category.CategoryName == categoryName)
        );

    public IEnumerable<Product> GetProductsForCategory(string categoryName)
    {
        return productsForCategoryCQ.Invoke(this, categoryName).ToList();
    }
}

```

Ta metoda pomocnika będzie można wywołać w następujący sposób:

```
this.productsGrid.DataSource = context.GetProductsForCategory(selectedCategory);
```

3.3.2 redagowania za pośrednictwem CompiledQuery

Możliwość tworzenia za pośrednictwem dowolnego zapytania LINQ jest niezwykle przydatna funkcja; Aby to zrobić, możesz po prostu wywołać metodę po element IQueryble takich jak *Skip()* lub *Count()*. Jednak sposób więc zasadniczo zwraca nowy obiekt IQueryble. Nie ma nic do uniemożliwić Ci z technicznego punktu widzenia redagowania za pośrednictwem CompiledQuery, w ten sposób spowoduje, że Generowanie nowego obiektu IQueryble, wymaga przechodzącego przez kompilator planu ponownie.

Spowoduje, że niektóre składniki użytkowania IQueryble złożone obiekty, aby włączyć zaawansowane funkcje. Na przykład, ASP. GridView NET firmy może być powiązane z danymi do obiektu IQueryble za pomocą właściwości metody SelectMethod. Kontrolki GridView zostanie następnie tworzą za pośrednictwem tego obiektu IQueryble umożliwia sortowanie i stronicowanie za pośrednictwem modelu danych. Jak widać, za pomocą CompiledQuery dla widoku GridView nie osiągnie kompilowanym zapytaniu, ale wygeneruje nowe zapytanie autocompiled.

Zespół Doradczy klientów to omówiono w nim ich "Potencjalnych wydajności problemy z skompilowany LINQ

zapytanie ponownie komplikuje" w blogu: <http://blogs.msdn.com/b/appfabriccat/archive/2010/08/06/potential-performance-issues-with-compiled-linq-query-re-compiles.aspx>.

Jedno miejsce, gdzie może wystąpić ten jest podczas dodawania filtrów stopniowego do zapytania. Na przykład założmy, że masz strony klienci z kilku list rozwijanych opcjonalne filtry (na przykład, kraj i OrdersCount). Filtry te można utworzyć za pośrednictwem wyników IQueryables CompiledQuery, ale takie działanie spowoduje w nowym zapytaniu przechodzenia przez kompilator planu, za każdym razem, aby uruchomić go.

```
using (NorthwindEntities context = new NorthwindEntities())
{
    IQueryable<Customer> myCustomers = context.InvokeCustomersForEmployee();

    if (this.orderCountFilterList.SelectedItem.Value != defaultFilterText)
    {
        int orderCount = int.Parse(orderCountFilterList.SelectedValue);
        myCustomers = myCustomers.Where(c => c.Orders.Count > orderCount);
    }

    if (this.countryFilterList.SelectedItem.Value != defaultFilterText)
    {
        myCustomers = myCustomers.Where(c => c.Address.Country == countryFilterList.SelectedValue);
    }

    this.customersGrid.DataSource = myCustomers;
    this.customersGrid.DataBind();
}
```

Aby uniknąć tego ponownej komplikacji, można ponownie napisać CompiledQuery uwzględnienie możliwych filtrów:

```
private static readonly Func<NorthwindEntities, int, int?, string, IQueryable<Customer>>
customersForEmployeeWithFiltersCQ = CompiledQuery.Compile(
    (NorthwindEntities context, int empId, int? countFilter, string countryFilter) =>
    context.Customers.Where(c => c.Orders.Any(o => o.EmployeeID == empId))
    .Where(c => countFilter.HasValue == false || c.Orders.Count > countFilter)
    .Where(c => countryFilter == null || c.Address.Country == countryFilter)
);
```

Którego będzie można wywołać w interfejsie użytkownika, takich jak:

```
using (NorthwindEntities context = new NorthwindEntities())
{
    int? countFilter = (this.orderCountFilterList.SelectedIndex == 0) ?
        (int?)null :
        int.Parse(this.orderCountFilterList.SelectedValue);

    string countryFilter = (this.countryFilterList.SelectedIndex == 0) ?
        null :
        this.countryFilterList.SelectedValue;

    IQueryable<Customer> myCustomers = context.InvokeCustomersForEmployeeWithFilters(
        countFilter, countryFilter);

    this.customersGrid.DataSource = myCustomers;
    this.customersGrid.DataBind();
}
```

Kosztem w tym miejscu to polecenie wygenerowanego magazynu będzie ono mieć zawsze filtry z sprawdzanie wartości null, ale powinny być stosunkowo proste dla serwera bazy danych w celu optymalizacji:

```
...
WHERE ((@0 = (CASE WHEN (@p_linq_1 IS NOT NULL) THEN cast(1 as bit) WHEN (@p_linq_1 IS NULL) THEN cast(0 as bit) END)) OR ([Project3].[C2] > @p_linq_2)) AND (@p_linq_3 IS NULL OR [Project3].[Country] = @p_linq_4)
```

3.4 buforowanie metadanych

Entity Framework obsługuje także buforowanie metadanych. To jest zasadniczo buforowanie informacji o typie i informacje dotyczące mapowania typu w bazie danych w różnych połączeniach do tego samego modelu. Pamięć podręczna metadanych jest unikatowa dla każdej domeny aplikacji.

3.4.1 pamięć podręczna metadanych algorytmu

1. Informacje o metadanych dla modelu, znajduje się w obiekcie ItemCollection każdy obiekt EntityConnection.
 - Jako notatka boczna istnieją różne obiekty ItemCollection dla różnych części modelu. Na przykład StoreItemCollections zawiera informacje o modelu bazy danych; ObjectItemCollection zawiera informacje o modelu danych; EdmItemCollection zawiera informacje o modelu koncepcyjnego.
2. Jeśli dwa połączenia używają tych samych parametrów połączenia, będą miały to samo wystąpienie ItemCollection.
3. Parametry połączenia funkcjonalnie równoważne, ale różnych w formie tekstu może spowodować innych metadanych w pamięci podręcznej. Firma Microsoft tokenizację parametry połączenia, więc po prostu zmieniając kolejność tokenów powinna być rozwiązywana WE udostępnionych metadanych. Jednak dwa ciągi połączeń, które wydają się funkcjonalne może nie zostać ocenione jako identyczne po tokenizacji.
4. ItemCollection jest okresowo sprawdzane pod kątem użycia. Jeśli okaże się, że obszar roboczy nie uzyska dostępu niedawno, zostanie ona oznaczona na oczyszczenie na następny czyszczania pamięci podręcznej.
5. Jedynie tworzenie EntityConnection spowoduje, że pamięć podręczna metadanych, ma zostać utworzony (chociaż kolekcji elementów, które w nim nie zostaną zainicjowane, dopóki nie jest otwarte połączenie). Ten obszar roboczy pozostanie w pamięci, dopóki buforowania algorytm okaże się, że nie jest "w użyciu".

Zespół Doradczy klientów zapisane wpis w blogu, który opisuje zawierający odwołanie do obiektu ItemCollection w celu uniknięcia "wycofywania", korzystając z dużych modeli:

<<http://blogs.msdn.com/b/appfabriccat/archive/2010/10/22/metadataworkspace-reference-in-wcf-services.aspx>> .

3.4.2 relacji między buforowanie metadanych i buforowanie planu zapytania

Wystąpienie pamięci podręcznej planu zapytania, znajduje się w obiekcie MetadataWorkspace ItemCollection typów magazynu. Oznacza to, że polecenia magazynu pamięci podręcznej stosowanych w odniesieniu do zapytań dla dowolnego kontekstu tworzone przy użyciu danego obiektu MetadataWorkspace. Oznacza to również, że jeśli masz dwa ciągi połączeń są nieco inne, które nie są zgodne po tokenizowaniu, użytkownik będzie miał inne zapytanie, planowanie wystąpienia pamięci podręcznej.

3.5 wyniki buforowania

Z wynikami buforowania (znany także jako "second-level buforowanie") należy dysponować wyniki zapytań w lokalnej pamięci podręcznej. Wydając kwerendę, najpierw zobaczysz przypadku wyniki są dostępne lokalnie przed zapytaniem względem magazynu. Gdy wyniki z pamięci podręcznej nie są bezpośrednio obsługiwane przez program Entity Framework, jest możliwość dodania drugiego poziomu pamięci podręcznej przy użyciu dostawcy zawijania. Przykład dostawcy zawijania z pamięcią podręczną drugiego poziomu jest Alachisoft firmy [Entity Framework drugi poziom Cache oparta na NCache](#).

Ta implementacja pamięć podręczna drugiego poziomu jest wprowadzony funkcje, które odbywa się po ocenie wyrażenie LINQ (i funcletized) i planu wykonywania zapytania jest obliczana lub pobrane z pierwszego poziomu pamięci podręcznej. Pamięć podręczna drugiego poziomu następnie zapisze tylko wyniki pierwotne bazy danych, dlatego potok materializacja nadal wykonuje później.

3.5.1 dodatkowe informacje dotyczące wyników z pamięci podręcznej za pomocą dostawcy zawijania

- Julie Lerman został zapisany w artykule MSDN "Second-Level buforowania w Entity Framework i Windows Azure", o tym, jak można zaktualizować dostawcy zawijania przykładowe pamięci podrzcznej programu AppFabric systemu Windows Server: <https://msdn.microsoft.com/magazine/hh394143.aspx>
- Jeśli pracujesz z Entity Framework 5, blog zespołu ma wpis, w której opisano Rozpoczynanie pracy z pamięci podrzcznej dostawcy programu Entity Framework 5:
[<http://blogs.msdn.com/b/adonet/archive/2010/09/13/ef-caching-with-jarek-kowalski-s-provider.aspx>](http://blogs.msdn.com/b/adonet/archive/2010/09/13/ef-caching-with-jarek-kowalski-s-provider.aspx). Zawiera on również szablon T4, które ułatwiają Automatyzowanie dodanie 2. buforowanie na poziomie projektu.

4 Autocompiled zapytań

Podczas generowania zapytania względem bazy danych przy użyciu platformy Entity Framework, jego musi ona przejść serię kroków przed faktycznie materializowanie wyniki. jeden taki krok nie jest kompilowanie zapytania. Znanych zapytań jednostki SQL ma dobrą wydajność, jak automatycznie są buforowane, więc drugi lub trzeci czas wykonania tego samego zapytania, można pominąć kompilatora planu i zamiast tego użyj buforowanego planu.

Entity Framework 5 wprowadzono buforowania automatycznego dla programu LINQ do zapytań jednostki także. W poprzednich wersjach programu Entity Framework CompiledQuery, aby przyspieszyć tworzenie wydajność była powszechną praktyką dzięki temu upewnisz się LINQ do kwerendy jednostek podlega buforowaniu. Ponieważ buforowanie teraz odbywa się automatycznie bez użycia CompiledQuery, nazywamy tę funkcję "autocompiled zapytania". Aby uzyskać więcej informacji o pamięci podrzcznej planu zapytania i jego mechanics Zobacz buforowanie planu zapytania.

Wykrywa platformy Entity Framework, gdy zapytanie wymaga, aby ponownie skompilowana, a nie tak, gdy zapytanie jest wywoływana, nawet wtedy, gdy miała zostać skompilowany przed. Typowe warunki, które powodują zapytania do ponownej komplikacji są:

- Zmiana MergeOption skojarzonej z zapytaniem. Pamięci podrzcznej zapytania nie będą używane, zamiast tego kompilator plan zostaną ponownie uruchomione i nowo utworzonego planu pobiera buforowany.
- Zmiana wartości ContextOptions.UseCSharpNullComparisonBehavior. Możesz uzyskać ten sam efekt jak zmiana MergeOption.

Inne warunki może uniemożliwić korzystanie z pamięci podrzcznej przez zapytanie. Typowe przykłady to:

- Za pomocą interfejsu `IEnumerable<T>`. Zawiera<`T`>(wartość `T`).
- Za pomocą funkcji, które generują zapytania za pomocą stałych.
- Korzystanie z właściwości obiektu nie jest zamapowany.
- Łączenie zapytania do innego zapytania, który wymaga, aby ponownie skompilowana.

4.1 przy użyciu interfejsu `IEnumerable<T>`. Zawiera<`T`>(wartość `T`)

Entity Framework, nie będzie buforować zapytań, które wywołują `IEnumerable<T>`. Zawiera<`T`>(wartość `T`) względem kolekcji w pamięci, ponieważ wartości kolekcji są traktowane jako nietrwałe. Poniższe przykładowe zapytanie nie będzie zapisywane, dzięki czemu będzie on przetworzony przez kompilator plan:

```
int[] ids = new int[10000];
...
using (var context = new MyContext())
{
    var query = context.MyEntities
        .Where(entity => ids.Contains(entity.Id));

    var results = query.ToList();
    ...
}
```

Należy zauważyć, że rozmiar `IEnumerable`, względem której zawiera jest wykonywane zapytanie określa, jak

szynko lub wolno jest kompilowana. Może to spowodować obniżenie wydajności znacznie korzystając z dużych kolekcjach, takiego jak pokazano w powyższym przykładzie.

Entity Framework 6 zawiera optymalizacje w sposobie `IEnumerable<T>`. Zawiera`<T>`(wartość `T`) działa, gdy zapytania są wykonywane. Kod SQL, który jest generowany jest znacznie szybszy, aby wygenerować i bardziej czytelny i w większości przypadków jest również wykonywana szybciej na serwerze.

4.2 przy użyciu funkcji, które generują zapytania za pomocą stałych

Operatory `Skip()`, `Take()`, `Contains()` i `DefaultIfEmpty()` LINQ nie tworzą zapytania SQL z parametrami, ale zamiast tego umieść wartości przekazane do nich jako stałe. W związku z tym zapytań, które w przeciwnym razie mogły być identyczne zakończą się zanieczyszczenie zapytanie plan pamięci podręcznej, zarówno na stosie EF, jak i na serwerze bazy danych, a nie uzyskać reutilized, chyba że tych samych stałych są używane podczas wykonywania kolejnych zapytań. Na przykład:

```
var id = 10;
...
using (var context = new MyContext())
{
    var query = context.MyEntities.Select(entity => entity.Id).Contains(id);

    var results = query.ToList();
    ...
}
```

W tym przykładzie każdym razem, gdy to zapytanie jest wykonywane, podając inną wartość dla identyfikatora kwerendy zostanie skompilowany w nowym planie.

W szczególności zwróć uwagę na korzystanie z `Skip` i `Take` podczas ustalania stronicowania. W EF6 metody te mają przeciążenia lambda, które skutecznie sprawia, że plan pamięci podręcznej zapytań do wielokrotnego użytku ponieważ EF można przechwytywać zmienne przekazywane do tych metod i tłumaczyć je na `SQLparameters`. Pomaga to również zachowywać pamięci podręcznej bardziej przejrzysty, ponieważ w przeciwnym razie każdego zapytania z inną stałą `Skip` i `Take` otrzymamy swój własny wpis pamięci podręcznej planu zapytania.

Należy wziąć pod uwagę następujący kod, który jest nieoptymalne, ale jest przeznaczone wyłącznie do spróbowujemy tej klasy zapytania:

```
var customers = context.Customers.OrderBy(c => c.LastName);
for (var i = 0; i < count; ++i)
{
    var currentCustomer = customers.Skip(i).FirstOrDefault();
    ProcessCustomer(currentCustomer);
}
```

Szybsze wersję tego samego kodu obejmowałaby wywoływanie `Pomiń` z wyrażenia lambda:

```
var customers = context.Customers.OrderBy(c => c.LastName);
for (var i = 0; i < count; ++i)
{
    var currentCustomer = customers.Skip(() => i).FirstOrDefault();
    ProcessCustomer(currentCustomer);
}
```

Drugi fragment kodu może działać 11% szybciej, ponieważ jest używany ten sam plan zapytania, za każdym razem, gdy zapytanie jest uruchomione, pozwala zaoszczędzić czas procesora CPU, co pozwala uniknąć zanieczyszczenie pamięci podręczną zapytań. Ponadto ponieważ parametru do pomijania jest zamknięcie kod także wygląda to teraz:

```
var i = 0;
var skippyCustomers = context.Customers.OrderBy(c => c.LastName).Skip(() => i);
for (; i < count; ++i)
{
    var currentCustomer = skippyCustomers.FirstOrDefault();
    ProcessCustomer(currentCustomer);
}
```

4.3 przy użyciu właściwości obiektu bez zamapowane

Podczas zapytania jest używana właściwości typu-zamapowany obiekt jako parametr, a następnie zapytanie będzie nie Pobieranie pamięci podręcznej. Na przykład:

```
using (var context = new MyContext())
{
    var myObject = new NonMappedType();

    var query = from entity in context.MyEntities
                where entity.Name.StartsWith(myObject.MyProperty)
                select entity;

    var results = query.ToList();
    ...
}
```

W tym przykładzie założono, że klasa NonMappedType nie jest częścią modelu jednostki. To zapytanie można łatwo zmienić nie używając typu nie są mapowane, i zamiast tego użyć zmiennej lokalnej jako parametru zapytania:

```
using (var context = new MyContext())
{
    var myObject = new NonMappedType();
    var myValue = myObject.MyProperty;
    var query = from entity in context.MyEntities
                where entity.Name.StartsWith(myValue)
                select entity;

    var results = query.ToList();
    ...
}
```

W tym przypadku zapytania będą mogli uzyskać pamięci podręcznej i będą mogli korzystać z pamięci podręcznej planu zapytania.

4.4 — łączenie zapytań, które wymagają ponownej komplikacji

Tym samym przykładzie jak wyżej Jeśli masz drugiego zapytania, która opiera się na zapytaniach, który musi być ponownie komplikowane, cały drugiego zapytania będzie również ponownie komplikowana. Oto przykład, aby zilustrować ten scenariusz:

```

int[] ids = new int[10000];
...
using (var context = new MyContext())
{
    var firstQuery = from entity in context.MyEntities
                     where ids.Contains(entity.Id)
                     select entity;

    var secondQuery = from entity in context.MyEntities
                      where firstQuery.Any(otherEntity => otherEntity.Id == entity.Id)
                      select entity;

    var results = secondQuery.ToList();
    ...
}

```

W przykładzie przedstawiono ogólny, ale ilustruje, jak łączenie firstQuery powoduje secondQuery będą mogli uzyskać pamięci podręcznej. Jeśli firstQuery nie była kwerendą, która wymaga ponownej komplikacji, następnie secondQuery mogłoby być buforowane.

Zapytania NoTracking 5

5.1 wyłączenie śledzenia zmian, aby zmniejszyć koszty zarządzania stanu

Jeśli jesteś w scenariuszu tylko do odczytu i chcesz uniknąć zadań ładowania obiektów w obiekcie ObjectStateManager, możesz odpytywać "Bez śledzenia". Można wyłączyć śledzenia zmian na poziomie zapytania.

Pamiętaj jednak, że, wyłączając możesz śledzenia zmian są efektywne wyłączenie pamięci podręcznej obiektów. Po wykonaniu zapytania dotyczącego jednostki, firma Microsoft nie można pominąć materializacja przez pobieranie wyników zapytania wcześniej zmaterializowanego w obiekcie ObjectStateManager. Jeśli są wielokrotnie wykonywaniem zapytań dotyczących tych samych jednostek na tym samym kontekście, można napotkać faktycznie wydajności korzyść z wyłączenia śledzenia zmian.

Podczas wykonywania zapytania za pomocą obiektu ObjectContext, gdy jest ustawiona i zapytania, które składają się na nich będzie dziedziczyć skuteczne MergeOption zapytania nadzawanego wystąpienia ObjectQuery i obiektu ObjectSet zapamięta MergeOption. Korzystając z typu DbContext, wywołując modyfikator AsNoTracking() na DbSet można wyłączyć śledzenia.

5.1.1 wyłączenie śledzenia zmian dla zapytania przy użyciu typu DbContext

Aby przełączyć tryb zapytania na NoTracking, łańcuch wywołanie metody AsNoTracking() w zapytaniu. W odróżnieniu od ObjectQuery DbSet i DbQuery klasy w interfejsie API DbContext braku modyfikowalną właściwość MergeOption.

```

var productsForCategory = from p in context.Products.AsNoTracking()
                           where p.Category.CategoryName == selectedCategory
                           select p;

```

5.1.2 wyłączenie śledzenia na poziomie zapytania, przy użyciu obiektu ObjectContext zmian

```

var productsForCategory = from p in context.Products
                           where p.Category.CategoryName == selectedCategory
                           select p;

((ObjectQuery)productsForCategory).MergeOption = MergeOption.NoTracking;

```

5.1.3 wyłączenie śledzenia zmian dla całej jednostki można ustawić przy użyciu obiektu ObjectContext

```

context.Products.MergeOption = MergeOption.NoTracking;

var productsForCategory = from p in context.Products
                           where p.Category.CategoryName == selectedCategory
                           select p;

```

5.2 metryki testu ukazujące korzyści w zakresie wydajności kwerend NoTracking

W tym teście spojrzymy kosztem wypełnianie obiekt ObjectStateManager porównując śledzenia zapytań NoTracking Navision modelu. Zobacz dodatek opis modelu Navision i typów kwerend, które zostały wykonane. W tym teście możemy iteracji przez listę zapytań i wykonać jeden raz każdego z nich. Uruchomiliśmy dwie odmiany testu, drugi raz z NoTracking zapytań i jeden raz z domyślną opcję scalania "TylkoDołącz". Przeprowadziliśmy poszczególnych odmian 3 razy i wykonać wartości średniej przebiegów. Między testy możemy Wyczyść pamięć podręczną zapytań w programie SQL Server i zmniejszyć tempdb, uruchamiając następujące polecenia:

1. POLECENIE DBCC DROPCLEANBUFFERS
2. POLECENIE DBCC FREEPROCCACHE
3. DBCC SHRINKDATABASE (bazy danych tempdb, 0)

Wyniki, mediana ponad 3 przebiegów testów:

	NIE ŚLEDZENIA — ZESTAW ROBOCZY	BRAK ŚLEDZENIA — GODZINA	DOŁĄCZ DO TYLKO — ZESTAW ROBOCZY	DOŁĄCZ TYLKO — GODZINA
Entity Framework 5	460361728	1163536 ms	596545536	1273042 ms
Entity Framework 6	647127040	190228 ms	832798720	195521 ms

Entity Framework 5 mają mniejsze zużycie pamięci na koniec uruchom niż Entity Framework 6. Dodatkowej pamięci używane przez program Entity Framework 6 jest wynikiem struktur więcej pamięci i kod, który umożliwia deweloperom nowe funkcje i lepszą wydajność.

Istnieje również wyczyść różnicą w zużycie pamięci, korzystając z obiektu ObjectStateManager. Gdy rejestrowanie informacji o wszystkich jednostek, które firma Microsoft zmaterializowanego z bazy danych, platformy Entity Framework 5 zwiększyć jego rozmiaru o 30%. Entity Framework 6 zwiększyć jego rozmiaru, 28% sytuacji.

W czasie platformy Entity Framework 6 przewysza stosowane przekształcania Entity Framework 5 w tym teście przez duże margines. Entity Framework 6 test zakończył się w około 16% czasu używanego przez Entity Framework 5. Ponadto Entity Framework 5 czasochłonne 9% więcej ukończone, gdy jest używany obiekt ObjectStateManager. W odróżnieniu od platformy Entity Framework 6 używa więcej czasu, korzystając z obiekt ObjectStateManager % 3.

6 opcje wykonywania zapytań

Entity Framework oferuje kilka różnych sposobów, aby wykonać zapytanie. Firma Microsoft będzie zapoznaj się z następujących opcji, porównaj zalet i wad każdego z nich i sprawdzić ich charakterystyk wydajności:

- Składnik LINQ to Entities.
- Brak śledzenia LINQ to Entities.
- Jednostka SQL przez ObjectQuery.
- Jednostka SQL przez EntityCommand.
- ExecuteStoreQuery.
- SqlQuery.
- CompiledQuery.

6.1 zapytaniach składnika LINQ to Entities

```
var q = context.Products.Where(p => p.Category.CategoryName == "Beverages");
```

Specjałiści

- Odpowiedni dla operacje CUD.
- W pełni zmaterializowany obiektów.
- Najprostszą do zapisu przy użyciu składni wbudowane w języku programowania.
- Dobrą wydajność.

Wady

- Niektórych ograniczeń technicznych, takich jak:
 - Wzory DefaultIfEmpty zapytań OUTER JOIN powoduje bardziej złożone zapytania niż proste instrukcje OUTER JOIN w języku SQL jednostki.
 - Nadal nie można użyć NOTACJI z dopasowaniem wzorca ogólnego.

6.2. Brak śledzenia LINQ do zapytań jednostki

Gdy kontekstu pochodzi ObjectContext:

```
context.Products.MergeOption = MergeOption.NoTracking;  
var q = context.Products.Where(p => p.Category.CategoryName == "Beverages");
```

Gdy kontekstu pochodzi DbContext:

```
var q = context.Products.AsNoTracking()  
    .Where(p => p.Category.CategoryName == "Beverages");
```

Specjałiści

- Zwiększo wydajność przez regularne zapytań LINQ.
- W pełni zmaterializowany obiektów.
- Najprostszą do zapisu przy użyciu składni wbudowane w języku programowania.

Wady

- Nie nadaje się do operacje CUD.
- Niektórych ograniczeń technicznych, takich jak:
 - Wzory DefaultIfEmpty zapytań OUTER JOIN powoduje bardziej złożone zapytania niż proste instrukcje OUTER JOIN w języku SQL jednostki.
 - Nadal nie można użyć NOTACJI z dopasowaniem wzorca ogólnego.

Należy pamiętać, zapytania, które właściwości skalarne projektu nie są śledzone, nawet jeśli nie określono NoTracking. Na przykład:

```
var q = context.Products.Where(p => p.Category.CategoryName == "Beverages").Select(p => new { p.ProductName  
});
```

To określone zapytanie nie jawnie określone, jest NoTracking, ale ponieważ nie jest materializowanie typ, który ma znane przez menedżera stanu obiektu następnie zmaterializowanym wyniku nie jest śledzone.

6.3 jednostki SQL przez ObjectQuery

```
ObjectQuery<Product> products = context.Products.Where("it.Category.CategoryName = 'Beverages'");
```

Specjałiści

- Odpowiedni dla operacje CUD.
- W pełni zmaterializowany obiektów.
- Buforowanie planu zapytania obsługuje.

Wady

- Obejmuje ciągi zapytań tekstowych, które są bardziej podatne na błędy użytkowników niż konstrukcje zapytań wbudowane w języku.

6.4 jednostki SQL przez polecenie jednostki

```
EntityCommand cmd = eConn.CreateCommand();
cmd.CommandText = "Select p From NorthwindEntities.Products As p Where p.Category.CategoryName = 'Beverages'";

using (EntityDataReader reader = cmd.ExecuteReader(CommandBehavior.SequentialAccess))
{
    while (reader.Read())
    {
        // manually 'materialize' the product
    }
}
```

Specjałiści

- Obsługuje zapytania, buforowanie planu w programie .NET 4.0 (buforowanie planu jest obsługiwany przez wszystkie inne typy zapytań w .NET 4.5).

Wady

- Obejmuje ciągi zapytań tekstowych, które są bardziej podatne na błędy użytkowników niż konstrukcje zapytań wbudowane w języku.
- Nie nadaje się do operacje CUD.
- Wyniki nie są automatycznie zmaterializowany i musi być odczytywana z czytnika danych.

6.5 SqlQuery i ExecuteStoreQuery

SqlQuery w bazie danych:

```
// use this to obtain entities and not track them
var q1 = context.Database.SqlQuery<Product>("select * from products");
```

SqlQuery na DbSet:

```
// use this to obtain entities and have them tracked
var q2 = context.Products.SqlQuery("select * from products");
```

ExecuteStoreQuery:

```

var beverages = context.ExecuteStoreQuery<Product>(
    @"      SELECT      P.ProductID, P.ProductName, P.SupplierID, P.CategoryID, P.QuantityPerUnit, P.UnitPrice,
P.UnitsInStock, P.UnitsOnOrder, P.ReorderLevel, P.Discontinued, P.DiscontinuedDate
        FROM          Products AS P INNER JOIN Categories AS C ON P.CategoryID = C.CategoryID
        WHERE         (C.CategoryName = 'Beverages')"
);

```

Specjałiści

- Ogólnie największą wydajność, ponieważ kompilator plan jest pomijany.
- W pełni zmaterializowany obiektów.
- Odpowiedni dla operacje CUD w przypadku używania z DbSet.

Wady

- Zapytanie jest tekstową i występowania błędów.
- Zapytanie jest powiązany określonych wewnętrznej bazy danych przy użyciu semantyki magazynu zamiast semantyki pojęć.
- W przypadku dziedziczenia jest obecny, handcrafted zapytania trzeba uwzględnić warunki mapowania żądanego typu.

6.6 CompiledQuery

```

private static readonly Func<NorthwindEntities, string, IQueryable<Product>> productsForCategoryCQ =
CompiledQuery.Compile(
    (NorthwindEntities context, string categoryName) =>
    context.Products.Where(p => p.Category.CategoryName == categoryName)
);
...
var q = context.InvokeProductsForCategoryCQ("Beverages");

```

Specjałiści

- Udostępnia do poprawy wydajności 7% w porównaniu z regularnych zapytań LINQ.
- W pełni zmaterializowany obiektów.
- Odpowiedni dla operacje CUD.

Wady

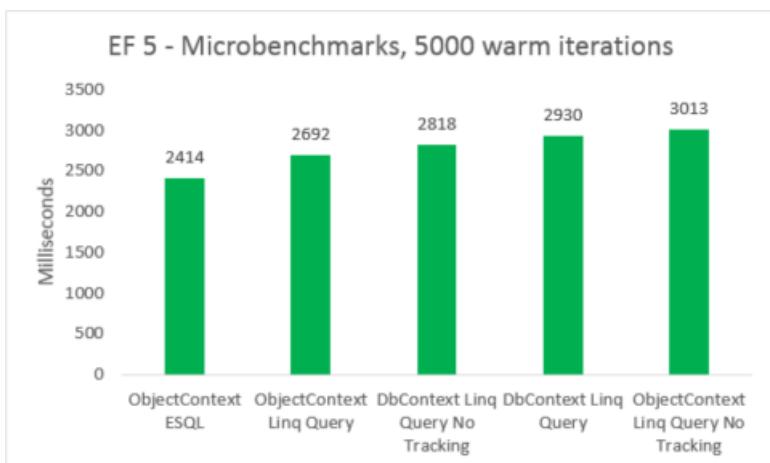
- Większą złożoność i koszty programowania.
- Zwiększenie wydajności zostaną utracone podczas redagowania na podstawie kompilowanym zapytaniu.
- Nie można zapisać niektórych zapytań LINQ jako CompiledQuery — na przykład projekcje typów anonimowych.

6.7 Porównanie wydajności opcji inne zapytanie

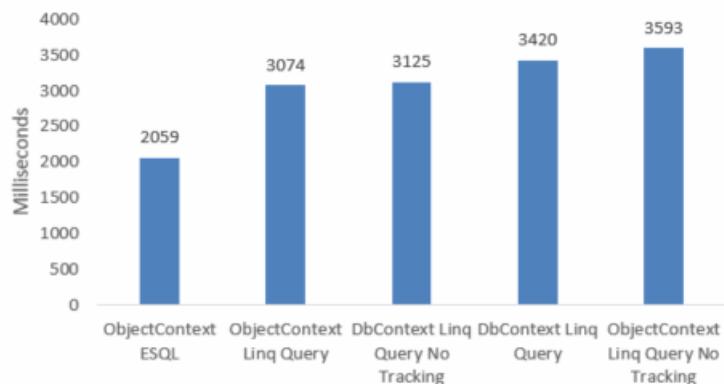
Proste microbenchmarks, której nie upłynął tworzenie kontekstu pojawiły się do testu. Firma Microsoft mierzy zapytań 5000 razy dla zestawu jednostek nie są buforowane w kontrolowanym środowisku. Numery te są pobierane z ostrzeżeniem: nie odzwierciedlają wartości rzeczywistych generowany przez aplikację, ale zamiast tego są one bardzo dokładny pomiar większość różnic w wydajności jest porównaniu różne opcje zapytań jabłka do jabłek, z wyłączeniem koszt utworzenia nowego kontekstu.

EF	TEST	CZAS (MS)	PAMIĘĆ
EF5	ObjectContext ESQL	2414	38801408

EF	TEST	CZAS (MS)	PAMIĘĆ
EF5	Zapytania Linq ObjectContext	2692	38277120
EF5	Linq typu DbContext zapytania nie śledzenia	2818	41840640
EF5	Zapytania Linq typu DbContext	2930	41771008
EF5	Linq do obiektu ObjectContext zapytania nie śledzenia	3013	38412288
EF6	ObjectContext ESQL	2059	46039040
EF6	Zapytania Linq ObjectContext	3074	45248512
EF6	Linq typu DbContext zapytania nie śledzenia	3125	47575040
EF6	Zapytania Linq typu DbContext	3420	47652864
EF6	Linq do obiektu ObjectContext zapytania nie śledzenia	3593	45260800



EF 6 - Microbenchmarks, 5000 warm iterations

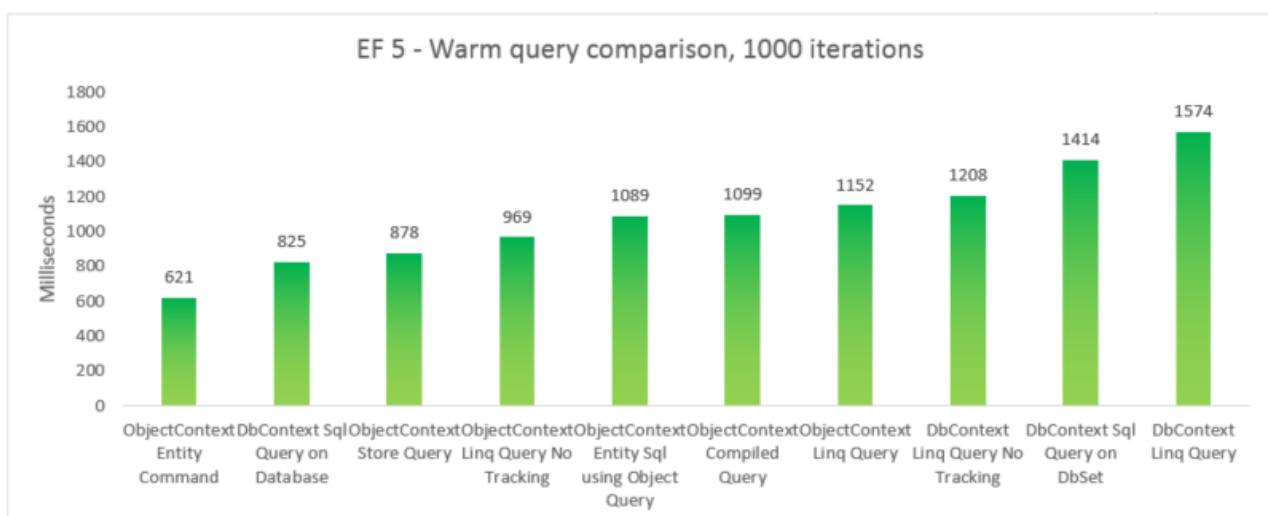


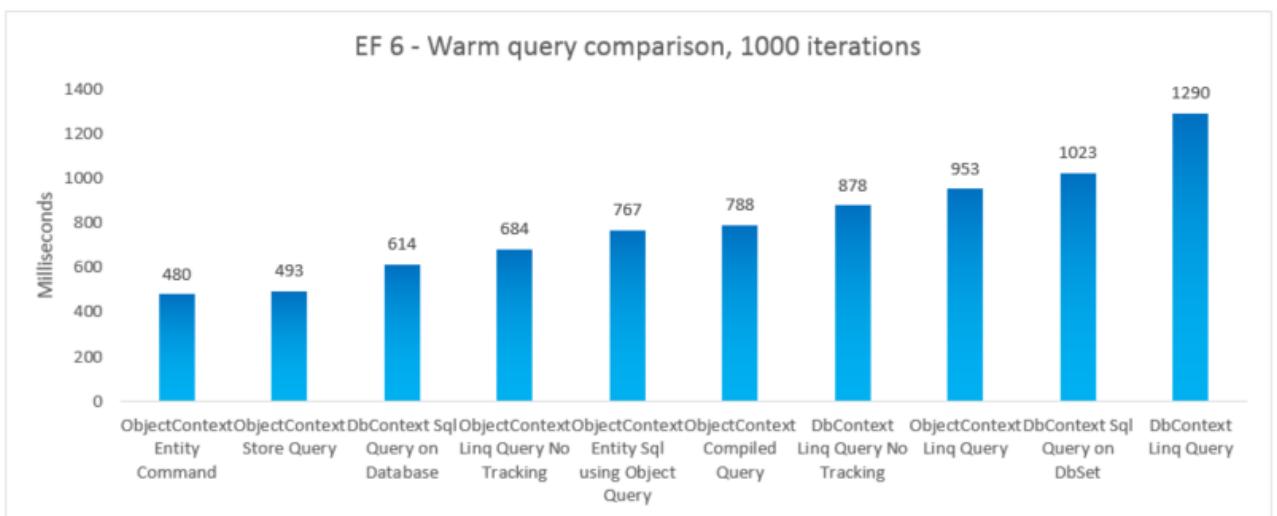
Microbenchmarks są bardzo wrażliwe na niewielkie zmiany w kodzie. W tym przypadku różnicy między kosztów Entity Framework 5 i Entity Framework 6 są spowodowane przez dodanie [przejmowanie i transakcyjnych ulepszenia](#). Te numery microbenchmarks są jednak namnożonego przetwarzania do bardzo małego fragmentu działania programu Entity Framework. Rzeczywiste scenariusze dostępu do ciepłych zapytania nie powinien zostać wyświetlony regresji wydajności podczas aktualniania programu Entity Framework 5 do programu Entity Framework 6.

Aby porównać wydajność rzeczywistych opcje inne zapytanie, utworzyliśmy 5 odmiany osobnego badania, gdzie możemy opcja inne zapytanie umożliwia wybór wszystkich produktów, których nazwa kategorii jest "Beverages". Każda iteracja obejmuje koszt tworzenia kontekstu, a koszt materializowanie wszystkie zwrócone jednostki. 10 iteracji są uruchamiane untimed przed przełączeniem sumę upłynął limit czasu 1000 iteracji. Wyniki wyświetlane są także jej medianą Uruchom pobranego z 5 uruchomienia każdego testu. Aby uzyskać więcej informacji zobacz dodatek B, który zawiera kod dla testu.

EF	TEST	CZAS (MS)	PAMIĘĆ
EF5	Polecenie ObjectContext jednostki	621	39350272
EF5	Kontekst DbContext zapytanie Sql w bazie danych	825	37519360
EF5	Query Store ObjectContext	878	39460864
EF5	Linq do obiektu ObjectContext zapytania nie śledzenia	969	38293504
EF5	Obiekt ObjectContext jednostki Sql za pomocą obiektu Query	1089	38981632
EF5	Obiekt ObjectContext komplikowanym zapytaniu.	1099	38682624
EF5	Zapytania Linq ObjectContext	1152	38178816
EF5	Linq typu DbContext zapytania nie śledzenia	1208	41803776

EF	TEST	CZAS (MS)	PAMIĘĆ
EF5	Zapytanie Sql DbContext na DbSet	1414	37982208
EF5	Zapytania Linq typu DbContext	1574	41738240
EF6	Polecenie ObjectContext jednostki	480	47247360
EF6	Query Store ObjectContext	493	46739456
EF6	Kontekst DbContext zapytanie Sql w bazie danych	614	41607168
EF6	Linq do obiektu ObjectContext zapytania nie śledzenia	684	46333952
EF6	Obiekt ObjectContext jednostki Sql za pomocą obiektu Query	767	48865280
EF6	Obiekt ObjectContext kompilowanym zapytaniu.	788	48467968
EF6	Linq typu DbContext zapytania nie śledzenia	878	47554560
EF6	Zapytania Linq ObjectContext	953	47632384
EF6	Zapytanie Sql DbContext na DbSet	1023	41992192
EF6	Zapytania Linq typu DbContext	1290	47529984





NOTE

Aby informacje były kompletne dodaliśmy odmiany, gdzie możemy wykonać kwerendy SQL jednostki EntityCommand. Jednak ponieważ wyniki nie są zmaterializowanego takich zapytań, porównanie niekoniecznie jabłek do jabłka. Test obejmuje bliskie zbliżenia do materializowanie próby porównywania bardziej sprawiedliwa.

W tym przypadku end-to-end Entity Framework 6 przewyższa stosowane przekształcania Entity Framework 5 ze względu na ulepszenia wydajności wprowadzone na kilka części stosu, w tym znacznie jaśniejszy inicjowania typu DbContext i szybsze MetadataCollection<T> wyszukiwania.

7 zagadnienia dotyczące wydajności czasu projektowania

7.1 strategie dziedziczenia

Inną ważną kwestią wydajności podczas korzystania z programu Entity Framework jest strategia dziedziczenia, których używasz. Entity Framework obsługuje 3 typy podstawowe dziedziczenia i ich kombinacje:

- Tabela na hierarchii (TPH) — gdzie każdy dziedziczenia zestaw mapy do tabeli z kolumną dyskryminatora, aby wskazać, które określonego typu w hierarchii jest reprezentowana w wierszu.
- Tabela według typu (TPT) — gdzie każdy typ ma własną tabelę w bazie danych. tabele podrzędne definiować tylko kolumny, które nie zawiera tabeli nadzędnej.
- Tabela na klasę (TPC) — gdzie każdy typ ma swój własny pełną tabelę w bazie danych. tabele podrzędne definiują ich pól, takich jak te zdefiniowane typów nadzędnych.

Jeśli model korzysta z dziedziczenia TPT, zapytania, które są generowane będzie bardziej skomplikowane niż te, które są generowane z innymi strategiami dziedziczenia, które mogą powstać w dłuższym czasie wykonywania w sklepie. Zwykle będzie trwać dłużej, do generowania zapytań za pośrednictwem modelu TPT i do zmaterializowania obiektów wynikowych.

Zobacz "zagadnienia dotyczące wydajności podczas korzystania z dziedziczenia TPT (Tabela na typ) w Entity Framework" w blogu MSDN: <<http://blogs.msdn.com/b/adonet/archive/2010/08/17/performance-considerations-when-using-tpt-table-per-type-inheritance-in-the-entity-framework.aspx>>.

7.1.1 unikanie TPT w aplikacjach pierwszego modelu lub Code First

Podczas tworzenia modelu przez istniejącą bazę danych, która ma schemat TPT nie masz wiele opcji. Jednak podczas tworzenia aplikacji przy użyciu modelu pierwszej lub Code First, należy unikać TPT dziedziczenia dla problemów z wydajnością.

Gdy używasz pierwszego modelu w Kreatorze Projektant jednostki, otrzymasz TPT wszystkie dziedziczenia w modelu. Jeśli chcesz przełączyć się do strategii TPH dziedziczenia z pierwszego modelu, można używać "jednostki projektanta bazy danych generowania Power Pack" dostępne z galerii Visual Studio (

<<http://visualstudiogallery.msdn.microsoft.com/df3541c3-d833-4b65-b942-989e7ec74c87/>>).

Z pomocą Code First skonfiguruj mapowanie modelu za pomocą dziedziczenia, platforma EF użyje TPH domyślnie, dlatego wszystkie jednostki w hierarchii dziedziczenia zostaną zmapowane do tej samej tabeli. Zobacz sekcję "Mapowanie z interfejs Fluent API" artykułu "Kodu pierwszy w jednostki Framework4.1" w MSDN Magazine (<http://msdn.microsoft.com/magazine/hh126815.aspx>) Aby uzyskać więcej informacji.

7.2 uaktualniasz EF4 w celu generowania modelu czasu

Ulepszanie specyficzne dla programu SQL Server do algorytmu, który generuje warstwę magazynu (SSDL) modelu jest dostępne w programie Entity Framework 5 i 6 i jako aktualizacja programu Entity Framework 4, po zainstalowaniu programu Visual Studio 2010 z dodatkiem SP1. Następujące wyniki testów pokazują ulepszanie podczas generowania modelu bardzo dużych, w tym przypadku modelu Navision. Aby uzyskać więcej informacji na ten temat, zobacz dodatku C.

W modelu danych zestawy jednostek 1005 4227 zestawów skojarzeń.

KONFIGURACJA	PODZIAŁ WYKORZYSTANY CZAS
Program Visual Studio 2010, platformy Entity Framework 4	Generowanie SSDL: 2 hr 27 min Generowanie mapowania: 1 sekundę Generowanie CSDL: 1 sekundę Generowanie ObjectLayer: 1 sekundę Generowanie widoku: 2 godz. 14 min
Visual Studio 2010 z dodatkiem SP1, platformy Entity Framework 4	Generowanie SSDL: 1 sekundę Generowanie mapowania: 1 sekundę Generowanie CSDL: 1 sekundę Generowanie ObjectLayer: 1 sekundę Generowanie widoku: 1 min 53 hr
Visual Studio 2013, platformy Entity Framework 5	Generowanie SSDL: 1 sekundę Generowanie mapowania: 1 sekundę Generowanie CSDL: 1 sekundę Generowanie ObjectLayer: 1 sekundę Generowanie widoku: 65 minut
Visual Studio 2013, platformy Entity Framework 6	Generowanie SSDL: 1 sekundę Generowanie mapowania: 1 sekundę Generowanie CSDL: 1 sekundę Generowanie ObjectLayer: 1 sekundę Generowanie widoku: 28 sekundach.

Warto zauważyć, że podczas generowania SSDL, obciążenia prawie całkowicie odbywa się na programu SQL Server, gdy oczekuje na komputerze deweloperskim klienta bezczynny wyniki, aby wrócić z serwera.

Przetwarzający należy szczególnie wdzięczni za to ulepszenie. Warto również zauważyć, że zasadniczo cały koszt generowania modelu odbywa się podczas generowania widoku teraz.

7.3 najpierw dzielenia dużych modeli z bazą danych i modelu pierwszy

W miarę zwiększania rozmiaru modelu powierzchni projektanta staje się przepełniony i trudne w użyciu. Firma Microsoft zwykle należy wziąć pod uwagę modelu z ponad 300 jednostek zbyt duży, aby skutecznie używać projektanta. Następujący wpis w blogu opisuje kilka opcji do dzielenia dużych modeli:

<<http://blogs.msdn.com/b/adonet/archive/2008/11/25/working-with-large-models-in-entity-framework-part-2.aspx>>.

Wpis został napisany dla pierwszej wersji programu Entity Framework, ale te kroki nadal mają zastosowanie.

7.4 zagadnienia dotyczące wydajności z kontrolą źródła danych jednostki

Zobaczyliśmy przypadków w wielowątkowych wydajności i testy obciążeniowe, gdzie wydajność aplikacji sieci web

za pomocą kontroli EntityDataSource pogorszy znacznie. Podstawową przyczyną jest to, że EntityDataSource wielokrotnie wywołuje MetadataWorkspace.LoadFromAssembly na zestawy przywoływane przez aplikację sieci Web, aby dowiedzieć się, typ, który będzie służyć jako jednostki.

Rozwiązanie jest równa ContextTypeName z EntityDataSource nazwą typu klasy pochodnej obiektu ObjectContext. Wyłącza mechanizm, który skanuje wszystkich przywoływanych zestawów typów jednostek.

Ponadto ustawienie pola ContextTypeName zapobiega to problem z funkcjonalnością gdzie EntityDataSource w programie .NET 4.0 zgłasza wyjątku ReflectionTypeLoadException, gdy nie może załadować typu z zestawu przy użyciu odbicia. Ten problem został rozwiązany w .NET 4.5.

7.5 jednostki POCO i serwery proxy śledzenia zmian

Entity Framework pozwala używać klas niestandardowych danych wraz z modelem danych bez żadnych modyfikacji klas danych, samodzielnie. Oznacza to, że za pomocą obiektów CLR "zwykły stary" (POCO), takie jak istniejące obiekty domeny, modełu danych. Te POCO klas danych (znany także jako zakres trwałość obiektów), które są mapowane do jednostek, które są zdefiniowane w modele danych, obsługują większość tego samego zapytania, wstawianie, aktualizowanie i usuwanie zachowania jako typy jednostek, które są generowane przez narzędzia modelu Entity Data Model.

Entity Framework można również tworzyć klasy pochodne typy POCO, w których są używane, jeśli chcesz włączyć funkcje, takie jak powolne ładowanie i automatyczne śledzenie zmian w jednostki POCO klasy w serwera proxy. Twoich zajęciach POCO muszą spełniać określone wymagania, aby umożliwić Entity Framework użyć serwerów proxy, zgodnie z opisem w tym miejscu: <http://msdn.microsoft.com/library/dd468057.aspx>.

Serwery proxy śledzenia szansy powiadomi przez menedżera stanu obiektu każdorazowo dowolne z właściwości jednostki ma swoją wartość, więc Entity Framework zna rzeczywistego stanu jednostki przez cały czas. Odbywa się to przez dodanie zdarzenia powiadomień do treści metod ustawiających właściwości, a ponieważ Menedżer stanu obiektu przetwarzania takie zdarzenia. Należy zauważyć, że tworzenie proxy jednostka będzie najczęściej być droższe niż tworzenia jednostki POCO-proxy z powodu dodano zestaw zdarzenia utworzone przez program Entity Framework.

Podczas jednostki POCO nie ma serwera proxy śledzenia zmian, zmiany zostaną znalezione, porównując zawartość jednostki względem kopię poprzedniego zapisanego stanu. To szczegółowe porównanie staną się długotrwałym procesem w przypadku wielu jednostek w kontekście użytkownika, lub gdy jednostek bardzo dużą ilość właściwości, nawet jeśli żaden z nich zmianie od czasu ostatniego porównania miało miejsce.

W podsumowaniu: zapłacisz wydajności trafień, podczas tworzenia serwera proxy śledzenia zmian, ale śledzenie zmian mogą pomóc przyspieszyć proces wykrywania zmian, jeśli obiekty wiele właściwości, gdy masz wiele jednostek w modelu. Dla jednostek z mniejszą liczbą właściwości, których ilość jednostek nie rozwój zbyt dużo masz serwery proxy śledzenia zmian nie może być wiele korzyści.

8 ładowanie powiązanych jednostek

8.1 powolne ładowanie programu vs. Wczesne ładowanie

Entity Framework oferuje kilka różnych sposobów, które można załadować jednostek, które są powiązane z jednostki docelowej. Na przykład, kiedy wykonujesz zapytanie dotyczące produktów, istnieją różne sposoby powiązane zamówienia zostaną załadowane do menedżera stanu obiektu. Z punktu widzenia wydajności będzie największych pytania, na które należy wziąć pod uwagę podczas ładowania powiązanych jednostek, powolne ładowanie lub Eager ładowania.

Korzystając z ładowania Eager, powiązanych jednostek jest ładowany wraz z Twojej docelowy zestaw jednostek. Używasz instrukcji dołączania w zapytaniu do wskazania, który związane z jednostkami, z którymi chcesz przełączyć.

Podczas korzystania z opóźnieniem ładowania, początkowego zapytania może dopiero w docelowym zestawem jednostek. Jednak zawsze wtedy, gdy uzyskujesz dostęp do właściwości nawigacji, inne zapytanie jest wystawiony

na podstawie magazynu można załadować obiektu pokrewnego.

Po załadowaniu jednostki dodatkowe pytania dla jednostki załaduje go bezpośrednio z poziomu Menedżera stanu obiektów, zarówno w przypadku korzystania z opóźnieniem ładowania lub wczesne ładowanie.

8.2 jak dokonać wyboru między powolne ładowanie i Eager ładowanie

Ważne jest, zrozumieć różnicę między powolne ładowanie i ładowanie Eager, dzięki czemu można było odpowiednim wyborem dla twojej aplikacji. Dzięki temu łatwiej ocenić zależnościami między wiele żądań względem bazy danych w porównaniu z pojedynczego żądania, który może stanowić duże ładunku. Może być odpowiednie do użycia w innych częściach wczesne ładowanie w niektórych części aplikacji i ładowanie z opóźnieniem.

Na przykład o tym, co dzieje się pod maską Założmy, że chcesz wykonać zapytanie dla klientów, którzy mieszkają w Wielkiej Brytanii i ich liczba zamówień.

Za pomocą wczesne ładowanie

```
using (NorthwindEntities context = new NorthwindEntities())
{
    var ukCustomers = context.Customers.Include(c => c.Orders).Where(c => c.Address.Country == "UK");
    var chosenCustomer = AskUserToPickCustomer(ukCustomers);
    Console.WriteLine("Customer Id: {0} has {1} orders", customer.CustomerID, customer.Orders.Count);
}
```

Przy użyciu ładowania z opóźnieniem

```
using (NorthwindEntities context = new NorthwindEntities())
{
    context.ContextOptions.LazyLoadingEnabled = true;

    //Notice that the Include method call is missing in the query
    var ukCustomers = context.Customers.Where(c => c.Address.Country == "UK");

    var chosenCustomer = AskUserToPickCustomer(ukCustomers);
    Console.WriteLine("Customer Id: {0} has {1} orders", customer.CustomerID, customer.Orders.Count);
}
```

Korzystając z wczesne ładowanie, będzie wystawiać pojedyncze zapytanie, które zwraca wszystkich klientów i zamówienia. Polecenie magazynu wygląda następująco:

```

SELECT
[Project1].[C1] AS [C1],
[Project1].[CustomerID] AS [CustomerID],
[Project1].[CompanyName] AS [CompanyName],
[Project1].[ContactName] AS [ContactName],
[Project1].[ContactTitle] AS [ContactTitle],
[Project1].[Address] AS [Address],
[Project1].[City] AS [City],
[Project1].[Region] AS [Region],
[Project1].[PostalCode] AS [PostalCode],
[Project1].[Country] AS [Country],
[Project1].[Phone] AS [Phone],
[Project1].[Fax] AS [Fax],
[Project1].[C2] AS [C2],
[Project1].[OrderID] AS [OrderID],
[Project1].[CustomerID1] AS [CustomerID1],
[Project1].[EmployeeID] AS [EmployeeID],
[Project1].[OrderDate] AS [OrderDate],
[Project1].[RequiredDate] AS [RequiredDate],
[Project1].[ShippedDate] AS [ShippedDate],
[Project1].[ShipVia] AS [ShipVia],
[Project1].[Freight] AS [Freight],
[Project1].[ShipName] AS [ShipName],
[Project1].[ShipAddress] AS [ShipAddress],
[Project1].[ShipCity] AS [ShipCity],
[Project1].[ShipRegion] AS [ShipRegion],
[Project1].[ShipPostalCode] AS [ShipPostalCode],
[Project1].[ShipCountry] AS [ShipCountry]
FROM (
  SELECT
    [Extent1].[CustomerID] AS [CustomerID],
    [Extent1].[CompanyName] AS [CompanyName],
    [Extent1].[ContactName] AS [ContactName],
    [Extent1].[ContactTitle] AS [ContactTitle],
    [Extent1].[Address] AS [Address],
    [Extent1].[City] AS [City],
    [Extent1].[Region] AS [Region],
    [Extent1].[PostalCode] AS [PostalCode],
    [Extent1].[Country] AS [Country],
    [Extent1].[Phone] AS [Phone],
    [Extent1].[Fax] AS [Fax],
    1 AS [C1],
    [Extent2].[OrderID] AS [OrderID],
    [Extent2].[CustomerID] AS [CustomerID1],
    [Extent2].[EmployeeID] AS [EmployeeID],
    [Extent2].[OrderDate] AS [OrderDate],
    [Extent2].[RequiredDate] AS [RequiredDate],
    [Extent2].[ShippedDate] AS [ShippedDate],
    [Extent2].[ShipVia] AS [ShipVia],
    [Extent2].[Freight] AS [Freight],
    [Extent2].[ShipName] AS [ShipName],
    [Extent2].[ShipAddress] AS [ShipAddress],
    [Extent2].[ShipCity] AS [ShipCity],
    [Extent2].[ShipRegion] AS [ShipRegion],
    [Extent2].[ShipPostalCode] AS [ShipPostalCode],
    [Extent2].[ShipCountry] AS [ShipCountry],
    CASE WHEN ([Extent2].[OrderID] IS NULL) THEN CAST(NULL AS int) ELSE 1 END AS [C2]
  FROM [dbo].[Customers] AS [Extent1]
  LEFT OUTER JOIN [dbo].[Orders] AS [Extent2] ON [Extent1].[CustomerID] = [Extent2].[CustomerID]
  WHERE N'UK' = [Extent1].[Country]
) AS [Project1]
ORDER BY [Project1].[CustomerID] ASC, [Project1].[C2] ASC

```

Podczas korzystania z opóźnieniem ładowania, będzie początkowo wydawać następujące zapytanie:

```

SELECT
[Extent1].[CustomerID] AS [CustomerID],
[Extent1].[CompanyName] AS [CompanyName],
[Extent1].[ContactName] AS [ContactName],
[Extent1].[ContactTitle] AS [ContactTitle],
[Extent1].[Address] AS [Address],
[Extent1].[City] AS [City],
[Extent1].[Region] AS [Region],
[Extent1].[PostalCode] AS [PostalCode],
[Extent1].[Country] AS [Country],
[Extent1].[Phone] AS [Phone],
[Extent1].[Fax] AS [Fax]
FROM [dbo].[Customers] AS [Extent1]
WHERE N'UK' = [Extent1].[Country]

```

I zawsze możesz uzyskać dostęp do właściwości nawigacji zamówienia klienta względem magazynu wystawiono innego zapytania podobne do następujących:

```

exec sp_executesql N'SELECT
[Extent1].[OrderID] AS [OrderID],
[Extent1].[CustomerID] AS [CustomerID],
[Extent1].[EmployeeID] AS [EmployeeID],
[Extent1].[OrderDate] AS [OrderDate],
[Extent1].[RequiredDate] AS [RequiredDate],
[Extent1].[ShippedDate] AS [ShippedDate],
[Extent1].[ShipVia] AS [ShipVia],
[Extent1].[Freight] AS [Freight],
[Extent1].[ShipName] AS [ShipName],
[Extent1].[ShipAddress] AS [ShipAddress],
[Extent1].[ShipCity] AS [ShipCity],
[Extent1].[ShipRegion] AS [ShipRegion],
[Extent1].[ShipPostalCode] AS [ShipPostalCode],
[Extent1].[ShipCountry] AS [ShipCountry]
FROM [dbo].[Orders] AS [Extent1]
WHERE [Extent1].[CustomerID] = @_EntityKeyValue1',N'@EntityKeyValue1 nchar(5)', @_EntityKeyValue1=N'AROUT'

```

Aby uzyskać więcej informacji, zobacz [ładowanie powiązanych obiektów](#).

8.2.1 powolne ładowanie i ładowanie Eager — ściągawka

Brak coś takiego jak opracowanie wybór wczesne ładowanie, a ładowanie z opóźnieniem. Spróbuj najpierw zrozumieć różnice między dwóch strategii, więc możecie dobrze świadomie decyzyjnie; należy również rozważyć, jeśli kod pasuje do dowolnego z następujących scenariuszy:

SCENARIUSZ	NASZE SUGESTII
Potrzebujesz uzyskać dostęp do wielu właściwości nawigacji z pobrano jednostek?	<p>Nie — prawdopodobnie wykona obie opcje. Jednak jeśli ładunek, który chodzi o przeniesienie zapytanie nie jest zbyt duży, może wystąpić korzyści wydajności za pomocą wczesne ładowanie, co będzie wymagać mniej sieci rund do zmaterializowania obiektów.</p> <p>Tak — Jeśli potrzebujesz uzyskać dostęp do wielu właściwości nawigacji z jednostkami, jak, za pomocą wielu instrukcji #include w zapytaniu z wczesne ładowanie. Uwzględnić więcej jednostek, większy ładunek zapytanie zwróci. Po uwzględnieniu trzech lub więcej jednostek do zapytania, należy wziąć pod uwagę przełączenie na leniwy ładowania.</p>

SCENARIUSZ	NASZE SUGESTII
Czy wiesz, jakie dane będą dokładnie potrzebne w czasie wykonywania?	<p>Nie — powolne ładowanie będą lepsze dla Ciebie. W przeciwnym razie użytkownik może pozostać wykonywaniem zapytań dotyczących danych, nie będą potrzebne.</p> <p>Tak — Eager, ładowanie prawdopodobnie jest najlepszym sposobem; aby ułatwić szybsze ładowanie całym zestawie. Jeśli zapytanie wymaga pobierania bardzo dużych ilości danych, a ten staje się zbyt wolno, a następnie spróbuj leniwy zamiast tego podczas ładowania.</p>
Kod wykonuje dalekie od wartości bazy danych (opóźnienie sieci zwiększone)	<p>Nie — w przypadku opóźnienie sieci nie jest problemem, za pomocą powolne ładowanie może uprościć kod. Należy pamiętać, że topologię aplikacji, mogą ulec zmianie, więc nie odległości bazy danych dla przyznane.</p> <p>Tak — w przypadku sieci jest problemem, tylko można zdecydować, co lepiej dopasować się do danego scenariusza. Zazwyczaj wcześnie ładowanie jest lepiej, ponieważ wymaga mniejszej liczby rund.</p>

8.2.2 problemów z wydajnością przy użyciu wielu obejmującym

Wzięliśmy to pytania wydajności, które obejmują problemów czas odpowiedzi serwera, źródłem problemu jest często zapytania z wieloma instrukcjami Include. Łącznie z powiązanymi obiektami w zapytaniu jest zaawansowanych, jest ważne, aby zrozumieć, co się dzieje w sposób niewidoczny.

Trwa stosunkowo długo dla zapytania z wieloma instrukcjami Include w nim przechodzić przez kompilator naszego wewnętrznego planu w taki sposób, aby wygenerować polecenia magazynu. Większość tego czasu jest poświęcony próby zoptymalizowania wynikowe zapytanie. Polecenie wygenerowanego magazynu będzie zawierać Outer Join lub Unii, dla każdego dołączenia, w zależności od Twojego mapowania. Kwerend, takich jak ta zostanie wyświetlane w dużych wykresów połączonych ze swojej bazy danych w jednym ładunku acerbae problemach przepustowości, szczególnie w przypadku, gdy jest dużo nadmiarowości w ładunku (na przykład, gdy wiele poziomów dołączania są używane do przechodzenia skojarzenia w kierunku jeden do wielu).

Możesz sprawdzić, czy w przypadkach, w którym zapytanie jest zwracany ładunków bardzo duże, uzyskując dostęp do podstawowych TSQL dla zapytania, używając ToTraceString i wykonując polecenie magazynu w SQL Server Management Studio, aby wyświetlić rozmiar ładunku. W takich przypadkach można wypróbować, aby zmniejszyć liczbę instrukcji dołączania w zapytaniu tylko Przenieś potrzebne dane. Lub można przerwać zapytania mniejszych sekwencji podzapytań, na przykład:

Przed przerwaniem kwerendy:

```
using (NorthwindEntities context = new NorthwindEntities())
{
    var customers = from c in context.Customers.Include(c => c.Orders)
                    where c.LastName.StartsWith(lastNameParameter)
                    select c;

    foreach (Customer customer in customers)
    {
        ...
    }
}
```

Po przerwaniu zapytania:

```

using (NorthwindEntities context = new NorthwindEntities())
{
    var orders = from o in context.Orders
                 where o.Customer.LastName.StartsWith(lastNameParameter)
                 select o;

    orders.Load();

    var customers = from c in context.Customers
                    where c.LastName.StartsWith(lastNameParameter)
                    select c;

    foreach (Customer customer in customers)
    {
        ...
    }
}

```

Funkcja będzie działać tylko na śledzonych zapytań, jak firma Microsoft wprowadza korzystanie z możliwości kontekst ma automatycznie przeprowadzić korektę rozdzielcośc i skojarzenie tożsamości.

Podobnie jak w przypadku ładowania z opóźnieniem, kosztem będzie więcej zapytań dla mniejszych ładunków. Umożliwia także projekcje poszczególne właściwości jawnie wybrać tylko potrzebne dane z każdej jednostki, ale można będzie nie być Trwa ładowanie jednostek w tym przypadku, a aktualizacje nie będą obsługiwane.

8.2.3 obejście pozwalające uzyskać powolne ładowanie właściwości

Entity Framework nie obsługuje obecnie powolne ładowanie właściwości skalarne lub zbyt złożone. Jednak w przypadkach, w którym masz tabelę, która obejmuje dużych obiektów, takich jak obiekt BLOB, umożliwia dzielenie tabeli dzielenie dużych właściwości osobne jednostki. Na przykład założmy, że masz tabelę produktu, która zawiera kolumnę zdjcie varbinary. Jeśli często nie trzeba dostęp do tej właściwości w zapytaniach, można użyć tabeli podział w ramach tylko te części jednostki, które zwykle wymagają. Jednostka reprezentująca fotografia produktu tylko zostaną załadowane, gdy potrzebujesz jawnie.

Dobre zasób, który pokazuje, jak włączyć dzielenia tabeli jest "Tabela podział w programie Entity Framework" Gil Fink wpis w blogu: <<http://blogs.microsoft.co.il/blogs/gilf/archive/2009/10/13/table-splitting-in-entity-framework.aspx>>.

9 inne zagadnienia

9.1 wyrzucanie elementów bezużytecznych serwera

Niektórzy użytkownicy mogą wystąpić rywalizacja, który ogranicza równoległość, muszą być oczekiwana, gdy moduł odśmiecania pamięci nie jest poprawnie skonfigurowana. Zawsze, gdy EF jest używany w scenariuszu wielowątkowych, lub w dowolnej aplikacji, który przypomina systemu po stronie serwera, upewnij się włączyć serwer wyrzucania elementów bezużytecznych. Można to zrobić za pomocą proste ustawienie w pliku konfiguracji aplikacji:

```

<?xmlversion="1.0" encoding="utf-8" ?>
<configuration>
    <runtime>
        <gcServer enabled="true" />
    </runtime>
</configuration>

```

To powinno zmniejszyć swoje rywalizacji wątków i zwiększa przepustowość sieci nawet o 30% w scenariuszach procesora CPU jest przepełniony. Ogólnie rzecz biorąc zawsze należy sprawdzić, jak aplikacja zachowuje się przy użyciu klasycznego wyrzucania elementów bezużytecznych (który lepiej jest ona dostrojona dla scenariuszy po stronie interfejsu użytkownika i klienta) oraz serwer wyrzucania elementów bezużytecznych.

9.2 AutoDetectChanges do

Jak wspomniano wcześniej, platformy Entity Framework może wyświetlać problemy z wydajnością, gdy pamięć podrzczna obiekt zawiera wiele jednostek. Niektóre operacje, takie jak Dodaj, Usuń, wyszukiwanie, zapis i SaveChanges, wyzwalały do metody DetectChanges, którego może używać dużej ilości procesora CPU, oparte na wielkości pamięci podrzcznej obiektów stał się. Przyczyną jest, że pamięci podrzcznej obiektów i obiektu stanu Menedżera próbę pozostają zsynchronizowane po każdej operacji wykonywane do kontekstu, aby produkowane danych może być prawidłowe w obszarze szerokiej gamy scenariuszy.

Zazwyczaj jest dobrą praktyką jest pozostawienie programu Entity Framework automatyczna zmiana wykrywanie włączone dla całego cyklu życia aplikacji. Jeśli scenariusz jest negatywny wpływowy wysokie użycie procesora CPU i profiliów wskazują, że przyczyna nadmiernego jest wywołanie metody DetectChanges, należy wziąć pod uwagę tymczasowo wyłączać AutoDetectChanges we fragmencie poniższych kod:

```
try
{
    context.Configuration.AutoDetectChangesEnabled = false;
    var product = context.Products.Find(productId);
    ...
}
finally
{
    context.Configuration.AutoDetectChangesEnabled = true;
}
```

Przed wyłączeniem AutoDetectChanges, warto poznać, może to spowodować Entity Framework utracą możliwość śledzenia określonych informacji o zmiany, które pojawiają się na jednostkach. Jeśli obsługiwane nieprawidłowo, może to spowodować niespójność danych w swojej aplikacji. Aby uzyskać więcej informacji na temat wyłączania AutoDetectChanges, przeczytaj <<http://blog.oneunicorn.com/2012/03/12/secrets-of-detectchanges-part-3-switching-off-automatic-detectchanges/>>.

9.3 kontekst na żądanie

Konteksty platformy Entity Framework są przeznaczone do służyć jako środowisko krótkotrwałe wystąpień w celu zapewnienia najbardziej optymalną wydajność. Konteksty powinny być krótkie krótkotrwałe i usuwane i jako takie zostały zaimplementowane do bardzo i reutilize metadanych, jeśli to możliwe. W scenariuszach sieci web należy pamiętać o tym i nie kontekstu dla więcej niż czas trwania pojedynczego żądania. Podobnie w scenariuszach bez sieci web kontekstu powinny zostać odrzucone zależnie od zrozumieć różne poziomy buforowania w programie Entity Framework. Ogólnie rzecz biorąc jeden unikać występowania wystąpienie kontekstu, w całym cyklu życia aplikacji, a także kontekst na wątek i konteksty statyczne.

9.4 semantyka wartości null bazy danych

Entity Framework domyślnie spowoduje wygenerowanie kodu SQL, który ma C# wartość null, semantyka porównania. Należy wziąć pod uwagę poniższe przykładowe zapytanie:

```

int? categoryId = 7;
int? supplierId = 8;
decimal? unitPrice = 0;
short? unitsInStock = 100;
short? unitsOnOrder = 20;
short? reorderLevel = null;

var q = from p in context.Products
        where p.Category.CategoryName == "Beverages"
            || (p.CategoryID == categoryId
                || p.SupplierID == supplierId
                || p.UnitPrice == unitPrice
                || p.UnitsInStock == unitsInStock
                || p.UnitsOnOrder == unitsOnOrder
                || p.ReorderLevel == reorderLevel)
        select p;

var r = q.ToList();

```

W tym przykładzie firma Microsoft porównujemy różne zmienne dopuszczającego wartość null, przed nullable właściwości jednostki, takie jak `IDDostawcy` i `UnitPrice`. Wygenerowany SQL dla tego zapytania zostanie wyświetcone pytanie, jeśli wartość tego parametru jest taka sama jak wartość kolumny lub parametru i wartości w kolumnach mają wartość null. To spowoduje ukrycie sposób serwer bazy danych obsługuje wartości null i zapewni spójne C# wartość null, środowisko pochodzących od różnych dostawców innej bazy danych. Z drugiej strony, wygenerowany kod jest nieco zawiłe i mogą nie działać poprawnie, gdy ilość porównania w klauzuli `where` instrukcji kwerendy zwiększa się do dużej liczby.

Jest jednym ze sposobów, aby rozwiązać ten problem przy użyciu bazy danych, semantyka wartości null. Należy zauważać, że to może potencjalnie działać inaczej niż w C# null semantyki od teraz platformy Entity Framework wygeneruje prostsze SQL Server, który uwidacznia sposób aparatu bazy danych obsługuje wartości null. Semantyka wartości null dla bazy danych może być aktywowany na kontekstowe o jeden wiersz jednej konfiguracji względem kontekstowy konfiguracji:

```
context.Configuration.UseDatabaseNullSemantics = true;
```

Mały, średni rozmiar zapytania nie będą wyświetlane zwiększenie wydajności zauważalne podczas korzystania z bazy danych, semantyka wartości null, ale różnica staną się bardziej zauważalne w zapytaniach z dużą liczbą potencjalnych porównania wartości null.

W powyższym zapytaniu przykład różnicę w wydajności była mniej niż 2% microbenchmark, działające w środowisku kontrolowanym.

9.5 asynchronousne

Obsługa Framework 6 wprowadzone jednostki operacji asynchronousnych w przypadku uruchamiania na .NET 4.5 lub nowszej. W większości przypadków aplikacji, które mają we/wy związane z rywalizacji o zasoby będą korzystać maksymalnie z zapytania asynchronousnego i operacje zapisywania. Jeśli aplikacja nie odczuwa rywalizacji o zasoby we/wy, użycie `async` w przypadku najlepszych były uruchamiane synchronicznie i zwracają wynik, w tym samym czasie jako synchroniczne wywołanie lub w najgorszym przypadku, po prostu Odrocz wykonywania zadania asynchronousnego i dodać dodatkowe `tim e` do wykonania danego scenariusza.

Instrukcje dotyczące sposobu asynchronousnego programowania pracy, która pomoże przy wyborze rozwiązania, jeśli `async` poprawi wydajność aplikacji odwiedzanych przez użytkownika

<http://msdn.microsoft.com/library/hh191443.aspx>. Aby uzyskać więcej informacji dotyczących używania operacji asynchronousnych na platformie Entity Framework, zobacz [Async zapytania i Zapisz](#).

9.6 NGEN

Entity Framework 6 nie jest dostarczany w domyślnej instalacji programu .NET framework. W efekcie zestawów

platformy Entity Framework nie są domyślnie, co oznacza, że cały kod platformy Entity Framework podlega tych samych kosztów JIT'ing jako innego zestawu MSIL przez ngen. Może to obniżyć środowisko F5 podczas opracowywania, a także podczas zimnego uruchamiania aplikacji w środowiskach produkcyjnych. W celu zmniejszenia kosztów procesora CPU i pamięci JIT'ing wskazane jest NGEN obrazy platformy Entity Framework, zgodnie z potrzebami. Aby uzyskać więcej informacji na temat sposobu zwiększenia wydajności uruchamiania programu Entity Framework 6 za pomocą narzędzia NGEN, zobacz [zwiększanie wydajności uruchamiania za pomocą narzędzia NGen](#).

Zbierając 9,7 najpierw kod i EDMX

Entity Framework przyczyniło się do problemu niezgodności impedancji między programowaniem dzięki obiektowej i relacyjnymi bazom danych dzięki reprezentacji w pamięci modelu koncepcyjnego (obiekty), schemat magazynu (baza danych) i mapowanie między dwoma. Te metadane nosi nazwę modelu Entity Data Model lub EDM w skrócie. Z tym EDM Entity Framework pochodzi widoków do przesyłania danych z obiektów w pamięci do bazy danych i kopii.

Gdy Entity Framework jest używany przy użyciu pliku EDMX oznacza formalnie określa modelu koncepcyjnego, schemat magazynu i mapowanie, a następnie etap podczas ładowania modelu ma tylko do sprawdzania, czy EDM jest poprawny (na przykład, upewnij się, Brak Brak mapowań), następnie Generowanie widoków, a następnie zweryfikować widoków i mają te metadane, które są gotowe do użycia. Wykonania może następnie kwerendę, lub tylko nowe dane zapisane w magazynie danych.

Podejście Code First jest Centrum, zaawansowane generator modelu Entity Data Model. Entity Framework musi produkować EDM z udostępnionego kodu; robi to analizowanie klas, które są zaangażowane w modelu, stosując konwencje i konfigurowanie modelu za pomocą interfejsu API Fluent. Po utworzeniu EDM Entity Framework zasadniczo zachowuje się taki sam sposób, jak będzie miał już obecne w projekcie pliku EDMX. W związku z tym budowania modelu z Code First dodaje dodatkowe złożoność, która przekłada się na wolniejszych czas uruchamiania programu Entity Framework, w porównaniu z koniecznością EDMX. Koszt jest całkowicie zależny od rozmiaru i złożoność modelu, który jest konstruowany.

Wybór użycia EDMX a Code First, jest ważne, aby dowiedzieć się, że elastyczność wprowadzone przez rozwiązanie Code First zwiększa koszt budowania modelu po raz pierwszy. Jeśli aplikacja może wytrzymać koszt tego obciążenia po raz pierwszy to zazwyczaj Code First będą preferowany sposób, aby przejść.

10 badanie wydajności

10.1 przy użyciu programu Visual Studio Profiler

Jeśli występują problemy z wydajnością za pomocą programu Entity Framework, można użyć profiler, takiego jak wbudowany w program Visual Studio, aby zobaczyć, gdzie aplikacja spędza czas. To narzędzie, firma Microsoft służącego do generowania wykresów kołowych "Eksplorowanie wydajności programu ADO.NET Entity Framework — część 1" wpis w blogu (<http://blogs.msdn.com/b/adonet/archive/2008/02/04/exploring-the-performance-of-the-ado-net-entity-framework-part-1.aspx>) ukazują, gdzie Entity Framework spędza czas podczas wykonywania kwerend ścieżce nieaktywnej i bez wyłączenia zasilania.

Wpis w blogu "Profilowania platformy Entity Framework przy użyciu Profiler 2010 usługi Visual Studio" napisane przez dane i modelowanie zespół doradczy klientów przedstawiono przykład rzeczywistych jak profiler one używane do badania problemów z wydajnością. <<http://blogs.msdn.com/b/dmcat/archive/2010/04/30/profiling-entity-framework-using-the-visual-studio-2010-profiler.aspx>>. Ten wpis został napisany dla aplikacji systemu windows. Jeśli chcesz profilować aplikację sieci web narzędzi rejestratora wydajności Windows (WPR) i Windows Analizator wydajności (WPA) może działać lepiej niż pracy w programie Visual Studio. WPR i WPA są częścią zestawu narzędzi wydajności Windows, który jest dołączony do Windows Assessment and Deployment Kit (<http://www.microsoft.com/download/details.aspx?id=39982>).

10.2 profilowanie/bazy danych aplikacji

Narzędzia, takie jak profiler wbudowany w program Visual Studio poinformować Cię, w którym aplikacja jest

poświęcania czasu. Inny rodzaj profiler jest dostępny, przeprowadza analizy dynamicznej uruchomionej aplikacji, zarówno w produkcji wstępnej lub w zależności od potrzeb, a szuka typowych pułapek oraz niezalecane wzorce dostępu do bazy danych.

Dwa komercyjnego profilowania są Profiler Framework jednostki (<http://efprof.com>) i ORMPProfiler (<http://ormprofiler.com>).

Jeśli aplikacja to aplikacja MVC za pomocą funkcji Code First, możesz użyć MiniProfiler StackExchange firmy Scott Hanselman opis tego narzędzia w jego blogu znajduje się na:

<http://www.hanselman.com/blog/NuGetPackageOfTheWeek9ASPNETMiniProfilerFromStackExchangeRocksYourWorld.aspx>.

Aby uzyskać więcej informacji na profilowaniu aktywności bazy danych aplikacji, zobacz artykuł w MSDN Magazine Julie Lerman pod tytułem [profilowania działań w bazie danych platformy Entity Framework](#).

10.3 Rejestrator bazy danych

Jeśli używasz platformy Entity Framework 6 należy również rozważyć korzystanie z funkcji wbudowanej funkcje rejestrowania. Właściwość bazy danych kontekstu można zobowiązany do rejestrowania swojej działalności za pośrednictwem prostej konfiguracji jednego wiersza:

```
using (var context = newQueryComparison.DbC.NorthwindEntities())
{
    context.Database.Log = Console.WriteLine;
    var q = context.Products.Where(p => p.Category.CategoryName == "Beverages");
    q.ToList();
}
```

W tym przykładzie działań w bazie danych będą rejestrowane w konsoli, ale można skonfigurować właściwości rejestrowania, aby wywołać dowolną akcję<ciąg> delegować.

Jeśli chcesz włączyć rejestrowanie w bazie danych bez konieczności ponownego kompilowania i korzystania z programu Entity Framework 6.1 lub nowszej, możesz to zrobić, dodając interceptor w pliku web.config lub app.config aplikacji.

```
<interceptors>
    <interceptor type="System.Data.Entity.Infrastructure.Interception.DatabaseLogger, EntityFramework">
        <parameters>
            <parameter value="C:\Path\To\My\LogOutput.txt"/>
        </parameters>
    </interceptor>
</interceptors>
```

Aby uzyskać więcej informacji na temat dodawania rejestrowanie bez konieczności ponownego kompilowania przejdź do pozycji <http://blog.oneunicorn.com/2014/02/09/ef-6-1-turning-on-logging-without-recompiling/>.

Dodatek 11

11.1 środowisko testowe A.

To środowisko używa ustawień maszyny 2 z bazy danych na osobnym komputerze od aplikacji klienckiej. Maszyny są w tej samej perspektywy regału sprzętowego, więc opóźnienie sieci jest stosunkowo niska, ale bardziej realistycznego niż środowisku pojedynczego komputera.

11.1.1 serwer aplikacji

11.1.1.1 środowisko oprogramowania

- Środowisko oprogramowania programu Entity Framework 4

- Nazwa systemu operacyjnego: System Windows Server 2008 R2 Enterprise z dodatkiem SP1.

- Program Visual Studio 2010 — Ultimate.
- Visual Studio 2010 z dodatkiem SP1 (tylko dla niektórych porównania).
- Środowisko oprogramowania programu Entity Framework 5 i 6
 - Nazwa systemu operacyjnego: Windows 8.1 Enterprise
 - Visual Studio 2013 — Ultimate.

11.1.1.2 środowisko sprzętu

- Dwurdzeniowy procesor: Intel(R) Xeon(R) L5520 Procesora W3530 @ 2,27 GHz, 2261 Mhz8 GHz, 4 rdzenie, 84 procesorów logicznych.
- RamRAM 2412 GB.
- 136 GB SCSI250GB SATA 7200 obr./min 3GB/s dysku podzielić na 4 partycjami.

11.1.2 serwer bazy danych

11.1.2.1 środowisko oprogramowania

- Nazwa systemu operacyjnego: Windows Server 2008 R28.1 Enterprise z dodatkiem SP1.
- SQL Server 2008 R22012.

11.1.2.2 środowisko sprzętu

- Pojedynczy procesor: Intel(R) Xeon(R) Procesora L5520 @ 2,27 GHz, 2261 MhzES-1620 0 @ 3,60 GHz, 4 rdzenie, 8 procesorów logicznych.
- RamRAM 824 GB.
- 465 GB ATA500GB SATA 7200 obr./min 6GB/s dysku podzielić na 4 partycjami.

11.2 testy porównanie wydajności zapytań B.

Northwind model była używana do wykonywania tych testów. Został on wygenerowany z bazy danych za pomocą projektanta programu Entity Framework. Następujący kod został użyty porównać wydajność opcje wykonywania zapytań:

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Data.Common;
using System.Data.Entity.Infrastructure;
using System.Data.EntityClient;
using System.Data.Objects;
using System.Linq;

namespace QueryComparison
{
    public partial class NorthwindEntities : ObjectContext
    {
        private static readonly Func<NorthwindEntities, string, IQueryable<Product>> productsForCategoryCQ =
            CompiledQuery.Compile(
                (NorthwindEntities context, string categoryName) =>
                    context.Products.Where(p => p.Category.CategoryName == categoryName)
            );

        public IQueryable<Product> InvokeProductsForCategoryCQ(string categoryName)
        {
            return productsForCategoryCQ(this, categoryName);
        }
    }

    public class QueryTypePerfComparison
    {
        private static string entityConnectionString =
            @"metadata=res://*/Northwind.csdl|res://*/Northwind.ssdl|res://*/Northwind.msl;provider=System.Data.SqlClient;
provider connection string='data source=.;initial catalog=Northwind;integrated
security=True;multipleactiveresultsets=True;App=EntityFramework"";

        public void LINQIncludingContextCreation()
        {

```

```

    {
        using (NorthwindEntities context = new NorthwindEntities())
        {
            var q = context.Products.Where(p => p.Category.CategoryName == "Beverages");
            q.ToList();
        }
    }

    public void LINQNoTracking()
    {
        using (NorthwindEntities context = new NorthwindEntities())
        {
            context.Products.MergeOption = MergeOption.NoTracking;

            var q = context.Products.Where(p => p.Category.CategoryName == "Beverages");
            q.ToList();
        }
    }

    public void CompiledQuery()
    {
        using (NorthwindEntities context = new NorthwindEntities())
        {
            var q = context.InvokeProductsForCategoryCQ("Beverages");
            q.ToList();
        }
    }

    public void ObjectQuery()
    {
        using (NorthwindEntities context = new NorthwindEntities())
        {
            ObjectQuery<Product> products = context.Products.Where("it.Category.CategoryName =
'Beverages'");
            products.ToList();
        }
    }

    public void EntityCommand()
    {
        using (EntityConnection eConn = new EntityConnection(entityConnectionString))
        {
            eConn.Open();
            EntityCommand cmd = eConn.CreateCommand();
            cmd.CommandText = "Select p From NorthwindEntities.Products As p Where p.Category.CategoryName =
'Beverages'";

            using (EntityDataReader reader = cmd.ExecuteReader(CommandBehavior.SequentialAccess))
            {
                List<Product> productsList = new List<Product>();
                while (reader.Read())
                {
                    DbDataRecord record = (DbDataRecord)reader.GetValue(0);

                    // 'materialize' the product by accessing each field and value. Because we are
materializing products, we won't have any nested data readers or records.
                    int fieldCount = record.FieldCount;

                    // Treat all products as Product, even if they are the subtype DiscontinuedProduct.
                    Product product = new Product();

                    product.ProductID = record.GetInt32(0);
                    product.ProductName = record.GetString(1);
                    product.SupplierID = record.GetInt32(2);
                    product.CategoryID = record.GetInt32(3);
                    product.QuantityPerUnit = record.GetString(4);
                    product.UnitPrice = record.GetDecimal(5);
                    product.UnitsInStock = record.GetInt16(6);
                    product.UnitsOnOrder = record.GetInt16(7);
                }
            }
        }
    }
}

```

```

        product.ReorderLevel = record.GetInt16(8);
        product.Discontinued = record.GetBoolean(9);

        productsList.Add(product);
    }
}
}

public void ExecuteStoreQuery()
{
    using (NorthwindEntities context = new NorthwindEntities())
    {
        ObjectResult<Product> beverages = context.ExecuteStoreQuery<Product>(
@"
    SELECT      P.ProductID, P.ProductName, P.SupplierID, P.CategoryID, P.QuantityPerUnit, P.UnitPrice,
P.UnitsInStock, P.UnitsOnOrder, P.ReorderLevel, P.Discontinued
    FROM        Products AS P INNER JOIN Categories AS C ON P.CategoryID = C.CategoryID
    WHERE       (C.CategoryName = 'Beverages')"
);
        beverages.ToList();
    }
}

public void ExecuteStoreQueryDbContext()
{
    using (var context = new QueryComparison.DbC.NorthwindEntities())
    {
        var beverages = context.Database.SqlQuery<QueryComparison.DbC.Product>(
@"
    SELECT      P.ProductID, P.ProductName, P.SupplierID, P.CategoryID, P.QuantityPerUnit, P.UnitPrice,
P.UnitsInStock, P.UnitsOnOrder, P.ReorderLevel, P.Discontinued
    FROM        Products AS P INNER JOIN Categories AS C ON P.CategoryID = C.CategoryID
    WHERE       (C.CategoryName = 'Beverages')"
);
        beverages.ToList();
    }
}

public void ExecuteStoreQueryDbSet()
{
    using (var context = new QueryComparison.DbC.NorthwindEntities())
    {
        var beverages = context.Products.SqlQuery(
@"
    SELECT      P.ProductID, P.ProductName, P.SupplierID, P.CategoryID, P.QuantityPerUnit, P.UnitPrice,
P.UnitsInStock, P.UnitsOnOrder, P.ReorderLevel, P.Discontinued
    FROM        Products AS P INNER JOIN Categories AS C ON P.CategoryID = C.CategoryID
    WHERE       (C.CategoryName = 'Beverages')"
);
        beverages.ToList();
    }
}

public void LINQIncludingContextCreationDbContext()
{
    using (var context = new QueryComparison.DbC.NorthwindEntities())
    {
        var q = context.Products.Where(p => p.Category.CategoryName == "Beverages");
        q.ToList();
    }
}

public void LINQNoTrackingDbContext()
{
    using (var context = new QueryComparison.DbC.NorthwindEntities())
    {
        var q = context.Products.AsNoTracking().Where(p => p.Category.CategoryName == "Beverages");
        q.ToList();
    }
}
}

```

```
}
```

Model Navision 11,3 C.

Baza danych Navision jest używana do pokazu Microsoft Dynamics — NAV. dużej bazy danych Wygenerowany model koncepcyjny zawiera 1005 jednostki zestawów i zestawów skojarzeń 4227. Model używany w teście jest "stała" — nie dziedziczenia dodano do niego.

11.3.1 zapytania używane do testów Navision

Lista zapytań, używanych przy użyciu modelu Navision zawiera 3 kategorie zapytań jednostki SQL:

11.3.1.1 wyszukiwanie

Proste wyszukiwanie zapytania nie agregacji

- Liczba: 16232
- Przykład:

```
<Query complexity="Lookup">
  <CommandText>Select value distinct top(4) e.Idle_Time From NavisionFKContext.Session as e</CommandText>
</Query>
```

11.3.1.2 SingleAggregating

Normalne BI zapytanie o wiele agregacji, ale nie sumy częściowe (jedno zapytanie)

- Liczba: 2313
- Przykład:

```
<Query complexity="SingleAggregating">
  <CommandText>NavisionFK.MDF_SessionLogin_Time_Max()</CommandText>
</Query>
```

Gdzie MDF_SessionLogin_czasu_Max() jest zdefiniowany w modelu w postaci:

```
<Function Name="MDF_SessionLogin_Time_Max" ReturnType="Collection(DateTime)">
  <DefiningExpression>SELECT VALUE Edm.Min(E.Login_Time) FROM NavisionFKContext.Session as E</DefiningExpression>
</Function>
```

11.3.1.3 AggregatingSubtotals

Zapytanie analizy Biznesowej za pomocą agregacji i sumy częściowe (za pośrednictwem wszystkich Unia)

- Liczba: 178
- Przykład:

```

<Query complexity="AggregatingSubtotals">
    <CommandText>
using NavisionFK;
function AmountConsumed(entities Collection([CRONUS_International_Ltd__Zone])) as
(
    Edm.Sum(select value N.Block_Movement FROM entities as E, E.CRONUS_International_Ltd__Bin as N)
)
function AmountConsumed(P1 Edm.Int32) as
(
    AmountConsumed(select value e from NavisionFKContext.CRONUS_International_Ltd__Zone as e where
e.Zone_Ranking = P1)
)
-----
-----
(
    select top(10) Zone_Ranking, Cross_Dock_Bin_Zone, AmountConsumed(GroupPartition(E))
    from NavisionFKContext.CRONUS_International_Ltd__Zone as E
    where AmountConsumed(E.Zone_Ranking) > @MinAmountConsumed
    group by E.Zone_Ranking, E.Cross_Dock_Bin_Zone
)
union all
(
    select top(10) Zone_Ranking, Cast(null as Edm.Byte) as P2, AmountConsumed(GroupPartition(E))
    from NavisionFKContext.CRONUS_International_Ltd__Zone as E
    where AmountConsumed(E.Zone_Ranking) > @MinAmountConsumed
    group by E.Zone_Ranking
)
union all
{
    Row(Cast(null as Edm.Int32) as P1, Cast(null as Edm.Byte) as P2, AmountConsumed(select value E
                                                from
NavisionFKContext.CRONUS_International_Ltd__Zone as E
                                                where AmountConsumed(E.Zone_Ranking)
> @MinAmountConsumed))
}</CommandText>
<Parameters>
    <Parameter Name="MinAmountConsumed" DbType="Int32" Value="10000" />
</Parameters>
</Query>

```

Zwiększenie wydajności uruchamiania za pomocą narzędzia NGen

13.09.2018 • 9 minutes to read • [Edit Online](#)

NOTE

EF6 począwszy tylko — funkcje, interfejsów API itp. z opisem na tej stronie zostały wprowadzone w programie Entity Framework 6. Jeśli używasz starszej wersji, niektóre lub wszystkie informacje, nie ma zastosowania.

Program .NET Framework obsługuje generowanie obrazów natywnych dla zarządzanych aplikacji i bibliotek, aby ułatwić szybsze uruchamianie aplikacji, a także w niektórych przypadkach Użyj mniejszej ilości pamięci. Obrazy natywne są tworzone przez tłumaczenie zestawów kodu zarządzanego w plikach zawierających instrukcji maszyny macierzystej, zanim aplikacja zostanie wykonany, zwalniania kompilatora .NET JIT (Just-In-Time), trzeba wygenerować macierzysty zgodnie z instrukcjami na środowisko uruchomieniowe aplikacji.

Przed wersją 6 środowiska uruchomieniowego EF core biblioteki były częścią programu .NET Framework i obrazy natywne są generowane automatycznie dla nich. Począwszy od wersji 6 całego środowiska uruchomieniowego EF została połączona w pakiet NuGet platformy EntityFramework. Obrazy natywne mają do chwili obecnej można wygenerować za pomocą narzędzia wiersza polecenia NGen.exe, aby uzyskać wyniki podobne.

Empiryczne obserwacje pokazują, że obrazy natywne zestawów środowiska uruchomieniowego EF można wyciąć od 1 do 3 sekund czas uruchamiania aplikacji.

Jak używać NGen.exe

Najbardziej podstawowa funkcja narzędzia NGen.exe jest "Zainstaluj" (oznacza to, aby utworzyć i zachować na dysku) obrazów natywnych dla zestawu i wszystkich jego zależności bezpośrednich. Poniżej przedstawiono, jak można uzyskać, który:

1. Otwórz okno wiersza polecenia jako administrator
2. Zmień bieżący katalog roboczy do lokalizacji zestawów, który chcesz wygenerować obrazów natywnych dla:

```
cd <*Assemblies location*>
```

3. W zależności od używanego systemu operacyjnego i konfiguracji aplikacji może być konieczne generowanie obrazów natywne dla architektury 32-bitowej i 64-bitowa architektura lub obu.

Dla 32-bitowych, uruchom:

```
%WINDIR%\Microsoft.NET\Framework\v4.0.30319\ngen install <Assembly name>
```

Dla 64-bitowy, uruchom:For 64 bit run:

```
%WINDIR%\Microsoft.NET\Framework64\v4.0.30319\ngen install <Assembly name>
```

TIP

Generowanie obrazów macierzystych dla niewłaściwego architektury jest bardzo Częsty błąd. W razie wątpliwości można po prostu generowanie obrazów natywnych dla wszystkich architektur, które dotyczą system operacyjny zainstalowany na maszynie.

NGen.exe obsługuje także inne funkcje, takie jak odinstalowywanie i wyświetlanie zainstalowanych obrazów natywnych, kolejkowania generowania wielu obrazów itd. Aby uzyskać więcej szczegółów dotyczących użycia przeczytaj [dokumentacji NGen.exe](#).

Kiedy należy używać NGen.exe

Jeśli chodzi o określić zestawy, które generuje obrazy natywne dla w aplikacji opartej na EF w wersji 6 lub nowszego, należy rozważyć następujące opcje:

- **Głównym zestawie środowiska uruchomieniowego EF, EntityFramework.dll:** Typowa aplikacja na podstawie EF jest wykonywany znacząca ilość kodu z tego zestawu podczas uruchamiania lub w jego pierwszego dostępu do bazy danych. W związku z tym tworzenie obrazów natywnych tego zestawu powoduje wygenerowanie największy wzrost wydajności uruchamiania.
- **Każdy zespół dostawcy EF używanych przez aplikację:** czas uruchamiania mogą również zyskać nieco dzięki generowanie obrazów macierzystych, które z nich. Na przykład jeśli aplikacja używa dostawcy EF dla programu SQL Server należy wygenerować obraz natywny dla EntityFramework.SqlServer.dll.
- **Zestawy aplikacji i inne zależności:** [dokumentacji NGen.exe](#) opisano ogólne kryteria wyboru zestawy, które można wygenerować obrazów natywnych dla i wpływu na obrazy natywne dotyczące zabezpieczeń, Zaawansowane opcje, takie jak "trwałego powiązania" scenariuszy, takich jak przy użyciu obrazów natywnych w debugowania i profilowania w scenariuszach itp.

TIP

Upewnij się, starannie mierzenie wpływu na wydajność uruchamiania i ogólną wydajność aplikacji przy użyciu obrazów macierzystych i porównać ich rzeczywiste wymagania. Natomiast obrazy natywne pomoże ogólnie poprawy uruchomienia wydajność i w niektórych przypadkach zmniejszenie zużycia pamięci, nie wszystkie scenariusze skorzystają równomiernie. Na przykład w stanie stabilności wykonywania (gdy co najmniej raz wywołania wszystkich metod, które są używane przez aplikację) kod wygenerowany przez kompilator JIT może w rzeczywistości przynieść wydajność nieco lepiej niż obrazy natywne.

Za pomocą NGen.exe w komputerze deweloperskim

Podczas tworzenia .NET JIT kompilatora oferuje najlepsze rozwiązanie ogólne dla kodu, które zmieniają się często. Generowanie obrazów macierzystych skompilowanych zależności, takich jak zestawy środowiska uruchomieniowego EF może pomóc przyspieszyć programowanie i testowanie przez wycinanie kilka sekund na początku każdego wykonania.

Dobrym miejscem do śledzenia Znajdź zestawy środowiska uruchomieniowego EF jest lokalizacja pakietu NuGet dla rozwiązania. Na przykład w aplikacji przy użyciu programu EF 6.0.2 z programem SQL Server i przeznaczonych dla platformy .NET 4.5 lub nowszej w oknie wiersza polecenia można wpisać następujące (Pamiętaj otworzyć go jako administrator):

```
cd <Solution directory>\packages\EntityFramework.6.0.2\lib\net45  
%WINDIR%\Microsoft.NET\Framework\v4.0.30319\ngen install EntityFramework.SqlServer.dll  
%WINDIR%\Microsoft.NET\Framework64\v4.0.30319\ngen install EntityFramework.SqlServer.dll
```

NOTE

To wykorzystuje fakt, że instalowanie obrazów natywnych dla platformy EF dostawcy dla programu SQL Server zostanie również domyślnie zainstalowany obrazy natywne dla zestawu głównego środowiska uruchomieniowego EF. To działa, ponieważ NGen.exe może wykryć, czy EntityFramework.dll jest bezpośrednią zależność zestawu EntityFramework.SqlServer.dll, znajdującego się w tym samym katalogu.

Tworzenie obrazów natywnych podczas instalacji

Zestaw narzędzi WiX obsługuje kolejkowania generowanie obrazów natywnych dla zestawów zarządzanych podczas instalacji, jak wyjaśniono w tym [poradnik](#). Kolejny alternatywny sposób polega na utworzeniu zadania instalacji niestandardowej, wykonaj polecenie NGen.exe.

Weryfikowanie, czy obrazy natywne są używane na platformie EF

Można sprawdzić, czy określona aplikacja używa natywny zestaw, wyszukując załadowanych zestawów, które mają rozszerzenie ".ni.dll" lub ".ni.exe". Na przykład obraz natywny dla zestawu głównego środowiska uruchomieniowego EF zostanie wywołana EntityFramework.ni.dll. Prosty sposób sprawdzić załadowanych zestawów .NET procesu jest użycie [Eksplorator procesów](#).

Inne zagadnienia, które należy pamiętać o

Tworzenie obrazu natywnego zestawu, nie należy mylić z rejestraniem zestawu w globalnej pamięci podręcznej zestawów (Global Assembly Cache). NGen.exe umożliwia tworzenie obrazów zestawów, które nie znajdują się w pamięci podręcznej GAC, a w rzeczywistości wielu zastosowań, które korzystały z określonej wersji EF mogą udostępniać tego samego obrazu natywnego. Windows 8 może automatycznie tworzyć obrazy natywne dla zestawów umieszczane w pamięci podręcznej GAC, środowiska uruchomieniowego EF jest zoptymalizowany do wdrożenia wraz z aplikacją i nie jest zalecane rejestrowanie w pamięci podręcznej GAC, ponieważ ma negatywny wpływ na rozpoznawania zestawu i Obsługa aplikacji wśród innych aspektów.

Widoki wstępnie wygenerowanego mapowania

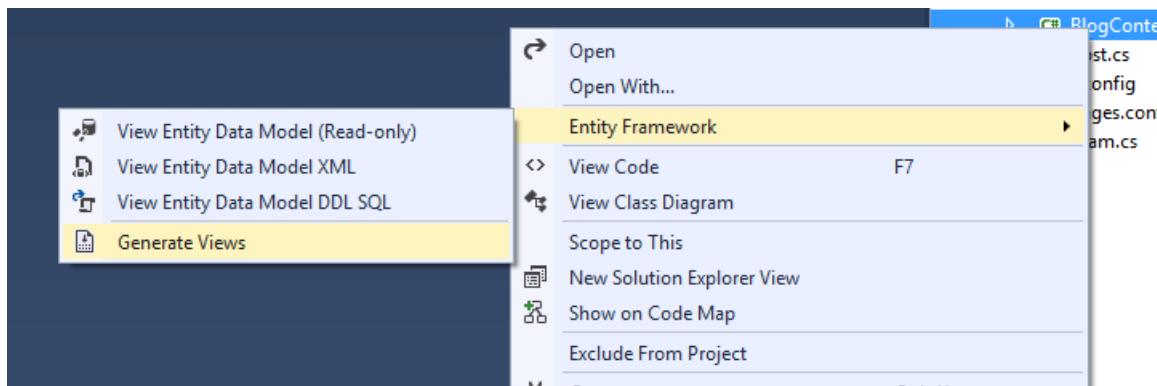
01.10.2018 • 7 minutes to read • [Edit Online](#)

Zanim Entity Framework można wykonać zapytania i zapisać zmiany w źródle danych, go wygenerować zestaw widoków mapowania dostępu do bazy danych. Widoki te mapowania są zbiór instrukcji SQL jednostki, która reprezentuje bazy danych w sposób abstrakcyjny i są częścią metadanych, które są buforowane dla domeny aplikacji. Jeśli tworzysz wiele wystąpień tego samego kontekstu w tej samej domenie aplikacji, będą ponownie używane widokach mapowania z pamięci podrzcznej metadanych zamiast ponownego generowania ich. Ponieważ generowanie Widok mapowania jest znaczna część całkowity koszt wykonania pierwszego zapytania, platformy Entity Framework umożliwiają wstępnie wygenerować widokach mapowania i umieścić je w projekcie skompilowany. Aby uzyskać więcej informacji, zobacz [zagadnienia związane z wydajnością \(Entity Framework\)](#).

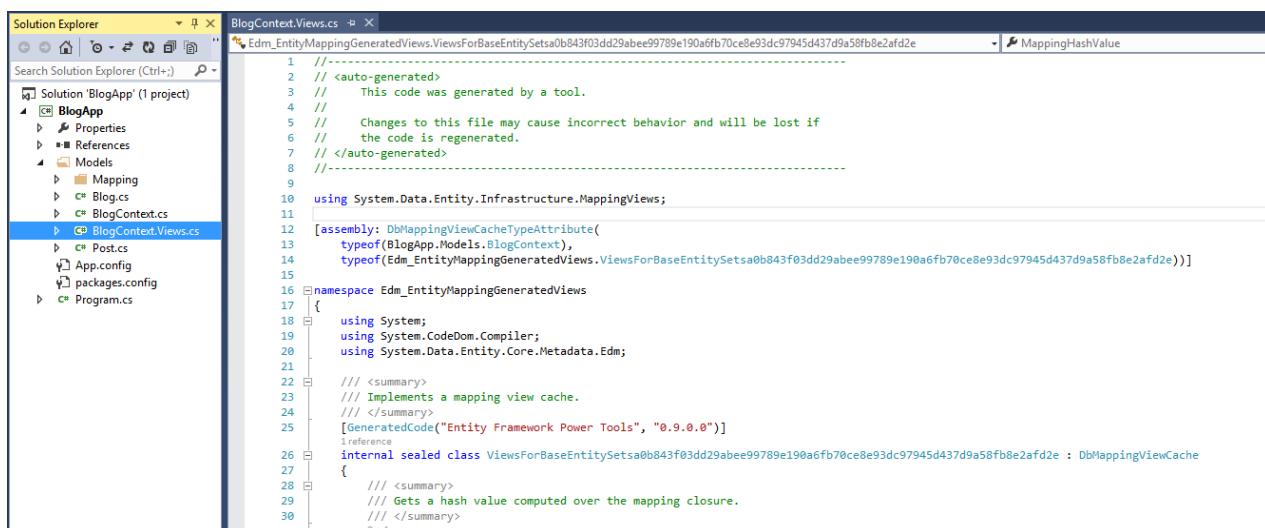
Generowanie mapowanie widoków przy użyciu programu EF Power Tools Community Edition

Najłatwiej można wstępnie wygenerować widoków jest użycie [EF Power Tools Community Edition](#). Po utworzeniu zainstalowano narzędzia zasilania masz opcję menu, aby wygenerować widoki, zgodnie z poniższymi instrukcjami.

- Aby uzyskać **Code First** modeli kliknij prawym przyciskiem myszy plik kodu zawierający klasy DbContext.
- Aby uzyskać **projektancie platformy EF** modeli kliknij prawym przyciskiem myszy w pliku EDMX.



Po zakończeniu procesu będą dostępne podobny do następującego wygenerowane klasy



Teraz po uruchomieniu aplikacji EF użyje tej klasy można załadować widoki, zgodnie z potrzebami. W przypadku zmiany modelu i ponownie generuje ta klasa EF spowoduje zgłoszenie wyjątku.

Generowanie widoków mapowania z kodu — od wersji EF6

Inny sposób, aby wygenerować widoków jest korzystanie z interfejsów API, który zapewnia EF. Przy użyciu tej metody należy swobody serializacji widoków, jednak lubisz, ale trzeba będzie również załadować widoków samodzielnie.

NOTE

EF6 poczawszy tylko — interfejsy API, przedstawione w tej sekcji zostały wprowadzone w programie Entity Framework 6. Jeśli używasz starszej wersji tych informacji nie ma zastosowania.

Generowanie widoków

Interfejsy API, aby wygenerować widoki znajdują się w klasie `System.Data.Entity.Core.Mapping.StorageMappingItemCollection`. Aby pobrać `StorageMappingCollection` dla kontekstu, za pomocą obiektu `MetadataWorkspace` dla obiektu `ObjectContext`. Jeśli używasz nowszej interfejsu API typu `DbContext`, wówczas użytkownik może uzyskać dostęp do tego za pomocą `IObjectContextAdapter`, takich jak poniżej, w tym kodzie mamy wystąpienie usługi `DbContext` pochodnej wywołuje `dbContext`:

```
var objectContext = ((IObjectContextAdapter) dbContext).ObjectContext;
var mappingCollection = (StorageMappingItemCollection)objectContext.MetadataWorkspace
    .GetItemCollection(DataSpace.CSSpace);
```

Po utworzeniu obiekt `StorageMappingItemCollection` można uzyskać dostęp do metod `GenerateViews` i `ComputeMappingHashValue`.

```
public Dictionary<EntitySetBase, DbMappingView> GenerateViews(IList<EdmSchemaError> errors)
public string ComputeMappingHashValue()
```

Pierwsza metoda tworzy słownik z wpis dla każdego widoku w mapowaniu kontenera. Druga metoda oblicza wartość skrótu dla mapowania jeden kontener i jest używany w czasie wykonywania do sprawdzania poprawności, model nie zmienił się od czasu widoki zostały wstępnie wygenerowane. Zastąpienia dwie metody są udostępniane dla złożonych scenariuszy obejmujących wiele mapowań kontenera.

Podczas generowania widoków spowoduje wywołanie metody `GenerateViews` i następnie zapisać wynikowy `EntitySetBase` i `DbMappingView`. Należy również przechowywać wyznaczania wartości skrótu, generowany przez metodę `ComputeMappingHashValue`.

Trwa ładowanie widoków

Aby załadować widoki generowane przez metodę `GenerateViews`, możesz zapewnić EF klasę, która dziedziczy z klasy abstrakcyjnej `DbMapViewCache`. `DbMapViewCache` określa dwie metody, które należy zaimplementować:

```
public abstract string MappingHashValue { get; }
public abstract DbMapView GetView(EntitySetBase extent);
```

Właściwość `MappingHashValue` musi zwracać wyznaczania wartości skrótu, generowany przez metodę `ComputeMappingHashValue`. Gdy EF jest przejście do zadania dla widoków najpierw wygeneruje i porównania wartości skrótu modelu przy użyciu skrótu zwracane przez tę właściwość. Jeśli nie są zgodne EF spowoduje zgłoszenie wyjątku `EntityCommandCompilationException`.

Metoda `GetView` będzie akceptować `EntitySetBase` i muszą zwracać `DbMapView`, zawierający `EntitySql`, który został wygenerowany w tym został skojarzony z danym `EntitySetBase` w słowniku generowany przez metodę `GenerateViews`. Jeśli EF prosi o widoku, że nie masz następnie `GetView` powinna zwrócić wartość `null`.

Poniżej przedstawiono wyciąg z DbMappingViewCache, który jest generowany przy użyciu zaawansowanych narzędzi, jak opisano powyżej, w nim widać sposób przechowywania i pobierania EntitySql wymagane.

```
public override string MappingHashValue
{
    get { return "a0b843f03dd29abee99789e190a6fb70ce8e93dc97945d437d9a58fb8e2afed2e"; }
}

public override DbMappingView GetView(EntitySetBase extent)
{
    if (extent == null)
    {
        throw new ArgumentNullException("extent");
    }

    var extentName = extent.EntityContainer.Name + "." + extent.Name;

    if (extentName == "BlogContext.Blogs")
    {
        return GetView2();
    }

    if (extentName == "BlogContext.Posts")
    {
        return GetView3();
    }

    return null;
}

private static DbMappingView GetView2()
{
    return new DbMappingView(@"
        SELECT VALUE -- Constructing Blogs
        [BlogApp.Models.Blog](T1.Blog_BlogId, T1.Blog_Test, T1.Blog_title, T1.Blog_Active,
        T1.Blog_SomeDecimal)
        FROM (
        SELECT
            T.BlogId AS Blog_BlogId,
            T.Test AS Blog_Test,
            T.title AS Blog_title,
            T.Active AS Blog_Active,
            T.SomeDecimal AS Blog_SomeDecimal,
            True AS _from0
        FROM CodeFirstDatabase.Blog AS T
        ) AS T1");
}
```

Mają zastosowanie do programów EF swoje DbMappingViewCache dodasz Użyj DbMappingViewCacheTypeAttribute, określając kontekst, który został utworzony dla. W poniższym kodzie za pomocą klasy MyMapViewCache wnoszonym BlogContext.

```
[assembly: DbMappingViewCacheType(typeof(BlogContext), typeof(MyMapViewCache))]
```

W przypadku bardziej złożonych scenariuszy wystąpienia pamięci podręcznej Widok mapowania można podać, określając fabrykę pamięci podręcznej Widok mapowania. Można to zrobić poprzez implementację klasy abstrakcyjnej System.Data.Entity.Infrastructure.MappingViews.DbMappingViewCacheFactory. Wystąpienie fabryki pamięci podręcznej Widok mapowania, która jest używana można pobrać lub ustawić za pomocą StorageMappingItemCollection.MappingViewCacheFactoryproperty.

Dostawcy programu Entity Framework 6

25.10.2018 • 7 minutes to read • [Edit Online](#)

NOTE

EF6 począwszy tylko — funkcje, interfejsów API itp. z opisem na tej stronie zostały wprowadzone w programie Entity Framework 6. Jeśli używasz starszej wersji, niektóre lub wszystkie informacje, nie ma zastosowania.

Entity Framework jest obecnie opracowywany w ramach licencji open source i EF6 i powyżej nie zostanie dostarczona w ramach programu .NET Framework. Ma wiele zalet, ale wymaga również odbudowany EF dostawców dla zestawów platformy EF6. Oznacza to, że EF dostawców dla EF5, jak i poniżej nie będą działać przy użyciu platformy EF6, dopóki ich ponownej komplikacji.

Które są dostępni dostawcy dla platformy EF6?

Sobie sprawę z dostawcami, ponownej komplikacji dla platformy EF6 obejmują:

- **Dostawca programu Microsoft SQL Server**
 - Utworzona na podstawie [Entity Framework Otwórz bazy kodu źródłowego](#)
 - Dostarczany jako część [pakietu NuGet platformy EntityFramework](#)
- **Dostawca programu Microsoft SQL Server Compact Edition**
 - Utworzona na podstawie [Entity Framework Otwórz bazy kodu źródłowego](#)
 - Dostarczana z [pakietu EntityFramework.SqlServerCompact NuGet](#)
- **Devart dotConnect dostawców danych**
 - Brak dostawców innych firm [Devart](#) dla wielu baz danych w tym Oracle, MySQL, PostgreSQL, SQLite, Salesforce, DB2 i programu SQL Server
- **Dostawców oprogramowania CData**
 - Brak dostawców innych firm [oprogramowania CData](#) z różnych magazynów danych, w tym Salesforce, usługa Azure Table Storage, MySql i wielu innych
- **Dostawca firebird**
 - Dostępne jako [pakietu NuGet](#)
- **Visual Fox Pro dostawcy**
 - Dostępne jako [pakietu NuGet](#)
- **MySQL**
 - [MySQL Connector/Net](#)
- **PostgreSQL**
 - Npgsql jest dostępna jako [pakietu NuGet](#)
- **Oracle**
 - ODP.NET jest dostępna jako [pakietu NuGet](#)

Należy pamiętać, włączenia na tej liście nie wskazuje poziom funkcjonalności lub pomocy technicznej dla danego dostawcy, tylko że komplikacji dla platformy EF6 została udostępniona.

Rejestrowanie dostawców EF

Począwszy od dostawcy programu Entity Framework 6 EF można zarejestrować za pomocą konfiguracji albo oparte na kodzie lub w pliku konfiguracji aplikacji.

Rejestracja pliku konfiguracji

Rejestracja dostawcy EF w pliku app.config lub web.config ma następujący format:

```
<entityFramework>
  <providers>
    <provider invariantName="My.Invariant.Name" type="MyProvider.MyProviderServices, MyAssembly" />
  </providers>
</entityFramework>
```

Należy pamiętać, że często jeśli dostawca programu EF jest instalowany z pakietów NuGet, a następnie pakietu NuGet spowoduje automatyczne dodanie tej rejestracji do pliku konfiguracji. Po zainstalowaniu pakietu NuGet do projektu, który nie jest projektem startowym aplikacji, należy skopiować rejestracji do pliku konfiguracji dla Twój projekt startowy.

"Invariantname" w tym rejestracji jest taka sama nazwa niezmienna, używany do identyfikowania dostawcy ADO.NET. Ten znajduje się jako atrybut "niezmiennej" w rejestracji DbProviderFactories, a atrybut "providerName" podczas rejestracji ciągu połączenia. Niezmienna Nazwa do użycia powinny też obejmować w dokumentacji dostawcy. Przykłady nazw niezmiennej są "System.Data.SqlClient" dla programu SQL Server i "System.Data.SqlServerCe.4.0" dla programu SQL Server Compact.

"Type" w tym rejestracji jest nazwa kwalifikowanego dla zestawu typu dostawcy, która dziedziczy po "System.Data.Entity.Core.Common.DbProviderServices". Na przykład ciąg do użycia dla programu SQL Compact jest "System.Data.Entity.SqlServerCompact.SqlCeProviderServices EntityFramework.SqlServerCompact". Typ do użycia w tym miejscu powinny być objęte dokumentacją dostawcy.

Oparte na kodzie rejestracji

Począwszy od platformy Entity Framework 6 konfiguracji całej aplikacji na platformie EF można określić w kodzie. Szczegółowe informacje można znaleźć [konfiguracji Entity Framework Code-Based](#). Normalny sposób, aby zarejestrować dostawcę EF przy użyciu konfiguracji na podstawie kodu jest Utwórz nową klasę dziedziczącą po System.Data.Entity.DbConfiguration i umieść go w tym samym zestawie jako swojej klasy DbContext. Klasa DbConfiguration należy następnie zarejestruj dostawcę w jego konstruktorze. Na przykład aby zarejestrować SQL Compact dostawcy klasy DbConfiguration wygląda następująco:

```
public class MyConfiguration : DbConfiguration
{
    public MyConfiguration()
    {
        SetProviderServices(
            SqlCeProviderServices.ProviderInvariantName,
            SqlCeProviderServices.Instance);
    }
}
```

W tym kodzie "SqlCeProviderServices.ProviderInvariantName" to wygodne dla programu SQL Server Compact ciągu niezmienna Nazwa dostawcy ("System.Data.SqlClient") i SqlCeProviderServices.Instance zwraca pojedyncze wystąpienie SQL Compact Dostawca EF.

Co się stanie, jeśli dostawca potrzebuję nie jest dostępna?

Jeśli dostawca jest dostępny we wcześniejszych wersjach programu EF, następnie zachęcamy Cię do skontaktuj się z właścicielem dostawcy i poproś go o utworzenia wersji EF6. Należy dołączyć odwołanie do [dokumentacji dla modelu dostawcy EF6](#).

Można napisać dostawcę samodzielnie?

Oczywiście prawdopodobnie utworzyć dostawcę EF samodzielnie, mimo że nie należy rozważyć trivial przedsiębiorstwa. Powyższe łącze o modelu dostawcy EF6 jest dobrym miejscem do rozpoczęcia. Możesz również przydatne może okazać się do użycia kodu dla programu SQL Server i SQL CE dostawcy uwzględnione w [codebase "open source" EF](#) jako punktu wyjścia lub odwołania.

Należy pamiętać, że począwszy od platformy EF6 dostawcy EF jest mniej ściśle powiązany źródłowy dostawca ADO.NET. Ta funkcja ułatwia pisanie dostawcy programu EF bez konieczności pisania lub opakowywania klas ADO.NET.

Dostawca modelu Entity Framework 6

25.10.2018 • 28 minutes to read • [Edit Online](#)

Model dostawcy programu Entity Framework umożliwia Entity Framework ma być używany z różnymi typami serwera bazy danych. Na przykład jeden dostawca można podłączyć umożliwia EF ma być używany dla programu Microsoft SQL Server, podczas gdy umożliwia EF ma być używany dla programu Microsoft SQL Server Compact Edition można podłączyć do innego dostawcy. Dostawcy dla platformy EF6, które sobie sprawę z znajduje się na [dostawcy programu Entity Framework](#) strony.

Niektóre zmiany były wymagane do metody, gdy EF wchodzi w interakcję z dostawcami, aby umożliwić EF mogą być wprowadzane w ramach licencji open source. Te zmiany wymagają odbudowywania EF dostawców dla zestawów platformy EF6 wraz z nowe mechanizmy Rejestracja dostawcy.

Ponowne tworzenie

Przy użyciu platformy EF6 kodu core, który został wcześniej częścią programu .NET Framework jest teraz są dostarczane jako poza pasmem (OOB), zestawy. Szczegółowe informacje na temat tworzenia aplikacji w ramach platformy EF6 znajduje się na [aktualizowania aplikacji dla platformy EF6](#) strony. Dostawców będzie również trzeba odbudować, korzystając z tych instrukcji.

Przegląd typów dostawcy

Dostawca usługi EF jest naprawdę kolekcja właściwe dla dostawcy usług zdefiniowane przez typy CLR, które tych usług rozszerza z (dla klasy bazowej) ani nie implementuje (w przypadku interfejsu). Są dwa z tych usług podstawowych i niezbędne dla platformy EF w ogóle działać. Inne są opcjonalne i wystarczy do zaimplementowania. Jeśli określonych jest wymagane i/lub domyślnej implementacji tych usług nie działa dla konkretnej bazy danych serwer docelowy.

Typy podstawowe dostawców

DbProviderFactory

EF zależy od tego, typ pochodzący od posiadania [System.Data.Common.DbProviderFactory](#) do wykonywania wszystkich dostęp do niskiego poziomu bazy danych. DbProviderFactory nie jest częścią platformy EF, lecz dotyczy klasy w .NET Framework, która służy punkt wejścia dla dostawcy ADO.NET, który może służyć przez EF, O/RMs lub bezpośrednio przez aplikację można uzyskać rodzajami połączeń, polecenia, parametry i inne abstrakcje ADO.NET dostawcy sposób niezależny od. Więcej informacji na temat DbProviderFactory można znaleźć w [dokumentacji MSDN dotyczącej ADO.NET](#).

DbProviderServices

EF zależy od tego, mające typ pochodzący od DbProviderServices zapewniające dodatkowe funkcje wymagane przez EF na podstawie funkcje już udostępniane przez dostawcę danych ADO.NET. W starszych wersjach programu EF klasy DbProviderServices było częścią programu .NET Framework i zostało znaleziony w przestrzeni nazw System.Data.Common. Począwszy od platformy EF6 tej klasy jest teraz częścią EntityFramework.dll i znajduje się w przestrzeni nazw System.Data.Entity.Core.Common.

Więcej informacji na temat podstawowych funkcji implementacji DbProviderServices znajduje się na [MSDN](#). Jednak należy pamiętać, że od czasu zapisania tych informacji nie jest aktualizowana dla platformy EF6 mimo że większość koncepcji są nadal ważne. Implementacji programu SQL Server i programu SQL Server Compact DbProviderServices również są sprawdzane w celu [bazy kodu typu open-source](#) i może służyć jako przydatne dane dla innych implementacji.

W starszych wersjach programu EF implementacji DbProviderServices używać pochodzi bezpośrednio od dostawcy ADO.NET. Zostało to zrobione, rzutowanie DbProviderFactory na IServiceProvider i wywołanie metody GetService. To ściśle powiązane dostawcy EF DbProviderFactory. Ta sprzężenia zablokowany EF jest przenoszony z .NET Framework i w związku z tym dla platformy EF6 to ścisłego sprzężenia została usunięta, a implementacja DbProviderServices jest obecnie zarejestrowany bezpośrednio w pliku konfiguracji aplikacji lub oparte na kodzie Konfiguracja opisany bardziej szczegółowo *rejestrowanie DbProviderServices* poniższej sekcji.

Usługi dodatkowe

Oprócz podstawowych usług opisanych powyżej są również wiele innych usług używanych przez EF, które są zawsze lub czasami specyficzne dla dostawcy. Domyślnej implementacji właściwe dla dostawcy, te usługi mogą być dostarczane przez implementację DbProviderServices. Aplikacje można również zastąpić implementacjami tych usług lub dostarczać implementacje, gdy typem DbProviderServices nie dostarcza domyślny. Jest to opisane bardziej szczegółowo w *rozpoznawanie dodatkowych usług* poniższej sekcji.

Poniżej przedstawiono typy dodatkowych usług, które dostawcy mogą być przydatne do dostawcy. Więcej szczegółów na temat każdego z tych typów usług można znaleźć w dokumentacji interfejsu API.

IDbExecutionStrategy

Jest to opcjonalna usługa, która umożliwia dostawcy zaimplementować ponownych prób lub inne zachowanie, gdy zapytań i poleceń, które są wykonywane względem bazy danych. Jeśli nie dostarczono żadnej implementacji, EF będzie po prostu wykonaj polecenia i propagowanie wyjątki zgłaszone. Dla programu SQL Server ta usługa służy do zapewnienia zasad ponawiania, co jest szczególnie przydatne, jeśli działających w odniesieniu do serwerów bazy danych opartej na chmurze, takich jak SQL Azure.

IDbConnectionFactory

Jest to opcjonalna usługa, która umożliwia dostawcy utworzyć obiekty DbConnection zgodnie z Konwencją, gdy podana nazwa bazy danych. Należy pamiętać, że gdy ta usługa może zostać rozpoznana przez implementację DbProviderServices została już obecne od EF 4.1 i może również być jawnie ustawione w pliku konfiguracji lub w kodzie. Dostawca tylko otrzymają Państwo rozpoznać tej usługi, jeśli on zarejestrowany jako domyślnego dostawcę (zobacz *domyślny dostawca* poniżej) i jeśli domyślna fabryka połączenia nie został ustawiony gdzie indziej.

DbSpatialServices

Jest to opcjonalne usługi, które umożliwia dostawcy dodać obsługę typów przestrzennych geometry i położenia geograficznego. Implementacja tej usługi należy podać w kolejności dla aplikacji EF za pomocą typów przestrzennych. DbSptialServices zostanie poproszony o na dwa sposoby. Po pierwsze, właściwe dla dostawcy usług przestrzennych, są żądane przy użyciu obiektu DbProviderInfo (zawierający niezmiennej token nazwy, a manifestu) jako klucza. Po drugie DbSpatialServices można monit o wpisanie bez klucza. Służy to rozwiązać "globalne przestrzenne dostawcę" użytą podczas tworzenia autonomicznego DbGeography lub DbGeometry typów.

MigrationSqlGenerator

Jest to opcjonalna usługa, która umożliwia migracji EF, służący do generowania SQL używane podczas tworzenia i modyfikowania schematy bazy danych przez rozwiązanie Code First. Implementacja jest wymagana w celu zapewnienia obsługi migracji. Jeśli podano implementację go będzie również użyte podczas tworzenia bazy danych przy użyciu metody Database.Create lub inicjatory bazy danych.

FUNC < DbConnection, string, HistoryContextFactory >

Jest to opcjonalna usługa, która umożliwia dostawcy skonfigurować mapowanie HistoryContext do `__MigrationHistory` tabeli używanej przez migracje EF. HistoryContext jest pierwszy typu DbContext kodu i można skonfigurować przy użyciu normalnych wygodnego interfejsu API można zmienić elementów, takich jak nazwy tabeli i specyfikacji mapowania kolumn. Domyślna implementacja tej usługi dla wszystkich

dostawców zwracane przez EF mogą działać w przypadku serwera bazy danych, jeśli wszystkie domyślne tabel i kolumn mapowania są obsługiwane przez tego dostawcę. W takim przypadku dostawca nie musi dostarczyć implementację tej usługi.

IDbProviderFactoryResolver

Jest to opcjonalne zapewniającej uzyskiwanie DbProviderFactory poprawne z danego obiektu DbConnection. Domyślna implementacja tej usługi dla wszystkich dostawców zwracane przez EF jest przeznaczony do pracy dla wszystkich dostawców. Jednak podczas uruchamiania na .NET 4, DbProviderFactory nie jest dostępny publicznie w jeden jego DbConnections. W związku z tym EF używa niektóre heurystyki do wyszukiwania zarejestrowanych dostawców w celu znalezienia dopasowania. Istnieje możliwość, że w przypadku niektórych dostawców te algorytmy heurystyczne zakończy się niepowodzeniem, a w takich sytuacjach dostawcy należy podać nową metodę implementacji.

Rejestrowanie DbProviderServices

W pliku konfiguracji (app.config lub web.config) lub przy użyciu konfiguracji na podstawie kodu aplikacji można zarejestrować implementacji DbProviderServices do użycia. W obu przypadkach rejestracji używa dostawcy "niezmiennej nazwie" jako klucza. Dzięki temu wielu dostawów były rejestrowane i używane w jednej aplikacji. Niezmienna nazwa używana w przypadku rejestracji programu EF jest taka sama jak nazwa niezmienna, używana dla ciągów połączenia i Rejestracja dostawcy ADO.NET. Na przykład dla programu SQL Server niezmiennej nazwie "System.Data.SqlClient" jest używany.

Rejestracja pliku konfiguracji

Typ DbProviderServices do użycia jest zarejestrowany jako element dostawcę na liście dostawów entityFramework części pliku konfiguracji aplikacji. Na przykład:

```
<entityFramework>
  <providers>
    <provider invariantName="My.Invariant.Name" type="MyProvider.MyProviderServices, MyAssembly" />
  </providers>
</entityFramework>
```

Typu ciąg musi być typu kwalifikowanego zestawu nazwę implementacji DbProviderServices do użycia.

Oparte na kodzie rejestracji

Począwszy od platformy EF6 dostawców można również zarejestrować przy użyciu kodu. Dzięki temu dostawcy EF można używać bez zmian do pliku konfiguracji aplikacji. Używać konfiguracji oparte na kodzie aplikacji należy utworzyć klasę DbConfiguration, zgodnie z opisem w [dokumentacji oparty na kodzie konfiguracji](#). Konstruktor klasy DbConfiguration następnie należy wywołać SetProviderServices, aby zarejestrować dostawcę EF. Na przykład:

```
public class MyConfiguration : DbConfiguration
{
    public MyConfiguration()
    {
        SetProviderServices("My.New.Provider", new MyProviderServices());
    }
}
```

Rozpoznawanie dodatkowych usług

Jak wspomniano powyżej, w *dostawcy typów Przegląd* sekcji DbProviderServices klasy można również do rozpoznania usług dodatkowych. Jest to możliwe, ponieważ DbProviderServices implementuje IDbDependencyResolver i każdego zarejestrowanego typu DbProviderServices jest dodawany jako "domyślny

mechanizm rozwiązywania konfliktów". Mechanizm IDbDependencyResolver jest opisany bardziej szczegółowo w [Rozdziale zależności](#). Jednak nie jest niezbędnie zapoznać się z pojęciami w tej specyfikacji do rozpoznania usług dodatkowych, dostawcy.

Najczęstszym sposobem dostawcy do rozpoznania usług, dodatkowe jest wywołanie DbProviderServices.AddDependencyResolver dla każdej usługi w konstruktorze klasy DbProviderServices. Na przykład SqlProviderServices (Dostawca EF dla programu SQL Server) ma kod podobny do tego inicjowania:

```
private SqlProviderServices()
{
    AddDependencyResolver(new SingletonDependencyResolver<IDbConnectionFactory>(
        new SqlConnectionFactory()));

    AddDependencyResolver(new ExecutionStrategyResolver<DefaultSqlExecutionStrategy>(
        "System.data.SqlClient", null, () => new DefaultSqlExecutionStrategy()));

    AddDependencyResolver(new SingletonDependencyResolver<Func<MigrationSqlGenerator>>(
        () => new SqlServerMigrationSqlGenerator(), "System.data.SqlClient"));

    AddDependencyResolver(new SingletonDependencyResolver<DbSpatialServices>(
        SqISpatialServices.Instance,
        k =>
    {
        var asSpatialKey = k as DbProviderInfo;
        return asSpatialKey == null
            || asSpatialKey.ProviderInvariantName == ProviderInvariantName;
    }));
}
```

Ten konstruktor korzysta z następujących klas pomocniczych:

- SingletonDependencyResolver: zapewnia prosty sposób rozwiązania pojedynczego wystąpienia usług — oznacza to, usług, dla których tego samego wystąpienia jest zwracana zawsze nazwa GetService. Przejściowy usługi często są rejestrowane jako fabryki singleton, która będzie służyć do tworzenia wystąpienia przejściowego na żądanie.
- ExecutionStrategyResolver: program rozpoznawania nazw specyficzne dla zwracania IExecutionStrategy implementacji.

Zamiast używania DbProviderServices.AddDependencyResolver istnieje również możliwość zastąpienia DbProviderServices.GetService i rozwiązać dodatkowych usług bezpośrednio. Ta metoda zostanie wywołana, gdy EF potrzebuje usługa definiuje na podstawie określonego typu, a w niektórych przypadkach dla danego klucza. Metoda powinna zwrócić usługi, jeśli można lub zwraca wartość null, jak zrezygnować przekazujących usługi i zezwolić innej klasy go rozwiązać. Na przykład aby rozwiązać domyślną fabrykę połączenie kodu w GetService może wyglądać mniej więcej tak:

```
public override object GetService(Type type, object key)
{
    if (type == typeof(IDbConnectionFactory))
    {
        return new SqlConnectionFactory();
    }
    return null;
}
```

Kolejność rejestracji

Gdy wiele implementacji DbProviderServices są rejestrowane w pliku konfiguracyjnym aplikacji będzie można dodać jako dodatkowej rozpoznawania nazw w kolejności, w jakiej występują na liście. Ponieważ rozwiązujący zawsze są dodawane do góry łańcucha dodatkowej programu rozpoznawania nazw, oznacza to, że dostawcy

na końcu listy otrzymają Państwo rozpoznać zależności przed innymi. (To może wydawać się nieco counter-intuitive na początku, ale dobrym pomysłem Wyobraź sobie, biorąc każdy dostawca z listy i układania na podstawie istniejącego dostawcy).

Zazwyczaj ta kolejność nie ma znaczenia, ponieważ większość usług dostawcy są właściwe dla dostawcy i dostosowane przez nazwa niezmienna dostawcy. Jednak dla usług, są nie kluczach nazwa niezmienna dostawcy lub niektórych innych właściwe dla dostawcy klucz, który zostanie rozwiązyany usługi oparte na ta kolejność. Na przykład jeśli go nie jest jawnie określona inaczej gdzieś else, domyślnej fabryka połączenia będą pochodzić z dostawcę najwyższego poziomu w łańcuchu.

Rejestracje plików dodatkowych konfiguracji

Istnieje możliwość jawnie zarejestrować niektórych usług dodatkowego dostawcę opisanych powyżej bezpośrednio w pliku konfiguracji aplikacji. Po zakończeniu to rejestrowania w pliku konfiguracji będą używane zamiast niczego zwracany przez metodę GetService implementacji DbProviderServices.

Rejestrowanie domyślnej fabryki połączenia

Począwszy od EF5 pakiet NuGet platformy EntityFramework automatycznie zarejestrowana fabryka połączenia programu SQL Express lub LocalDb fabryka połączenia w pliku konfiguracji.

Na przykład:

```
<entityFramework>
  <defaultConnectionFactory type="System.Data.Entity.Infrastructure.SqlConnectionFactory, EntityFramework"
  >
</entityFramework>
```

Typu jest nazwą typu kwalifikowanego zestawu usługi fabryka połączenia domyślnej musi implementować IDbConnectionFactory.

Zalecane jest dostawcy pakietu NuGet ustawienie domyślnej fabryki połączenia w ten sposób podczas instalacji. Zobacz pakietów NuGet dla dostawców poniżej.

Dodatkowe zmiany dostawcy EF6

Zmiany dostawcy przestrzenne

Dostawcy, obsługujące typów przestrzennych teraz należy zaimplementować pewne dodatkowe metody na klasy wywodzące się z DbSpatialDataReader:

- `public abstract bool IsGeographyColumn(int ordinal)`
- `public abstract bool IsGeometryColumn(int ordinal)`

Dostępne są również nowe wersje asynchroniczne istniejących metod, które są zalecane do zastąpienia domyślnej implementacji delegowania do metod synchronicznych i dlatego nie są wykonywane asynchronicznie:

- `public virtual Task<DbGeography> GetGeographyAsync(int ordinal, CancellationToken cancellationToken)`
- `public virtual Task<DbGeometry> GetGeometryAsync(int ordinal, CancellationToken cancellationToken)`

Natywna obsługa Enumerable.Contains

EF6 wprowadza nowy typ wyrażenia DbInExpression, który został dodany, aby rozwiązać problemy z wydajnością dotyczące użytkowania Enumerable.Contains w kwerendach LINQ. Klasa DbProviderManifest ma nowej metody wirtualnej SupportsInExpression, które jest wywoływanie przez EF, aby określić, jeśli dostawca obsługuje nowy typ wyrażenia. Dla zachowania zgodności z istniejącej implementacji dostawcy, metoda zwraca wartość false. Aby korzystać z tego ulepszenia, dostawcę EF6 można dodać kod do obsługi

`DbInExpression` i zastąpić `SupportsInExpression` aby zwracała wartość `true`. Przez wywołanie metody `DbExpressionBuilder.In`, można utworzyć wystąpienia elementu `DbInExpression`. Wystąpienie `DbInExpression` składa się z `DbExpression`, zwykle reprezentujący kolumnę tabeli oraz listę `DbConstantExpression` do testowania pod kątem dopasowania.

Pakiety NuGet dla dostawców

Jednym ze sposobów, aby udostępnić dostawcę EF6 jest zwolnić go jako pakiet NuGet. Przy użyciu pakietu NuGet ma następujące zalety:

- Jest łatwa w użyciu pakietu NuGet, aby dodać rejestrację dostawcy do pliku konfiguracji aplikacji
- Dodatkowych zmian do pliku konfiguracji można ustawić domyślną fabrykę połączenia tak, aby nawiązać połączenia przy Konwencji użyje zarejestrowanego dostawcy
- NuGet obsługuje dodanie przekierowań powiązań, aby dostawca EF6 będą nadal działać, nawet po opublikowaniu nowego pakietu EF

Na przykład to pakiet `EntityFramework.SqlServerCompact`, który jest dostępny w "[open source codebase](#)". Ten pakiet zawiera dobre szablon do tworzenia dostawcy EF pakietów NuGet.

Polecenia programu PowerShell

Po zainstalowaniu pakietu EntityFramework NuGet rejestruje moduł programu PowerShell, który zawiera dwa polecenia, które są bardzo przydatne w przypadku pakietów dostawcy:

- Dodaj `EFProvider` dodaje nową jednostkę dla dostawcy w pliku konfiguracji projektu docelowego i upewnia się, że jest na końcu listę zarejestrowanych dostawców.
- Dodaj `EFDefaultConnectionFactory` dodaje lub aktualizuje rejestracji `defaultConnectionFactory` w pliku konfiguracji projektu docelowego.

Oba te polecenia zajmie się dodanie sekcji `entityFramework` do pliku konfiguracji i dodawania kolekcji dostawców, jeśli to konieczne.

Jest ona przeznaczona, że te polecenia można wywołać z `install.ps1` skryptu NuGet. Na przykład `install.ps1` dla dostawcy SQL Compact wygląda podobnie do następującej:

```
param($installPath, $toolsPath, $package, $project)
Add-EFDefaultConnectionFactory $project 'System.Data.Entity.Infrastructure.SqlCeConnectionFactory,
EntityFramework' -ConstructorArguments 'System.Data.SqlClient.4.0'
Add-EFProvider $project 'System.Data.SqlClient.4.0'
'System.Data.Entity.SqlClient.SqlCeProviderServices, EntityFramework.SqlServerCompact'</pre>
```

Więcej informacji na temat tych poleceń można uzyskać przy użyciu `get-help` w oknie Konsola Menedżera pakietów.

Zawijanie dostawców

Dostawca zawijania to dostawcy EF i/lub ADO.NET, który otacza istniejącego dostawcy rozszerzać go za pomocą innych funkcji, takie jak profilowania, lub śledzenia możliwości. Zawijanie dostawcy mogą być rejestrowane w normalny sposób, ale jest często bardziej wygodne skonfigurować dostawcę zawijania w czasie wykonywania, przechwytyując rozwiązania związane z dostawcą usług. `OnLockingConfiguration` zdarzeń statycznych w klasie `DbConfiguration` można to zrobić.

`OnLockingConfiguration` jest wywoływana po EF stwierdził, gdzie otrzymany EF konfiguracji dla domeny aplikacji, ale zanim zostanie zablokowane do użycia. Przy uruchamianiu aplikacji (przed użyciem EF) aplikacji należy zarejestrować program obsługi zdarzeń dla tego zdarzenia. (Są rozważane, dodanie obsługi rejestrowania ten program obsługi w pliku konfiguracji, ale to nie jest jeszcze obsługiwane). Program obsługi

zdarzeń powinny następnie wywoływanie ReplaceService dla każdej usługi, który ma zostać opakowany.

Na przykład aby zawijać IDbConnectionFactory i DbProviderService, powinny być rejestrowane obsługą podobnie do następującej:

```
DbConfiguration.OnLockingConfiguration +=  
    (_, a) =>  
    {  
        a.ReplaceService<DbProviderServices>(  
            (s, k) => new MyWrappedProviderServices(s));  
  
        a.ReplaceService<IDbConnectionFactory>(  
            (s, k) => new MyWrappedConnectionFactory(s));  
    };
```

Usługa, która został rozwiązyany i teraz powinna być otoczona wraz z kluczem, który został użyty do rozwiązania usługi są przekazywane do programu obsługi. Program obsługi można opakować tę usługę i Zastąp zwrócone usługi przy użyciu opakowanej wersji.

Rozpoznawanie DbProviderFactory przy użyciu programu EF

DbProviderFactory jest jednym z typów podstawowych dostawcy wymagane przez EF, zgodnie z opisem w *dostawcy typów Przegląd powyższej sekcji*. Jak już wspomniano, nie jest typem EF i rejestracji zazwyczaj nie jest częścią konfiguracji EF, ale zamiast tego jest normalne rejestrację dostawcy ADO.NET w pliku machine.config lub pliku konfiguracji aplikacji.

Pomimo tego EF nadal używa jej mechanizm rozpoznawania zależności normalne podczas wyszukiwania DbProviderFactory do użycia. Domyślny mechanizm rozwiązywania konfliktów używa normalnej rejestracji ADO.NET w plikach konfiguracji, a więc jest to zazwyczaj niewidoczny. Jednak ze względu na rozwiązanie normalne zależności jest używany mechanizm oznacza, że IDbDependencyResolver może służyć do rozwiązyania DbProviderFactory, nawet wtedy, gdy nie została wykonana normalnej rejestracji ADO.NET.

Rozpoznawanie DbProviderFactory w ten sposób ma kilka skutki:

- Aplikację przy użyciu konfiguracji na podstawie kodu można dodać wywołania w klasie ich DbConfiguration w taki sposób, aby zarejestrować DbProviderFactory odpowiednio. Jest to szczególnie przydatne w przypadku aplikacji, które nie mają być (lub nie) należy użyć dowolnej z opartej na plikach konfiguracji na wszystkich.
- Usługa może zostać zawiązany lub zastąpić, korzystając z ReplaceService zgodnie z opisem w *zawijania dostawców* powyższej sekcji
- Teoretycznie implementacji DbProviderServices można rozpoznać DbProviderFactory.

Istotną kwestią należy zwrócić uwagę na sposób żadnej z tych czynności jest, że wpływają one tylko wyszukiwania DbProviderFactory przez EF. Inny kod bez EF mogą nadal oczekiwany dostawcy ADO.NET do zarejestrowania się w zwykły sposób i może się nie powieść, jeśli rejestracja nie zostanie znaleziony. Z tego powodu jest zwykle lepszym miejscem dla DbProviderFactory do zarejestrowania się w zwykły sposób ADO.NET.

Powiiązane usługi

Jeśli EF jest używany do rozpoznawania DbProviderFactory, również go powinno rozwiązać IProviderInvariantName i IDbProviderFactoryResolver usług.

IProviderInvariantName to usługa, która jest używana do określenia dla danego typu DbProviderFactory nazwę niezmienną dostawcy. Domyślna implementacja tej usługi używa rejestracji dostawcy ADO.NET. Oznacza to, że jeśli dostawcy ADO.NET nie jest zarejestrowany w normalny sposób, ponieważ DbProviderFactory jest rozwiązywany za EF, następnie również będzie wymagany do rozwiązania tej usługi.

Należy pamiętać, że program rozpoznawania nazw dla tej usługi jest automatycznie dodawane, korzystając z metody `DbConfiguration.SetProviderFactory`.

Zgodnie z opisem w *dostawcy typów Przegląd Powyższa* sekcja, `IDbProviderFactoryResolver` jest używany do uzyskiwania poprawne `DbProviderFactory` z danego obiektu `DbConnection`. Domyślna implementacja tej usługi podczas uruchamiania na .NET 4 używa rejestracji dostawcy ADO.NET. Oznacza to, że jeśli dostawcy ADO.NET nie jest zarejestrowany w normalny sposób, ponieważ `DbProviderFactory` jest rozwiązywany za EF, następnie również będzie wymagany do rozwiązania tej usługi.

Obsługa dostawców dla typów przestrzennych

25.10.2018 • 4 minutes to read • [Edit Online](#)

Entity Framework obsługuje pracę z danymi przestrzennymi za pośrednictwem DbGeography lub DbGeometry klasy. W ramach tych zajęć, zależą od funkcji specyficznych dla bazy danych udostępnianymi przez dostawcę programu Entity Framework. Nie wszyscy dostawcy obsługują dane przestrzenne i tych, które wykonują mogą mieć dodatkowe wymagania wstępne, takich jak instalacja zestawów typów przestrzennych. Więcej informacji na temat Obsługa dostawców dla typów przestrzennych znajduje się poniżej.

Dodatkowe informacje dotyczące sposobu używania typów przestrzennych w aplikacji znajdują się w dwóch wskazówkach, jeden dla Code First, drugie dla pierwszej bazy danych lub pierwszego modelu:

- [Typy w kodzie najpierw danych przestrzennych](#)
- [Typy danych przestrzennych w Projektancie platformy EF](#)

Wersje programu EF, które obsługują typów przestrzennych

Obsługa typów przestrzennych została wprowadzona w EF5. Jednak w EF5 typów przestrzennych są obsługiwane tylko w przypadku aplikacji jest przeznaczony dla i działa w .NET 4.5.

Począwszy od platformy EF6 typów przestrzennych są obsługiwane w przypadku aplikacji przeznaczonych dla platformy .NET 4.5 i .NET 4.

EF dostawcy, obsługujące typów przestrzennych

EF5

Dostawcy programu Entity Framework do EF5, które sobie sprawę z czy typów przestrzennych pomocy technicznej:

- Dostawca programu Microsoft SQL Server
 - Ten dostawca jest dostarczany jako część EF5.
 - Ten dostawca jest zależna od niektórych dodatkowych bibliotek niskiego poziomu, które muszą zostać zainstalowane — Zobacz szczegóły poniżej.
- [Devart dotConnect na oprogramowanie Oracle](#)
 - Jest to dostawca innych firm z Devart.

Jeśli znasz EF5 dostawcy obsługująca typów przestrzennych następnie można uzyskać w kontakcie i będziemy wszystkiego dodać go do tej listy.

EF6

Dostawcy programu Entity Framework, dla platformy EF6, które sobie sprawę z czy typów przestrzennych pomocy technicznej:

- Dostawca programu Microsoft SQL Server
 - Ten dostawca jest dostarczany jako część platformy EF6.
 - Ten dostawca jest zależna od niektórych dodatkowych bibliotek niskiego poziomu, które muszą zostać zainstalowane — Zobacz szczegóły poniżej.
- [Devart dotConnect na oprogramowanie Oracle](#)
 - Jest to dostawca innych firm z Devart.

Jeśli znasz EF6 dostawcy obsługująca typów przestrzennych następnie można uzyskać w kontakcie i będziemy

wszystkiego dodać go do tej listy.

Wymagania wstępne dotyczące typów przestrzennych z programem Microsoft SQL Server

Obsługa przestrzenne programu SQL Server jest zależna od niskiego poziomu, specyficzne dla programu SQL Server typu SqlGeography i SqlGeometry. Te typy na żywo w zestawie Microsoft.SqlServer.Types.dll i ten zestaw nie jest dostarczany jako część EF lub w ramach programu .NET Framework.

Po zainstalowaniu programu Visual Studio zostanie często również zainstalowana wersja programu SQL Server i będzie to obejmowało instalacji Microsoft.SqlServer.Types.dll.

Jeśli program SQL Server nie jest zainstalowany na komputerze, na których chcesz użyć typów przestrzennych lub typów przestrzennych zostały wykluczone z instalacji programu SQL Server, należy je zainstalować ręcznie. Typy można zainstalować przy użyciu `sqlsysClrTypes.msi`, który jest częścią programu Microsoft SQL Server Feature Pack. Typów przestrzennych programu SQL Server specyficzny dla wersji są, tak więc zaleca się [Wyszukaj "SQL Server Feature Pack"](#) w programie Microsoft Download Center, a następnie wybierz i Pobierz opcji, która odnosi się do wersji programu SQL Server, które będą używane.

Praca z serwerów proxy

13.09.2018 • 4 minutes to read • [Edit Online](#)

Podczas tworzenia wystąpień typów jednostki POCO, platformy Entity Framework często tworzy wystąpienia typu pochodnego dynamicznie generowanym, który działa jako serwer proxy dla tej jednostki. Ten serwer proxy zastępuje niektóre właściwości wirtualnego jednostka do wstawienia punkty zaczepienia do operacji wykonywanych automatycznie, gdy uzyskano dostęp do właściwości. Na przykład ten mechanizm jest używany do obsługi powolne ładowanie relacji. Techniki przedstawione w tym temacie stosuje się jednakowo do modeli utworzonych za pomocą Code First i projektancie platformy EF.

Wyłączanie tworzenia serwera proxy

Czasami przydatne jest zapobiec tworzenia wystąpień serwera proxy programu Entity Framework. Na przykład serializacji wystąpień bez serwera proxy jest znacznie prostsze niż serializacji wystąpień serwera proxy. Tworzenie serwera proxy można wyłączyć, usuwając zaznaczenie flagi `ProxyCreationEnabled`. Jest jednym miejscu, można to zrobić w Konstruktorze typu kontekstu. Na przykład:

```
public class BloggingContext : DbContext
{
    public BloggingContext()
    {
        this.Configuration.ProxyCreationEnabled = false;
    }

    public DbSet<Blog> Blogs { get; set; }
    public DbSet<Post> Posts { get; set; }
}
```

Należy pamiętać, że EF nie utworzy serwerów proxy dla typów w przypadku, gdy nie ma nic do serwera proxy w celu. Oznacza to, czy można również uniknąć serwery proxy, przez umieszczenie typy, które są zapieczętowane i/lub mieć nie właściwości wirtualnego.

Jawne utworzenie wystąpienia serwera proxy

Nie będzie można utworzyć wystąpienia serwera proxy, jeśli tworzony jest wystąpienie jednostki za pomocą nowego operatora. To może nie być problemem, ale jeśli musisz utworzyć wystąpienie serwera proxy (na przykład, dzięki czemu będzie działać z opóźnieniem ładowania lub serwer proxy sprawdzaniu spójności) następnie możesz to zrobić za pomocą metody `Create DbSet`. Na przykład:

```
using (var context = new BloggingContext())
{
    var blog = context.Blogs.Create();
```

Jeśli chcesz utworzyć wystąpienie Typ pochodny jednostki można ogólnego wersję Utwórz. Na przykład:

```
using (var context = new BloggingContext())
{
    var admin = context.Users.Create<Administrator>();
```

Zauważ, że metody Create, nie dodać lub dołączyć utworzonej jednostki do kontekstu.

Należy pamiętać, że metody Create stworzy wystąpienia typu jednostki, sam typ serwera proxy dla jednostki podczas tworzenia będzie mieć żadnej wartości, ponieważ nie będzie robić niczego. Na przykład jeśli typ jednostki jest zapieczętowany i/lub nie ma wirtualnej właściwości, a następnie utwórz będzie po prostu utworzyć wystąpienia typu jednostki.

Pobieranie typu rzeczywistego obiektu z typem serwera proxy

Typy serwerów proxy mają nazwy, które wyglądają mniej więcej tak:

System.Data.Entity.DynamicProxies.Blog_5E43C6C196972BF0754973E48C9C941092D86818CD94005E9A759
B70BF6E48E6

Można znaleźć typu jednostki dla tego typu proxy przy użyciu metody GetObjectType od obiektu ObjectContext. Na przykład:

```
using (var context = new BloggingContext())
{
    var blog = context.Blogs.Find(1);
    var entityType = ObjectContext.GetObjectType(blog.GetType());
}
```

Należy zauważyć, że jeśli typ jest przekazywany do Element GetObjectType jest wystąpieniem typu jednostki, który nie jest typ serwera proxy następnie typu jednostki, nadal jest zwracana. Oznacza to, że zawsze można używać tej metody można pobrać typu rzeczywistego jednostki bez żadnych innych sprawdzania, aby sprawdzić, czy typ jest typ serwera proxy, czy nie.

Testowanie za pomocą pozorowania framework

06.10.2018 • 12 minutes to read • [Edit Online](#)

NOTE

EF6 poczawszy tylko — funkcje, interfejsów API itp. z opisem na tej stronie zostały wprowadzone w programie Entity Framework 6. Jeśli używasz starszej wersji, niektóre lub wszystkie informacje, nie ma zastosowania.

Podczas pisania testów dla aplikacji często jest pożądane, aby uniknąć osiągnięcia bazy danych. Entity Framework można to osiągnąć, tworząc kontekst — w przypadku zachowanie zdefiniowane przez testy — korzystającej z danych w pamięci.

Opcje tworzenia testu wartości podwójnej precyzji

Istnieją dwa różne podejścia, które mogą służyć do tworzenia w pamięci wersję kontekstu.

- **Tworzenie własnych testów, wartości podwójnej precyzji** — ta strategia polega na pisanie własnego kontekstu i DbSets implementacji w pamięci. Zapewnia wysoki poziom kontroli nad jak zachowują się klasy, ale może obejmować pisanie i będącej właścicielem rozsądny kodu.
- **Umożliwia tworzenie testów wartości podwójnej precyzji pozorowania framework** — za pomocą pozorowania framework (na przykład Moq) może mieć implementacji w pamięci, kontekstu i zestawy tworzone dynamicznie w czasie wykonywania.

W tym artykule zajmuje się za pomocą pozorowania framework. W celu tworzenia własnych podwaja testu zobacz [testowanie za pomocą Your Own testowanie wartości podwójnej precyzji](#).

Aby zademonstrować przy użyciu programu EF pozorowania Framework są użyjemy Moq. Najprostszym sposobem uzyskania Moq jest zainstalować [Moq pakietu NuGet](#).

Testowanie za pomocą wersji pre-EF6

Scenariusz przedstawione w tym artykule jest zależna od pewne zmiany, które wprowadziliśmy DbSet w EF6. Testowanie za pomocą EF5 i starszych wersji dla [testowanie za pomocą kontekstu fałszywe](#).

Ograniczenia dotyczące wartości podwójnej precyzji EF testu w pamięci

Test w pamięci wartości podwójnej precyzji, może być dobrym sposobem zapewnienia poziomu zasięg bitów aplikacji korzystających z programu EF testów jednostkowych. Jednak w ten sposób używasz LINQ to Objects do wykonywania zapytań dotyczących danych w pamięci. Może to spowodować, że inaczej niż tłumaczenie zapytań SQL, która jest uruchamiana względem bazy danych przy użyciu programu EF firmy dostawcy LINQ (LINQ to Entities).

Przykładem takiej różnicy Trwa ładowanie powiązanych danych. Jeśli tworzysz szereg blogi każdy z powiązanymi wpisy, a następnie korzystając z danych w pamięci dla każdego bloga zawsze zostaną załadowane pokrewnych wpisów. Jednak podczas uruchamiania w bazie danych dane tylko zostanie załadowany. Jeśli używana jest metoda Include.

Z tego powodu zaleca się zawsze zawierać pewien stopień end-to-end testy (oprócz testy jednostkowe) w celu zapewnienia działania usługi aplikacji poprawnie względem bazy danych.

Zgodnie z tego artykułu

Ten artykuł zawiera kompletny kod ofert, kopiowane do programu Visual Studio, aby z niego skorzystać, jeśli chcesz. Najłatwiej utworzyć **projektu testu jednostkowego** i należy do obiektu docelowego **.NET Framework 4.5** do wykonania w sekcjach, korzystających z async.

Modelu platformy EF

Usługa użyjemy do przetestowania, korzysta z programu EF modelu składa się z BloggingContext i klas blogu i Post. Ten kod został wygenerowany przez projektanta programu EF lub model Code First.

```
using System.Collections.Generic;
using System.Data.Entity;

namespace TestingDemo
{
    public class BloggingContext : DbContext
    {
        public virtual DbSet<Blog> Blogs { get; set; }
        public virtual DbSet<Post> Posts { get; set; }
    }

    public class Blog
    {
        public int BlogId { get; set; }
        public string Name { get; set; }
        public string Url { get; set; }

        public virtual List<Post> Posts { get; set; }
    }

    public class Post
    {
        public int PostId { get; set; }
        public string Title { get; set; }
        public string Content { get; set; }

        public int BlogId { get; set; }
        public virtual Blog Blog { get; set; }
    }
}
```

Właściwości DbSet wirtualnych za pomocą projektanta EF

Należy pamiętać o tym, czy właściwości DbSet w kontekście są oznaczone jako wirtualny. Umożliwi to pozorowanie platformę, by dziedziczyć nasz kontekst i zastępowanie te właściwości z implementacją pozorowane.

Jeśli używasz Code First, a następnie w Twoich zajęciach można edytować bezpośrednio. Jeśli używasz projektancie platformy EF następnie należy edytować szablon T4, który generuje kontekstu. Otwórz <nazwa_modelu>. Plik context.TT, który jest zagnieżdżony w przypadku pliku edmx, znajdź następujący fragment kodu i Dodaj virtual — słowo kluczowe, jak pokazano.

```
public string DbSet(EntitySet entitySet)
{
    return string.Format(
        CultureInfo.InvariantCulture,
        "{0} virtual DbSet\<{1}> {2} {{ get; set; }}",
        Accessibility.ForReadOnlyProperty(entitySet),
        _typeMapper.GetTypeName(entitySet.ElementType),
        _code.Escape(entitySet));
}
```

Usługi w celu zbadania

Aby zademonstrować, testowanie za pomocą testu w pamięci wartości podwójnej precyzyji zamierzamy zapisywać dla BlogService kilka testów. Usługa jest w stanie tworzenie nowych blogów (AddBlog) i zwraca wszystkie blogi uporządkowane według nazwy (GetAllBlogs). Oprócz GetAllBlogs udostępniliśmy również metodę, która asynchronicznie pobierze wszystkie blogi uporządkowane według nazwy (GetAllBlogsAsync).

```
using System.Collections.Generic;
using System.Data.Entity;
using System.Linq;
using System.Threading.Tasks;

namespace TestingDemo
{
    public class BlogService
    {
        private BloggingContext _context;

        public BlogService(BloggingContext context)
        {
            _context = context;
        }

        public Blog AddBlog(string name, string url)
        {
            var blog = _context.Blogs.Add(new Blog { Name = name, Url = url });
            _context.SaveChanges();

            return blog;
        }

        public List<Blog> GetAllBlogs()
        {
            var query = from b in _context.Blogs
                        orderby b.Name
                        select b;

            return query.ToList();
        }

        public async Task<List<Blog>> GetAllBlogsAsync()
        {
            var query = from b in _context.Blogs
                        orderby b.Name
                        select b;

            return await query.ToListAsync();
        }
    }
}
```

-Query scenariuszy testowania

To wszystko, co należy zrobić, aby rozpocząć testowanie metody niebędącą zapytaniem. Następującego testu używa Moq, aby utworzyć kontekst. Następnie tworzy DbSet<blogu> i tworzącej mają być zwracane przez właściwości blogi kontekstu. Dalej aby utworzyć nowy BlogService, który jest następnie używany do tworzenia nowego bloga — przy użyciu metody AddBlog używany jest kontekst. Ponadto ten test sprawdza, czy usługa dodano nowego bloga i o nazwie SaveChanges w kontekście.

```

using Microsoft.VisualStudio.TestTools.UnitTesting;
using Moq;
using System.Data.Entity;

namespace TestingDemo
{
    [TestClass]
    public class NonQueryTests
    {
        [TestMethod]
        public void CreateBlog_saves_a_blog_via_context()
        {
            var mockSet = new Mock<DbSet<Blog>>();

            var mockContext = new Mock<BloggingContext>();
            mockContext.Setup(m => m.Blogs).Returns(mockSet.Object);

            var service = new BlogService(mockContext.Object);
            service.AddBlog("ADO.NET Blog", "http://blogs.msdn.com/adonet");

            mockSet.Verify(m => m.Add(It.IsAny<Blog>()), Times.Once());
            mockContext.Verify(m => m.SaveChanges(), Times.Once());
        }
    }
}

```

Scenariusze testowania zapytań

Aby można było wykonać zapytania wzgółdem naszym teście DbSet double musimy skonfigurować implementację elementu IQueryble. Pierwszym krokiem jest utworzenie niektórych danych w pamięci — używamy listy<blogu>. Następnie utworzymy kontekst i DBSet<Blog> następnie Podłączanie implementację IQueryble DbSet — są one po prostu delegowanie do LINQ do dostawcy obiektów, która współdziała z listy<T>.

Firma Microsoft można utworzyć BlogService, oparte na typach Double nasze badania i upewnij się, że dane, które firma Microsoft wrócić z GetAllBlogs są uporządkowane według nazwy.

```

using Microsoft.VisualStudio.TestTools.UnitTesting;
using Moq;
using System.Collections.Generic;
using System.Data.Entity;
using System.Linq;

namespace TestingDemo
{
    [TestClass]
    public class QueryTests
    {
        [TestMethod]
        public void GetAllBlogs_orders_by_name()
        {
            var data = new List<Blog>
            {
                new Blog { Name = "BBB" },
                new Blog { Name = "ZZZ" },
                new Blog { Name = "AAA" },
            }.AsQueryable();

            var mockSet = new Mock<DbSet<Blog>>();
            mockSet.As<IQueryable<Blog>>().Setup(m => m.Provider).Returns(data.Provider);
            mockSet.As<IQueryable<Blog>>().Setup(m => m.Expression).Returns(data.Expression);
            mockSet.As<IQueryable<Blog>>().Setup(m => m.ElementType).Returns(data.ElementType);
            mockSet.As<IQueryable<Blog>>().Setup(m => m.GetEnumerator()).Returns(data.GetEnumerator());

            var mockContext = new Mock<BloggingContext>();
            mockContext.Setup(c => c.Blogs).Returns(mockSet.Object);

            var service = new BlogService(mockContext.Object);
            var blogs = service.GetAllBlogs();

            Assert.AreEqual(3, blogs.Count);
            Assert.AreEqual("AAA", blogs[0].Name);
            Assert.AreEqual("BBB", blogs[1].Name);
            Assert.AreEqual("ZZZ", blogs[2].Name);
        }
    }
}

```

Testowanie za pomocą zapytania asynchronousne

Entity Framework 6 wprowadziliśmy zestaw metod rozszerzenia, które może służyć do asynchronousnego wykonywanie zapytania. Przykładami tych metod ToListAsync FirstAsync, ForEachAsync itp.

Ponieważ Entity Framework zapytań korzystaj z LINQ, metody rozszerzające są definiowane na IQueryable i IEnumerable. Ponieważ są one przeznaczone tylko do użycia z programem Entity Framework możesz otrzymać następujący błąd przy próbie z nich korzystać na zapytanie LINQ, która nie jest zapytaniem Entity Framework:

Źródło IQueryable nie implementuje IDbAsyncEnumerable{0}. Tylko źródła, które implementują IDbAsyncEnumerable może służyć do operacji asynchronousnych Entity Framework. Aby uzyskać więcej informacji, zobacz <http://go.microsoft.com/fwlink/?LinkId=287068>.

Podczas gdy metody asynchronousne są obsługiwane tylko przy uruchamianiu kwerendę EF, można ich używać w testu jednostkowego, gdy uruchomiona przed w pamięci testowanie double DbSet.

Aby można było używać metod asynchronousnych, musimy utworzyć DbAsyncQueryProvider w pamięci do przetworzenia zapytania asynchronousne. O ile można skonfigurować dostawcę zapytania za pomocą Moq, jest znacznie łatwiejsze utworzyć implementacja podwójnego testowego w kodzie. Kod dla tej implementacji jest następującą:

```

using System.Collections.Generic;
using System.Data.Entity.Infrastructure;
using System.Linq;
using System.Linq.Expressions;
using System.Threading;
using System.Threading.Tasks;

namespace TestingDemo
{
    internal class TestDbAsyncQueryProvider<TEntity> : IDbAsyncQueryProvider
    {
        private readonly IQueryProvider _inner;

        internal TestDbAsyncQueryProvider(IQueryProvider inner)
        {
            _inner = inner;
        }

        public IQueryable CreateQuery(Expression expression)
        {
            return new TestDbAsyncEnumerable<TEntity>(expression);
        }

        public IQueryable<TElement> CreateQuery<TElement>(Expression expression)
        {
            return new TestDbAsyncEnumerable<TElement>(expression);
        }

        public object Execute(Expression expression)
        {
            return _inner.Execute(expression);
        }

        public TResult Execute<TResult>(Expression expression)
        {
            return _inner.Execute<TResult>(expression);
        }

        public Task<object> ExecuteAsync(Expression expression, CancellationToken cancellationToken)
        {
            return Task.FromResult(Execute(expression));
        }

        public Task<TResult> ExecuteAsync<TResult>(Expression expression, CancellationToken cancellationToken)
        {
            return Task.FromResult(Execute<TResult>(expression));
        }
    }

    internal class TestDbAsyncEnumerable<T> : EnumerableQuery<T>, IDbAsyncEnumerable<T>, IQueryable<T>
    {
        public TestDbAsyncEnumerable(IEnumerable<T> enumerable)
            : base(enumerable)
        { }

        public TestDbAsyncEnumerable(Expression expression)
            : base(expression)
        { }

        public IDbAsyncEnumerator<T> GetAsyncEnumerator()
        {
            return new TestDbAsyncEnumerator<T>(this.AsEnumerable().GetEnumerator());
        }

        IDbAsyncEnumerator IDbAsyncEnumerable.GetAsyncEnumerator()
        {
            return GetAsyncEnumerator();
        }
    }
}

```

```
IQueryProvider IQueryable.Provider
{
    get { return new TestDbAsyncQueryProvider<T>(this); }
}
}

internal class TestDbAsyncEnumerator<T> : IDbAsyncEnumerator<T>
{
    private readonly IEnumerator<T> _inner;

    public TestDbAsyncEnumerator(IEnumerator<T> inner)
    {
        _inner = inner;
    }

    public void Dispose()
    {
        _inner.Dispose();
    }

    public Task<bool> MoveNextAsync(CancellationToken cancellationToken)
    {
        return Task.FromResult(_inner.MoveNext());
    }

    public T Current
    {
        get { return _inner.Current; }
    }

    object IDbAsyncEnumerator.Current
    {
        get { return Current; }
    }
}
```

Teraz, gdy dostawca zapytania asynchroniczne możemy napisać test jednostkowy dla naszej nowej metody GetAllBlogsAsync.

```

using Microsoft.VisualStudio.TestTools.UnitTesting;
using Moq;
using System.Collections.Generic;
using System.Data.Entity;
using System.Data.Entity.Infrastructure;
using System.Linq;
using System.Threading.Tasks;

namespace TestingDemo
{
    [TestClass]
    public class AsyncQueryTests
    {
        [TestMethod]
        public async Task GetAllBlogsAsync_orders_by_name()
        {

            var data = new List<Blog>
            {
                new Blog { Name = "BBB" },
                new Blog { Name = "ZZZ" },
                new Blog { Name = "AAA" },
            }.AsQueryable();

            var mockSet = new Mock<DbSet<Blog>>();
            mockSet.As< IDbAsyncEnumerable<Blog>>()
                .Setup(m => m.GetAsyncEnumerator())
                .Returns(new TestDbAsyncEnumerator<Blog>(data.GetEnumerator()));

            mockSet.As< IQueryable<Blog>>()
                .Setup(m => m.Provider)
                .Returns(new TestDbAsyncQueryProvider<Blog>(data.Provider));

            mockSet.As< IQueryable<Blog>>().Setup(m => m.Expression).Returns(data.Expression);
            mockSet.As< IQueryable<Blog>>().Setup(m => m.ElementType).Returns(data.ElementType);
            mockSet.As< IQueryable<Blog>>().Setup(m => m.GetEnumerator()).Returns(data.GetEnumerator());

            var mockContext = new Mock<BloggingContext>();
            mockContext.Setup(c => c.Blogs).Returns(mockSet.Object);

            var service = new BlogService(mockContext.Object);
            var blogs = await service.GetAllBlogsAsync();

            Assert.AreEqual(3, blogs.Count);
            Assert.AreEqual("AAA", blogs[0].Name);
            Assert.AreEqual("BBB", blogs[1].Name);
            Assert.AreEqual("ZZZ", blogs[2].Name);
        }
    }
}

```

Testowanie za pomocą własnych testów, wartości podwójnej precyzji

13.09.2018 • 14 minutes to read • [Edit Online](#)

NOTE

EF6 począwszy tylko — funkcje, interfejsów API itp. z opisem na tej stronie zostały wprowadzone w programie Entity Framework 6. Jeśli używasz starszej wersji, niektóre lub wszystkie informacje, nie ma zastosowania.

Podczas pisania testów dla aplikacji często jest pożądane, aby uniknąć osiągnięcia bazy danych. Entity Framework można to osiągnąć, tworząc kontekst — w przypadku zachowanie zdefiniowane przez testy — korzystającej z danych w pamięci.

Opcje tworzenia testu wartości podwójnej precyzji

Istnieją dwa różne podejścia, które mogą służyć do tworzenia w pamięci wersję kontekstu.

- **Tworzenie własnych testów, wartości podwójnej precyzji** — ta strategia polega na pisanie własnego kontekstu i DbSets implementacji w pamięci. Zapewnia wysoki poziom kontroli nad jak zachowują się klasy, ale może obejmować pisanie i będącej właścicielem rozsądny kodu.
- **Umożliwia tworzenie testów wartości podwójnej precyzji pozorowania framework** — za pomocą pozorowania framework (na przykład Moq) może mieć implementacji w pamięci, kontekstu i zestawy tworzone dynamicznie w czasie wykonywania.

W tym artykule poradzi sobie z tworzeniem własnych testów double. Uzyskać informacje o używaniu pozorowania framework zobacz [testowanie za pomocą struktury pozorowanie](#).

Testowanie za pomocą wersji pre-EF6

Kod przedstawiony w tym artykule jest zgodny z platformy EF6. Testowanie za pomocą EF5 i starszych wersji dla [testowanie za pomocą kontekstu fałszywe](#).

Ograniczenia dotyczące wartości podwójnej precyzji EF testu w pamięci

Test w pamięci wartości podwójnej precyzji, może być dobrym sposobem zapewnienia poziomu zasięg bitów aplikacji korzystających z programu EF testów jednostkowych. Jednak w ten sposób używasz LINQ to Objects do wykonywania zapytań dotyczących danych w pamięci. Może to spowodować, że inaczej niż tłumaczenie zapytań SQL, która jest uruchamiana względem bazy danych przy użyciu programu EF firmy dostawcy LINQ (LINQ to Entities).

Przykładem takiej różnicy Trwa ładowanie powiązanych danych. Jeśli tworzysz szereg blogi każdy z powiązanymi wpisy, a następnie korzystając z danych w pamięci dla każdego bloga zawsze zostaną załadowane pokrewnych wpisów. Jednak podczas uruchamiania w bazie danych dane tylko zostanie załadowany. Jeśli używana jest metoda Include.

Z tego powodu zaleca się zawsze zawierać pewien stopień end-to-end testy (oprócz testy jednostkowe) w celu zapewnienia działania usługi aplikacji poprawnie względem bazy danych.

Zgodnie z tego artykułu

Ten artykuł zawiera kompletny kod ofert, kopiowane do programu Visual Studio, aby z niego skorzystać, jeśli chcesz. Najłatwiej utworzyć **projektu testu jednostkowego** i należy do obiektu docelowego **.NET Framework 4.5** do wykonania w sekcjach, korzystających z async.

Tworzenie interfejsu kontekstu

Zamierzamy Przyjrzyj się testowanie to usługa, która korzysta z programu EF modelu. Aby można było zastąpić nasz kontekst EF wersją w pamięci do testowania, zdefiniujemy interfejs zostaną implementować nasz kontekst EF (oraz jej w pamięci podwójnej precyzyji).

Usługi, którą użyjemy do przetestowania spowodują zapytania i modyfikowanie danych za pomocą właściwości DbSet nasz kontekst i również wywołać funkcję SaveChanges wypychania zmian do bazy danych. Możemy więc dołączania te elementy członkowskie w interfejsie.

```
using System.Data.Entity;

namespace TestingDemo
{
    public interface IBloggingContext
    {
        DbSet<Blog> Blogs { get; }
        DbSet<Post> Posts { get; }
        int SaveChanges();
    }
}
```

Modelu platformy EF

Usługa użyjemy do przetestowania, korzysta z programu EF modelu składa się z BloggingContext i klas blogu i Post. Ten kod został wygenerowany przez projektanta programu EF lub model Code First.

```

using System.Collections.Generic;
using System.Data.Entity;

namespace TestingDemo
{
    public class BloggingContext : DbContext, IBloggingContext
    {
        public DbSet<Blog> Blogs { get; set; }
        public DbSet<Post> Posts { get; set; }
    }

    public class Blog
    {
        public int BlogId { get; set; }
        public string Name { get; set; }
        public string Url { get; set; }

        public virtual List<Post> Posts { get; set; }
    }

    public class Post
    {
        public int PostId { get; set; }
        public string Title { get; set; }
        public string Content { get; set; }

        public int BlogId { get; set; }
        public virtual Blog Blog { get; set; }
    }
}

```

Implementowanie interfejsu kontekstu za pomocą projektanta EF

Należy pamiętać, że nasz kontekst implementuje interfejs IBloggingContext.

Jeśli używasz Code First można edytować kontekst bezpośrednio do implementacji interfejsu. Jeśli używasz projektancie platformy EF następnie należy edytować szablon T4, który generuje kontekstu. Otwórz <nazwa_modelu>. Plik context.TT, który jest zagnieżdżony w przypadku pliku edmx, znajdź następujący fragment kodu i Dodaj w interfejsie, jak pokazano.

```
<#=Accessibility.ForType(container)#> partial class <#=code.Escape(container)#> : DbContext, IBloggingContext
```

Usługi w celu zbadania

Aby zademonstrować, testowanie za pomocą testu w pamięci wartości podwójnej precyzji zamierzamy zapisywać dla BlogService kilka testów. Usługa jest w stanie tworzenie nowych blogów (AddBlog) i zwraca wszystkie blogi uporządkowane według nazwy (GetAllBlogs). Oprócz GetAllBlogs udostępniliśmy również metodę, która asynchronicznie pobierze wszystkie blogi uporządkowane według nazwy (GetAllBlogsAsync).

```

using System.Collections.Generic;
using System.Data.Entity;
using System.Linq;
using System.Threading.Tasks;

namespace TestingDemo
{
    public class BlogService
    {
        private IBloggingContext _context;

        public BlogService(IBloggingContext context)
        {
            _context = context;
        }

        public Blog AddBlog(string name, string url)
        {
            var blog = new Blog { Name = name, Url = url };
            _context.Blogs.Add(blog);
            _context.SaveChanges();

            return blog;
        }

        public List<Blog> GetAllBlogs()
        {
            var query = from b in _context.Blogs
                        orderby b.Name
                        select b;

            return query.ToList();
        }

        public async Task<List<Blog>> GetAllBlogsAsync()
        {
            var query = from b in _context.Blogs
                        orderby b.Name
                        select b;

            return await query.ToListAsync();
        }
    }
}

```

Podwaja się tworzenie testu w pamięci

Teraz gdy mamy rzeczywiste modelu platformy EF i usługa, która służy nadszedł czas na tworzenie testów w pamięci double, firma Microsoft można używać do testowania. Utworzyliśmy testu TestContext double na nasz kontekst. W wartości podwójnej precyzji testu, otrzymujemy wybierz zachowanie, chcemy, aby można było obsługiwać testy użyjemy do uruchomienia. W tym przykładzie po prostu wychwycimy liczbę przypadków, gdy jest wywoływana SaveChanges, ale może zawierać dowolną logikę wymaganą wymaganego do weryfikacji scenariuszy, które testujesz.

Utworzyliśmy również TestDbSet, zapewniająca DbSet implementację w pamięci. Udostępniamy pełny implementacja dla wszystkich metod na DbSet (z wyjątkiem znaleźć), ale musisz wdrożyć elementów członkowskich, które będą używać Twojego scenariusza testu.

TestDbSet sprawia, że użycie niektórych innych klas infrastruktury, które wprowadziliśmy, aby upewnić się, że zapytania asynchroniczne mogą być przetwarzane.

```

using System;
using System.Collections.Generic;

```

```
using System.Collections.ObjectModel;
using System.Data.Entity;
using System.Data.Entity.Infrastructure;
using System.Linq;
using System.Linq.Expressions;
using System.Threading;
using System.Threading.Tasks;

namespace TestingDemo
{
    public class TestContext : IBloggingContext
    {
        public TestContext()
        {
            this.Blogs = new TestDbSet<Blog>();
            this.Posts = new TestDbSet<Post>();
        }

        public DbSet<Blog> Blogs { get; set; }
        public DbSet<Post> Posts { get; set; }
        public int SaveChangesCount { get; private set; }
        public int SaveChanges()
        {
            this.SaveChangesCount++;
            return 1;
        }
    }

    public class TestDbSet<TEntity> : DbSet<TEntity>, IQueryble, IEnumerable<TEntity>,
 IDbAsyncEnumerable<TEntity>
    where TEntity : class
    {
        ObservableCollection<TEntity> _data;
        IQueryable _query;

        public TestDbSet()
        {
            _data = new ObservableCollection<TEntity>();
            _query = _data.AsQueryable();
        }

        public override TEntity Add(TEntity item)
        {
            _data.Add(item);
            return item;
        }

        public override TEntity Remove(TEntity item)
        {
            _data.Remove(item);
            return item;
        }

        public override TEntity Attach(TEntity item)
        {
            _data.Add(item);
            return item;
        }

        public override TEntity Create()
        {
            return Activator.CreateInstance<TEntity>();
        }

        public override TDerivedEntity Create<TDerivedEntity>()
        {
            return Activator.CreateInstance<TDerivedEntity>();
        }
    }
}
```

```

public override ObservableCollection< TEntity> Local
{
    get { return _data; }
}

Type IQueryable.ElementType
{
    get { return _query.ElementType; }
}

Expression IQueryable.Expression
{
    get { return _query.Expression; }
}

IQueryProvider IQueryable.Provider
{
    get { return new TestDbAsyncQueryProvider< TEntity>(_query.Provider); }
}

System.Collections.IEnumerator System.Collections.IEnumerable.GetEnumerator()
{
    return _data.GetEnumerator();
}

IEnumerator< TEntity> IEnumerable< TEntity>.GetEnumerator()
{
    return _data.GetEnumerator();
}

IDbAsyncEnumerator< TEntity> IDbAsyncEnumerable< TEntity>.GetAsyncEnumerator()
{
    return new TestDbAsyncEnumerator< TEntity>(_data.GetEnumerator());
}
}

internal class TestDbAsyncQueryProvider< TEntity> : IDbAsyncQueryProvider
{
    private readonly IQueryProvider _inner;

    internal TestDbAsyncQueryProvider(IQueryProvider inner)
    {
        _inner = inner;
    }

    public IQueryable CreateQuery(Expression expression)
    {
        return new TestDbAsyncEnumerable< TEntity>(expression);
    }

    public IQueryable< TElement> CreateQuery< TElement>(Expression expression)
    {
        return new TestDbAsyncEnumerable< TElement>(expression);
    }

    public object Execute(Expression expression)
    {
        return _inner.Execute(expression);
    }

    public TResult Execute< TResult>(Expression expression)
    {
        return _inner.Execute< TResult>(expression);
    }

    public Task< object> ExecuteAsync(Expression expression, CancellationToken cancellationToken)
    {
        return Task.FromResult(Execute(expression));
    }
}

```

```

        public Task<TResult> ExecuteAsync<TResult>(Expression expression, CancellationToken cancellationToken)
    {
        return Task.FromResult(Execute<TResult>(expression));
    }
}

internal class TestDbAsyncEnumerable<T> : EnumerableQuery<T>, IDbAsyncEnumerable<T>, IQueryable<T>
{
    public TestDbAsyncEnumerable(IEnumerable<T> enumerable)
        : base(enumerable)
    { }

    public TestDbAsyncEnumerable(Expression expression)
        : base(expression)
    { }

    public IDbAsyncEnumerator<T> GetAsyncEnumerator()
    {
        return new TestDbAsyncEnumerator<T>(this.AsEnumerable().GetEnumerator());
    }

    IDbAsyncEnumerator IDbAsyncEnumerable.GetAsyncEnumerator()
    {
        return GetAsyncEnumerator();
    }

    IQueryProvider IQueryable.Provider
    {
        get { return new TestDbAsyncQueryProvider<T>(this); }
    }
}

internal class TestDbAsyncEnumerator<T> : IDbAsyncEnumerator<T>
{
    private readonly IEnumerator<T> _inner;

    public TestDbAsyncEnumerator(IEnumerator<T> inner)
    {
        _inner = inner;
    }

    public void Dispose()
    {
        _inner.Dispose();
    }

    public Task<bool> MoveNextAsync(CancellationToken cancellationToken)
    {
        return Task.FromResult(_inner.MoveNext());
    }

    public T Current
    {
        get { return _inner.Current; }
    }

    object IDbAsyncEnumerator.Current
    {
        get { return Current; }
    }
}
}

```

Implementowanie wyszukiwania

Metoda wyszukiwania jest trudne do zaimplementowania w ogólny sposób. Jeśli musisz przetestować kod, który

sprawia, że użyj metody Find, który najłatwiej utworzyć test znalezć DbSet dla każdego z typów jednostek, które muszą być obsługiwane. Następnie można napisać logikę, aby znaleźć konkretny typ jednostki, jak pokazano poniżej.

```
using System.Linq;

namespace TestingDemo
{
    class TestBlogDbSet : TestDbSet<Blog>
    {
        public override Blog Find(params object[] keyValues)
        {
            var id = (int)keyValues.Single();
            return this.SingleOrDefault(b => b.BlogId == id);
        }
    }
}
```

Pisanie niektórych testów

To wszystko, co należy zrobić, aby uruchomić testy. Następujący test tworzy TestContext, a następnie usługi na podstawie tego kontekstu. Usługa jest następnie używany do utworzenia nowego bloga — przy użyciu metody AddBlog. Ponadto ten test sprawdza, czy usługa dodano nowego bloga kontekstu blogi właściwości i wywołuje SaveChanges w kontekście.

To przykładowe typy rzeczy, które można przetestować za pomocą podwójnego testu w pamięci i można dostosować logiki wartości podwójnej precyzji testów i weryfikacji zgodnie z wymaganiami.

```
using Microsoft.VisualStudio.TestTools.UnitTesting;
using System.Linq;

namespace TestingDemo
{
    [TestClass]
    public class NonQueryTests
    {
        [TestMethod]
        public void CreateBlog_saves_a_blog_via_context()
        {
            var context = new TestContext();

            var service = new BlogService(context);
            service.AddBlog("ADO.NET Blog", "http://blogs.msdn.com/adonet");

            Assert.AreEqual(1, context.Blogs.Count());
            Assert.AreEqual("ADO.NET Blog", context.Blogs.Single().Name);
            Assert.AreEqual("http://blogs.msdn.com/adonet", context.Blogs.Single().Url);
            Assert.AreEqual(1, context.SaveChangesCount());
        }
    }
}
```

Oto inny przykład testu — tym razem taki, który wykonuje kwerendę. Uruchamia test, tworząc kontekstu testu z danymi w jego blogu właściwość — należy pamiętać, że dane nie są w kolejności alfabetycznej. Firma Microsoft można utworzyć BlogService, na podstawie naszych kontekstu testu i upewnić się, że dane, które firma Microsoft wrócić z GetAllBlogs są uporządkowane według nazwy.

```

using Microsoft.VisualStudio.TestTools.UnitTesting;

namespace TestingDemo
{
    [TestClass]
    public class QueryTests
    {
        [TestMethod]
        public void GetAllBlogs_orders_by_name()
        {
            var context = new TestContext();
            context.Blogs.Add(new Blog { Name = "BBB" });
            context.Blogs.Add(new Blog { Name = "ZZZ" });
            context.Blogs.Add(new Blog { Name = "AAA" });

            var service = new BlogService(context);
            var blogs = service.GetAllBlogs();

            Assert.AreEqual(3, blogs.Count);
            Assert.AreEqual("AAA", blogs[0].Name);
            Assert.AreEqual("BBB", blogs[1].Name);
            Assert.AreEqual("ZZZ", blogs[2].Name);
        }
    }
}

```

Ponadto będziemy pisać jeden więcej test, który korzysta z naszego metody asynchronicznej, upewnij się, że infrastruktura `async` jest dostępna w `TestDbSet` działa.

```

using Microsoft.VisualStudio.TestTools.UnitTesting;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace TestingDemo
{
    [TestClass]
    public class AsyncQueryTests
    {
        [TestMethod]
        public async Task GetAllBlogsAsync_orders_by_name()
        {
            var context = new TestContext();
            context.Blogs.Add(new Blog { Name = "BBB" });
            context.Blogs.Add(new Blog { Name = "ZZZ" });
            context.Blogs.Add(new Blog { Name = "AAA" });

            var service = new BlogService(context);
            var blogs = await service.GetAllBlogsAsync();

            Assert.AreEqual(3, blogs.Count);
            Assert.AreEqual("AAA", blogs[0].Name);
            Assert.AreEqual("BBB", blogs[1].Name);
            Assert.AreEqual("ZZZ", blogs[2].Name);
        }
    }
}

```

Testowalność i Entity Framework 4.0

27.09.2018 • 76 minutes to read • [Edit Online](#)

Scott Allen

Data publikacji: Maja 2010

Wprowadzenie

Ten dokument zawiera opis i pokazuje, jak pisać kodu sprawdzalnego działa zgodnie z ADO.NET Entity Framework 4.0 i Visual Studio 2010. W tym dokumencie nie podejmuje próby skupić się na określonych metodologii testowania, takich jak projektowanie oparte na testach (TDD) lub projektowania opartego na zachowanie (BDD). Zamiast tego ten dokument koncentruje się na temat sposobu pisania kodu, który używa programu ADO.NET Entity Framework, ale pozostaje ułatwia odizolować i przetestować w zautomatyzowany sposób. Zapoznamy się często używane wzorce projektowe, które ułatwiają dostęp do scenariuszy testowania w danych i zobacz, jak te wzorce są stosowane, gdy przy użyciu platformy. Również przyjrzymy określonych funkcji framework, aby zobaczyć, jak te funkcje mogą współdziałać w kodu sprawdzalnego działa zgodnie.

Co to jest kodu sprawdzalnego działa zgodnie?

Możliwość sprawdzenia oprogramowanie przy użyciu zautomatyzowanych testów jednostek oferuje wiele korzyści pożądane. Wszyscy wie, że dobry test zmniejsza liczbę usterek oprogramowania w aplikacji i zwiększa jakość aplikacji — ale przebiega testów jednostkowych w miejscu wykraczają daleko po prostu znajdowaniu usterek.

Zestaw testów jednostkowych dobre umożliwia zespół deweloperów zaoszczędzić czas i zachowuje kontrolę nad tworzonym przez nich oprogramowaniem. Zespół wprowadzić zmiany w istniejącym kodzie, Refaktoryzacja, ich przeprojektowywania i restrukturyzacji oprogramowania do nowych wymagań i dodawania nowych składników do aplikacji, korzystając z możliwości, wiedząc, zestawu testów, można sprawdzić działanie aplikacji. Testy jednostkowe są częścią cyklu szybkiej opinii w ułatwienia zmian i zachować łatwość oprogramowania w miarę wzrostu złożoności.

Testy jednostkowe jest dostarczany z ceną, jednak. Zespół będzie musiał poświęcić czas na tworzenie i obsługę testów jednostkowych. Nakład pracy wymagany do utworzenia tych testów jest bezpośrednio związana **testowalności** podstawowego oprogramowania. Jak łatwo jest oprogramowanie do testowania? Zespół projektowania oprogramowania za pomocą testowania należy pamiętać, utworzy szybciej niż zespół, Praca z oprogramowaniem bez sprawdzalnego działa zgodnie skuteczne testy.

Firma Microsoft zaprojektowała ADO.NET Entity Framework 4.0 (EF4) za pomocą testowania na uwadze. Nie oznacza to, że deweloperzy będzie zapisywać testów jednostkowych dla framework sam kod. Zamiast tego cele testowania EF4 ułatwiają tworzenie kodu sprawdzalnego działa zgodnie opartą na strukturze. Zanim przyjrzymy się konkretnie przykłady, warto zrozumieć jakość kodu sprawdzalnego działa zgodnie.

Jakość kodu sprawdzalnego działa zgodnie

Kod, który jest łatwy do testów zawsze będzie mieć co najmniej dwóch cech. Łatwo jest pierwszym kodu sprawdzalnego działa zgodnie **obserwować**. Biorąc pod uwagę pewne zestaw danych wejściowych, powinno być łatwe do sprawdzanie danych wyjściowych kodu. Na przykład następujące metody badania jest łatwe, ponieważ metoda bezpośrednio zwraca wynik obliczeń.

```
public int Add(int x, int y) {
    return x + y;
}
```

Testowanie metody jest trudne, jeśli metoda zapisuje obliczona wartość do gniazda sieciowego, tabeli bazy danych lub plików, podobnie do poniższego kodu. Test musi być wykonania dodatkowej pracy do pobrania wartości.

```
public void AddAndSaveToFile(int x, int y) {
    var results = string.Format("The answer is {0}", x + y);
    File.WriteAllText("results.txt", results);
}
```

Po drugie, jest łatwa do kodu sprawdzalnego działa zgodnie **izolowania**. Użyjmy następujący pseudo-kod złe przykład kodu sprawdzalnego działa zgodnie.

```
public int ComputePolicyValue(InsurancePolicy policy) {
    using (var connection = new SqlConnection("dbConnection"))
    using (var command = new SqlCommand(query, connection)) {

        // business calculations omitted ...

        if (totalValue > notificationThreshold) {
            var message = new MailMessage();
            message.Subject = "Warning!";
            var client = new SmtpClient();
            client.Send(message);
        }
    }
    return totalValue;
}
```

Metoda ta jest łatwy do obserwowania — firma Microsoft może przekazać ubezpieczeniowej i sprawdź, czy oczekiwany wynik pasuje do wartości zwracanej. Jednak do metody testowej musimy mieć zainstalowane przy użyciu poprawnego schematu bazy danych i konfigurowanie serwera SMTP, w przypadku, gdy metoda próbuje wysłać wiadomość e-mail.

Test jednostkowy tylko chciał sprawdzić, czy logiki obliczeń wewnętrz metody, ale test może zakończyć się niepowodzeniem, ponieważ serwer poczty e-mail jest w trybie offline lub przenieść serwer bazy danych. Oba te błędy są związane z zachowaniem testu będzie chciał sprawdzić, czy. Zachowanie jest trudne do izolowania.

Programistów, którzy Dokładamy wszelkich starań, aby zapisać często kodu sprawdzalnego działa zgodnie Dokładamy wszelkich starań zachować separacji w kodzie są zapisu. Powyżej metody należy skoncentrować się na obliczeń biznesowych i delegować szczegółów implementacji bazy danych i poczty e-mail do innych składników. Robert C. Martin wywołuje to pojedynczy zasady odpowiedzialności. Obiekt powinna hermetyzować pojedynczy, wąskie odpowiedzialność, takich jak obliczenia wartości zasady. Wszystkie inne zadania bazy danych i powiadomień należy odpowiedzialność innego obiektu. Kod napisany w ten sposób jest łatwiejsze do izolowania ponieważ koncentruje się na pojedynczym zadaniu.

Na platformie .NET mamy abstrakcji, potrzebne do działania w przekonaniu pojedynczej odpowiedzialności i uzyskania izolacji. Możemy użyć definicji interfejsu i wymusić kod, aby użyć abstrakcji interfejsu, a nie do konkretnego typu. W dalszej części tego dokumentu zobaczymy, jak metoda, takich jak zły przykład przedstawiony powyżej może współpracować z interfejsów *Szukaj* jak rozmawiamy w bazie danych. W czasie testu jednak możemy użyć zamiast fikcyjnego implementację, która nie skontaktuj się z bazą danych, ale zamiast tego przechowuje dane w pamięci. Ta implementacja fikcyjnego izoluje kodu z niepowiązanych problemów w kod dostępu do danych lub baza danych konfiguracji.

Istnieją dodatkowe korzyści z izolacji. Obliczenia biznesowego w ostatnim metody powinna trwać tylko kilka

milisekund, które można wykonać, ale same testy mogą działać przez kilka sekund jako przeskoków kodu wokół sieci i rozmowy na różnych serwerach. Testy jednostkowe, należy uruchomić szybki w celu ułatwienia niewielkich zmian. Testy jednostkowe powinny także powtarzalne oraz wystąpi niepowodzenie, ponieważ składnik niezwiązanych ze sobą na test ma problem. Pisanie kodu, który jest łatwy do obserwowania i do izolowania oznacza, że deweloperzy będzie teraz łatwiejsze pisania testów dla kodu, mogą spędzać mniej czasu oczekiwania na testami do wykonania, a więcej co ważniejsze, mogą spędzać mniej czasu, śledzenie błędów, które nie istnieją.

Miejmy nadzieję można docenić zalety testowania i zrozumieć cechy, które wykazuje kod sprawdzalnego działa zgodnie. Jesteśmy rozwiązać jak napisać kod, który współdziała z EF4, aby zapisać dane w bazie danych pozostając dostrzegalnych i łatwe izolowanie, ale najpierw będzie zawęzić naszym głównym celem w celu omówienia sprawdzalnego działa zgodnie projektów, aby uzyskać dostęp do danych.

Wzorce projektowe zapewniające trwałość danych

Obu przykładach zły przedstawiony wcześniej było zbyt wiele obowiązki. Pierwszy przykład zły musiał wykonać obliczenia i zapisu do pliku. Drugi przykład zły musiały odczytywać dane z bazy danych i wykonywanie obliczeń biznesowych i wysyłanie wiadomości e-mail. Projektowanie mniejszych metody, które Rozdzieli problemy i delegować odpowiedzialność za inne składniki wprowadzisz dotykowego do pisania kodu sprawdzalnego działa zgodnie. Celem jest tworzyć funkcje za pośrednictwem akcji abstrakcje niewielkiego i skupionego projektu.

Gdy chodzi o trwałości danych małe abstrakcje ukierunkowanych, których firma Microsoft szuka są więc wspólne zostało zostały opisane jako wzorców projektowych. Książki Martina Fowlera wzorców Enterprise architektury aplikacji była pierwszym pracy do opisania tych wzorców w drukowania. Firma Microsoft udostępnili krótki opis tych wzorców w poniższych sekcjach, zanim pokazujemy, jak te ADO.NET Entity Framework implementuje i współdziała z tych wzorców.

Wzorzec repozytorium

Fowlera wynika z repozytorium "pośredniczy między domeną i dane warstw mapowania za pomocą interfejsu przypominającego kolekcji do uzyskiwania dostępu do obiektów domeny". Celem wzorca repozytorium jest do izolowania kodu z minutiae dostępu do danych i jak widzieliśmy wcześniej izolacji jest cechę wymaganych do testowania.

Kluczem do izolacji jest, jak repozytorium przedstawia obiektów za pomocą interfejsu przypominającego kolekcji. Logika zapisu do użycia w repozytorium ma nie wiadomo, jak repozytorium zmateriaлизowania obiektów, których w przypadku żądania. Repozytorium może komunikować się z bazą danych lub po prostu może zwracać obiekty z kolekcji w pamięci. Wszystko, czego Twój kod musi wiedzieć, jest repozytorium pojawi się do zachowania kolekcji i pobierania, dodawanie i usuwanie obiektów z kolekcji.

W istniejących aplikacjach .NET konkretnych repozytorium często dziedziczy interfejs ogólny, jak pokazano poniżej:

```
public interface IRepository<T> {
    IEnumerable<T> FindAll();
    IEnumerable<T> FindBy(Expression<Func<T, bool>> predicate);
    T FindById(int id);
    void Add(T newEntity);
    void Remove(T entity);
}
```

Wybieramy kilka zmian do definicji interfejsu gdy firma Microsoft zapewnia implementację dla EF4, ale podstawowe pojęcia pozostają bez zmian. Kod można użyć konkretnego repozytorium implementacji interfejsu można pobrać jednostki, jego wartość klucza podstawowego, można pobrać kolekcję jednostek na podstawie oceny predykat, lub po prostu pobrać wszystkie dostępne jednostki. Kod można również dodawać i usuwać jednostki za pośrednictwem interfejsu repozytorium.

Biorąc pod uwagę obiektów IRepository pracowników, kod może wykonać następujące czynności.

```

var employeesNamedScott =
    repository
        .FindBy(e => e.Name == "Scott")
        .OrderBy(e => e.HireDate);
var firstEmployee = repository.FindById(1);
var newEmployee = new Employee() { /*... */};
repository.Add(newEmployee);

```

Ponieważ kod wykorzystuje interfejs (IRepository pracowników), firma Microsoft może dostarczyć kod różne implementacje interfejsu. Jedna implementacja może być implementacją wspierane przez EF4 i trwałość obiektów w bazie danych programu Microsoft SQL Server. Inną implementację (po jednym używanych przez firmę Microsoft podczas testowania) może być objęta obiektów listy pracowników w pamięci. Interfejs może pomóc w celu uzyskania izolacji w kodzie.

Zwróć uwagę, IRepository<T> interfejsu nie ujawnia operacji zapisu. Jak zaktualizować istniejące obiekty? Mogą pochodzić między definicje IRepository, które obejmują operację zapisu i implementacji tych repozytoriów, należy od razu utrwały obiektu do bazy danych. Jednak w wielu aplikacjach nie chcemy zachować obiekty indywidualnie. Zamiast tego chcemy Ożyw obiektów, by móc z różnych repozytoriów, zmodyfikować te obiekty jako część operacji biznesowych i następnie zachować wszystkie obiekty w ramach jednej operacji niepodzielnych. Na szczęście jest wzorzec tego typu zachowania.

Jednostka wzorzec pracy

Fowlera mówi, jednostka pracy będzie "Obsługa listy obiektów wpływ transakcji biznesowych i służy do koordynowania zapisu poza zmiany i rozwiązywanie problemów współbieżności". Jest odpowiedzialny za jednostkę pracy do śledzenia zmian do obiektów, firma Microsoft Ożyw z repozytorium i utrwała wszelkie zmiany, które wprowadziliśmy do obiektów, gdy o jednostce pracy, aby potwierdzić zmiany. Jest również odpowiedzialność za jednostkę pracy do wykonania nowe obiekty, firma Microsoft zostały dodane do wszystkich repozytoriów i wstawianie obiektów bazy danych, a także usunięcie Zarządzaj witryną.

Jeśli nigdy nie wykonano żadnej pracy z zestawami danych ADO.NET następnie będzie już można zapoznać się z jednostką wzorzec pracy. Zestawy danych ADO.NET wcześniej mieli możliwość Śledź nasze aktualizacje, usuwanie i wstawianie elementu DataRow obiektów i może (za pomocą adaptera TableAdapter) uzgadniają wszystkich zmian do bazy danych. Jednak obiektów DataSet modelu podzespołu odłączonej bazy danych. Jednostka wzorzec pracy wykazuje takie samo zachowanie, ale działa z obiektami biznesowymi i obiektów domeny, które są odizolowane od kod dostępu do danych i rozpoznaje bazy danych.

Abstrakcja do modelowania jednostek pracy w kodzie .NET może wyglądać następująco:

```

public interface IUnitOfWork {
    IRepository<Employee> Employees { get; }
    IRepository<Order> Orders { get; }
    IRepository<Customer> Customers { get; }
    void Commit();
}

```

Dzięki uwidocznieniu działania odwołania repozytorium przy użyciu jednostki pracy, którą firma Microsoft zapewnia pojedynczą jednostkę pracy obiekt ma możliwość śledzenia wszystkich jednostek zmaterializowanego podczas transakcji biznesowych. Implementacja metody zatwierdzania dla rzeczywistego jednostki pracy jest do uzgadniają zmiany w pamięci z bazą danych, gdzie się Magia.

Biorąc pod uwagę odwołaniem IUnitOfWork, kod można wprowadzać zmiany w obiektach firm pobierane z jednego lub więcej repozytoriów i Zapisz wszystkie zmiany, przy użyciu niepodzielnych operacji zatwierdzania.

```
var firstEmployee = unitofWork.Employees.FindById(1);
var firstCustomer = unitofWork.Customers.FindById(1);
firstEmployee.Name = "Alex";
firstCustomer.Name = "Christopher";
unitofWork.Commit();
```

Wzorzec obciążenia z opóźnieniem

Fowlera używa nazwy obciążenia z opóźnieniem do opisania "obiekt, który nie zawiera wszystkich danych potrzebujesz, ale wie, jak przygotować". Przezroczysty powolne ładowanie to ważna cecha ma podczas pisania kodu sprawdzalnego działa zgodnie biznesowych i Praca z relacyjnej bazy danych. Na przykład rozważmy poniższy kod.

```
var employee = repository.FindById(id);
// ... and later ...
foreach(var timeCard in employee.TimeCards) {
    // .. manipulate the timeCard
}
```

Jak jest wypełniana kolekcji kart Istnieją dwa możliwe odpowiedzi. Jedną odpowiedź jest, że repozytorium pracowników, po wyświetleniu monitu o pobranie pracownika, wysyła zapytanie do pobrania pracownika wraz z informacjami skojarzone karta godzin pracownika. Relacyjne bazy danych to zwykle wymaga zapytanie z klauzulą JOIN i może skutkować podczas pobierania informacji niż aplikacja potrzebuje. Co zrobić, jeśli aplikacja nigdy nie musi dotykać właściwości kart?

Druga odpowiedź jest możliwa załadować właściwości kart "na żądanie". To powolne ładowanie jest niewidoczne dla logiki biznesowej i niejawne, ponieważ kod nie jest wywoływany specjalne interfejsów API można pobrać informacji o karcie czasu. Kod zakłada, że informacje dotyczące karty czasu jest obecna, gdy są potrzebne. Ładowanie z opóźnieniem, które zazwyczaj polega na przechwytywaniu środowiska uruchomieniowego wywołań metody opisywanego zaangażowanych jest kilka magic. Przechwytyujący kodu jest odpowiedzialny za komunikować się z bazą danych i pobierania informacji o karcie czasu przy równoczesnym zachowaniu logika biznesowa może być logiki biznesowej. Ta magic obciążenia z opóźnieniem umożliwia kod firm, aby izolować sam z operacjami pobierania danych i skutkuje więcej kodu sprawdzalnego działa zgodnie.

Wadą ładowane z opóźnieniem jest fakt, że aplikacja *jest* potrzebne informacje karty godz., kod będzie wykonywał żadnych dodatkowych kwerend. To nie jest istotna dla wielu aplikacji, ale dla aplikacji zawierających poufne dane wydajności lub aplikacji liczba obiektów pracowników w pętli i wykonywanie zapytania do pobierania czasu kart podczas każdej iteracji pętli (problem często określany jako N + 1 problem z zapytania), powolne ładowanie jest przeciągania. W tych scenariuszach aplikacji może być eagerly załadować informacji o czasie karty w najbardziej efektywny sposób.

Na szczęście zobaczymy, jak EF4 obsługuje zarówno niejawne obciążen z opóźnieniem i wydajne eager ładuje możemy przenosić do następnej sekcji i implementacji tych wzorców.

Implementowanie wzorców z programu Entity Framework

Dobra wiadomość jest wszystkich wzorców projektowych, które firma Microsoft opisanego w ostatniej sekcji są proste do wdrożenia przy użyciu EF4. Aby zademonstrować są użyjemy prostej aplikacji ASP.NET MVC do edytowania i wyświetlania pracowników i ich skojarzone karta godzin. Rozpoczniemy pracę przy użyciu następujących "zwykłe stare CLR obiektów" (POCOs).

```

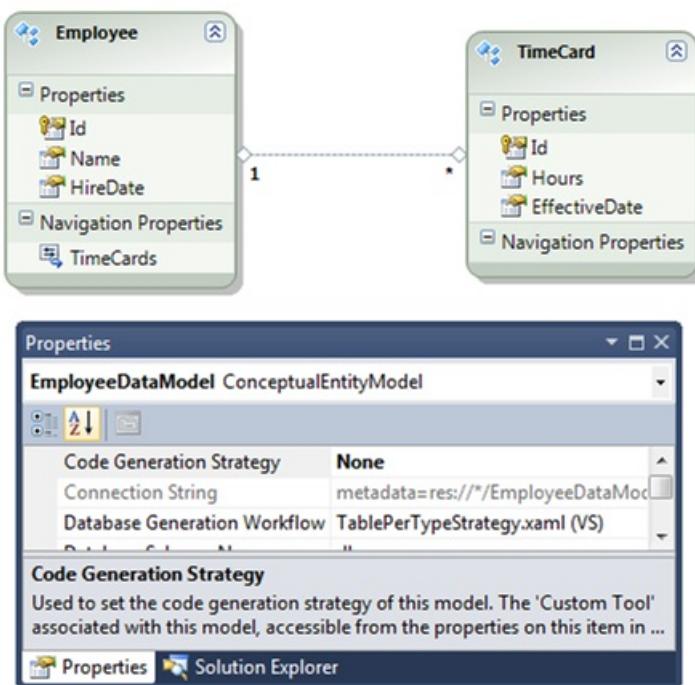
public class Employee {
    public int Id { get; set; }
    public string Name { get; set; }
    public DateTime HireDate { get; set; }
    public ICollection<TimeCard> TimeCards { get; set; }
}

public class TimeCard {
    public int Id { get; set; }
    public int Hours { get; set; }
    public DateTime EffectiveDate { get; set; }
}

```

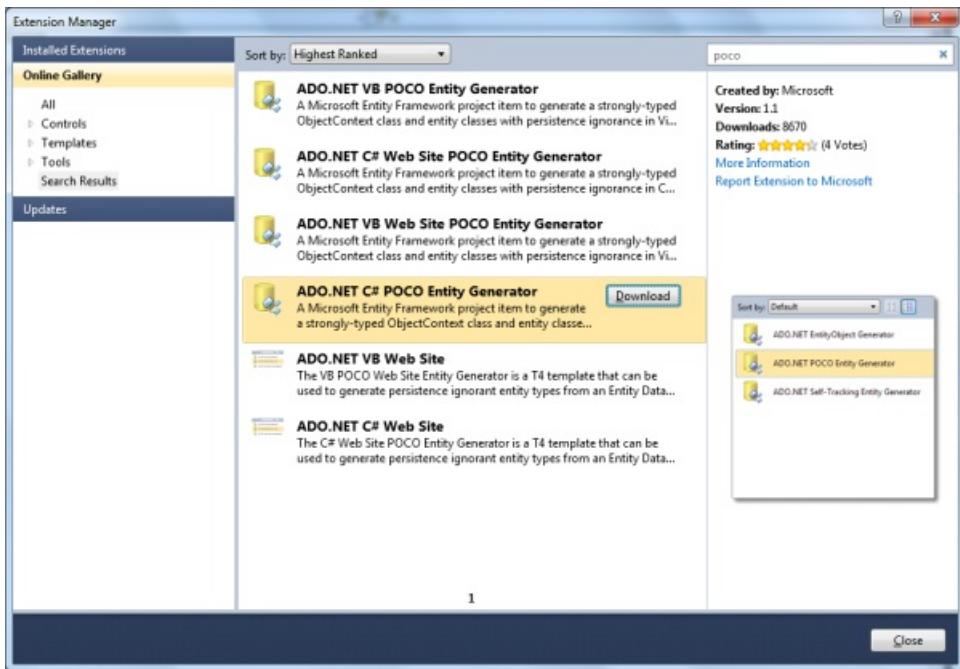
Te definicje klas będzie się nieznacznie zmieniać omówimy różne podejścia i funkcje EF4, ale celem jest zapewnienie tych klas jako trwałości zakresu (PI) jak to możliwe. Obiekt PI nie wie, jak, lub nawet *Jeśli*, stan przechowuje, są przechowywane w bazie danych. PI i POCOs idą ręka w rękę z oprogramowaniem sprawdzalnego działa zgodnie. Obiekty przy użyciu podejścia POCO są mniej ograniczone, bardziej elastyczne i łatwiejsze do testu, ponieważ mogą one działać bez bazy danych istnieje.

Za pomocą POCOs w miejscu możemy utworzyć Entity Data Model (EDM) w programie Visual Studio (patrz rysunek 1). Firma Microsoft nie będzie używać EDM do generowania kodu dla naszych jednostek. Zamiast tego chcemy użyć jednostki, które firma Microsoft lovingly pracowali ręcznie. Firma Microsoft będzie używała tylko EDM Generowanie naszych schemat bazy danych, a następnie podaj metadane EF4 potrzebuje do mapowania obiektów do bazy danych.



Rysunek 1.

Uwaga: Jeśli chcesz najpierw opracowanie modelu EDM jest możliwe czyszczenie, generowanie kodu POCO z EDM. Można to zrobić za pomocą rozszerzenia programu Visual Studio 2010, dostarczane przez zespół Data programowania. Aby pobrać rozszerzenie, uruchom Menedżera rozszerzeń z menu Narzędzia w programie Visual Studio i znajdowania galerii szablonów w trybie online "POCO" (zobacz rysunek 2). Istnieje kilka szablonów POCO są dostępne na platformie EF. Aby uzyskać więcej informacji na temat korzystania z tego szablonu, zobacz "[Instruktaż: obiektów POCO szablonu programu Entity Framework](#)".



Rysunek 2

Z tego POCO punkt początkowy przeanalizujemy dwa różne podejścia do kodu sprawdzalnego działa zgodnie. Pierwszym sposobem czy mogę wywołać podejście EF, ponieważ wykorzystuje abstrakcje z Entity Framework interfejs API, aby zaimplementować jednostkę pracy i repozytoriów. W drugiej metody możemy utworzyć własną abstrakcję niestandardowe repozytorium i wtedy wyświetlane zalety i wady każdej metody. Rozpoczniemy od wypróbowania podejście EF.

Implementacja przetwarzających EF

Należy wziąć pod uwagę następujące akcji kontrolera w projekcie ASP.NET MVC. Akcja pobiera obiekt pracowników i zwraca wynik, aby wyświetlić widok szczegółowy pracownika.

```
public ViewResult Details(int id) {
    var employee = _unitOfWork.Employees
        .Single(e => e.Id == id);
    return View(employee);
}
```

Jest kodu sprawdzalnego działa zgodnie? Istnieją co najmniej dwóch testów czy musimy zweryfikować zachowanie akcji. Najpierw chcemy sprawdzić, czy akcja zwraca widok poprawne — łatwe testu. Firma Microsoft będzie również chcieć napisać test, aby sprawdzić działanie powoduje pobranie poprawne pracowników i prosimy o poświadczenie zrobienie tego bez wykonywania kodu, aby w bazie danych. Należy pamiętać, że chcemy izolowanie testowanego kodu. Izolacja będzie upewnił się, że test nie zakończy się niepowodzeniem z powodu usterki w kod dostępu do danych lub baza danych konfiguracji. Jeśli test zakończy się niepowodzeniem, wiemy, że mamy usterkę w logiką kontrolera, a nie w niektórych niższe składnika poziomu systemu.

W celu uzyskania izolacji poprosimy niektóre elementy abstrakcji takich jak interfejsy, które firma Microsoft przedstawiony wcześniej dla repozytoriów i jednostki pracy. Pamiętaj, że wzorzec repozytorium jest przeznaczona do pośredniczy między obiektów domeny i warstwy mapowanie danych. W tym scenariuszu EF4 jest dane mapowania warstwy i udostępnia już abstrakcji podobne do repozytorium, o nazwie `IObjectSet<T>` (z przestrzeni nazw `System.Data.Objects`). Definicja interfejsu wygląda podobnie do poniższego.

```

public interface IObjectSet<TEntity> :
    IQueryable<TEntity>,
    IEnumerable<TEntity>,
    IQueryable,
    IEnumerable
    where TEntity : class
{
    void AddObject(TEntity entity);
    void Attach(TEntity entity);
    void DeleteObject(TEntity entity);
    void Detach(TEntity entity);
}

```

IObjectSet<T> spełnia wymagania dotyczące repozytorium, ponieważ jest on podobny kolekcji obiektów (za pośrednictwem interfejsu IEnumerable<T>) i dostarcza metod dodawania i usuwania obiektów z kolekcji symulowane. Metod dołączania i odłączania uwidocznić dodatkowe funkcje interfejsu API EF4. Aby użyć IObjectSet<T> jako interfejs dla repozytoriów potrzebujemy jednostkę pracy abstrakcji można powiązać ze sobą repozytoriów.

```

public interface IUnitOfWork {
    IObjectSet<Employee> Employees { get; }
    IObjectSet<TimeCard> TimeCards { get; }
    void Commit();
}

```

Jedną konkretną implementację tego interfejsu będzie się komunikował z programu SQL Server i łatwo utworzyć za pomocą klasy obiektu ObjectContext z EF4. Klasa obiektu ObjectContext jest prawdziwe jednostka pracy w interfejsie API EF4.

```

public class SqlUnitOfWork : IUnitOfWork {
    public SqlUnitOfWork() {
        var connectionString =
            ConfigurationManager
                .ConnectionStrings[ConnectionStringName]
                .ConnectionString;
        _context = new ObjectContext(connectionString);
    }

    public IObjectSet<Employee> Employees {
        get { return _context.CreateObjectSet<Employee>(); }
    }

    public IObjectSet<TimeCard> TimeCards {
        get { return _context.CreateObjectSet<TimeCard>(); }
    }

    public void Commit() {
        _context.SaveChanges();
    }

    readonly ObjectContext _context;
    const string ConnectionStringName = "EmployeeDataModelContainer";
}

```

Dzięki temu IObjectSet<T> w życie jest równie proste jak wywołania metody CreateObjectSet obiektu ObjectContext. Za kulisami środowisko używa metadanych zamieszczone w EDM, aby wygenerować konkretnego obiektu ObjectSet<T>. Używany będzie ze zwracaniem IObjectSet<T> interfejsu, ponieważ ułatwi to zachować testowania w kodzie klienta.

Ta implementacja konkretnych przydaje się w środowisku produkcyjnym, ale musimy skupić się na sposób użyjemy

naszego abstrakcji IUnitOfWork aby ułatwić testowanie.

Badanie wartości podwójnej precyzji

Aby wyizolować akcji kontrolera będziemy potrzebować możliwości przełączania się między rzeczywistych jednostkę pracy (wspariana przez obiekt ObjectContext) i testowej jednostki typu double lub "fałszywych" pracy (wykonywania operacji w pamięci). Typowym podejściem do wykonania tego rodzaju przełączania jest zezwala kontroler MVC wystąpienia jednostki pracy, ale zamiast tego przebiegu jednostkę pracy do kontrolera jako parametr konstruktora.

```
class EmployeeController : Controller {
    public EmployeeController(IUnitOfWork unitOfWork) {
        _unitOfWork = unitOfWork;
    }
    ...
}
```

Powyższy kod jest przykładem iniekcji zależności. Nie umożliwiamy kontrolera utworzyć jego zależności (jednostka pracy), ale wstrzykiwanie zależności do kontrolera. W projekcie MVC jest często używa się fabryki kontrolerów niestandardowych w połączeniu z odwrócenie kontenera kontrolek (IoC) do automatyzowania iniekcji zależności. Te tematy wykraczają poza zakres tego artykułu, ale możesz przeczytać więcej przez następujące odwołania na końcu tego artykułu.

Jednostka fałszywych wprowadzania pracy, które firma Microsoft można używać do testowania może wyglądać następująco.

```
public class InMemoryUnitOfWork : IUnitOfWork {
    public InMemoryUnitOfWork() {
        Committed = false;
    }
    public IObjectSet<Employee> Employees {
        get;
        set;
    }

    public IObjectSet<TimeCard> TimeCards {
        get;
        set;
    }

    public bool Committed { get; set; }
    public void Commit() {
        Committed = true;
    }
}
```

Zwrócić uwagę, że fałszywych jednostkę pracy ujawnia właściwość zatwierdzone. Czasami jest to przydatne dodać funkcje do fałszywych klasę, która ułatwić testowanie. W tym przypadku jest łatwo obserwować, jeśli kod zatwierdzenia jednostkę pracy, zaznaczając właściwość zatwierdzone.

Będziemy również potrzebować fałszywych IObjectSet<T> do przechowywania obiektów, pracowników i karty czasowej w pamięci. Oferujemy pojedynczą implementacją za pomocą typów ogólnych.

```

public class InMemoryObjectSet<T> : IObjectSet<T> where T : class
{
    public InMemoryObjectSet()
        : this(Enumerable.Empty<T>()) {
    }

    public InMemoryObjectSet(IEnumerable<T> entities) {
        _set = new HashSet<T>();
        foreach (var entity in entities) {
            _set.Add(entity);
        }
        _queryableSet = _set.AsQueryable();
    }

    public void AddObject(T entity) {
        _set.Add(entity);
    }

    public void Attach(T entity) {
        _set.Add(entity);
    }

    public void DeleteObject(T entity) {
        _set.Remove(entity);
    }

    public void Detach(T entity) {
        _set.Remove(entity);
    }

    public Type ElementType {
        get { return _queryableSet.ElementType; }
    }

    public Expression Expression {
        get { return _queryableSet.Expression; }
    }

    public IQueryProvider Provider {
        get { return _queryableSet.Provider; }
    }

    public IEnumerator<T> GetEnumerator() {
        return _set.GetEnumerator();
    }

    Ienumerator IEnumerable.GetEnumerator() {
        return GetEnumerator();
    }

    readonly HashSet<T> _set;
    readonly IQueryable<T> _queryableSet;
}

```

Ten test podwójnego deleguje większość swojej pracy, aby podstawowy zestaw HashSet<T> obiektu. Uwaga tego IObjectSet<T> wymaga ograniczenie generyczne wymuszanie T jako klasę (typ odwołania) i wymusza także NAS, aby zaimplementować interfejs IQueryable<T>. Ułatwia tworzenie kolekcji w pamięci, są traktowane jako element IQueryable<T> przy użyciu standardowego operatora zapytań LINQ AsQueryable.

Testy

Testy jednostkowe tradycyjnych użyje klasy jeden test do przechowywania wszystkich testów dla wszystkich akcji w pojedynczy kontroler MVC. Firma Microsoft można pisać testy lub dowolnego typu testu jednostkowego przy użyciu pamięci elementów sztucznych utworzyliśmy. Jednak w tym artykule, w których firma Microsoft zapobiegnie monolityczne testu klasy i zamiast tego pogrupować Nasze testy, aby skoncentrować się na konkretne funkcje. Na przykład, "Tworzenie nowego pracownika" może być funkcje, które chcemy sprawdzić, więc zostanie użyta klasa jeden test Aby zweryfikować akcji pojedynczy kontroler odpowiedzialnych za tworzenie nowego pracownika.

Brak wspólnego kodu ustawiń potrzebnych dla wszystkich tych klas testowych szczegółowe. Na przykład zawsze należy utworzyć naszych repozytoriów w pamięci i fałszywych jednostki pracy. Należy również wystąpienie kontrolera pracowników z fałszywych jednostek pracy, które są wstrzykiwane. Ten typowy kod instalacji zostaną udostępnione różnych klas testowych za pomocą klasy bazowej.

```

public class EmployeeControllerTestBase {
    public EmployeeControllerTestBase() {
        _employeeData = EmployeeObjectMother.CreateEmployees()
            .ToList();
        _repository = new InMemoryObjectSet<Employee>(_employeeData);
        _unitOfWork = new InMemoryUnitOfWork();
        _unitOfWork.Employees = _repository;
        _controller = new EmployeeController(_unitOfWork);
    }

    protected IList<Employee> _employeeData;
    protected EmployeeController _controller;
    protected InMemoryObjectSet<Employee> _repository;
    protected InMemoryUnitOfWork _unitOfWork;
}

```

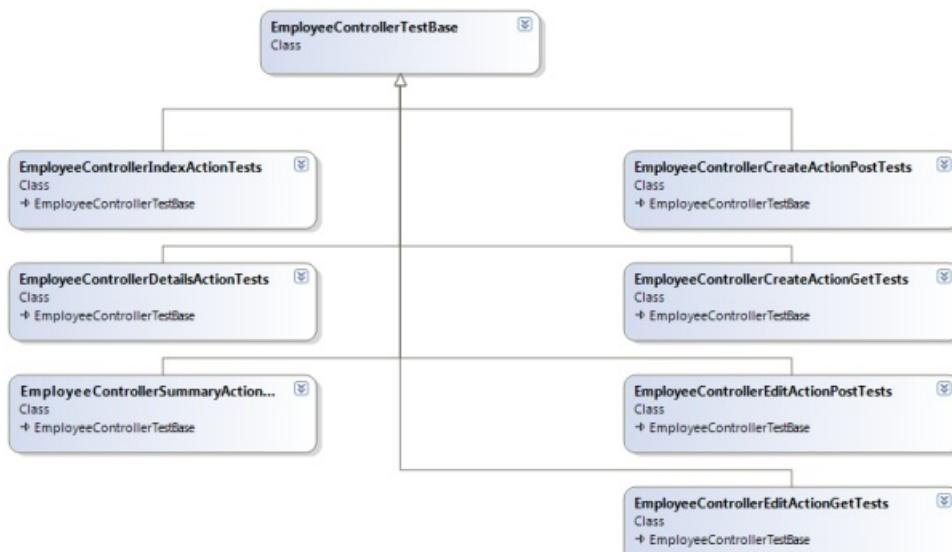
"Matka obiektu" używamy w klasie bazowej jest jeden wspólny wzorzec do tworzenia danych testowych. Matkę obiektów zawiera metodami factory do tworzenia wystąpienia testów jednostek do użycia w wielu świetlnymi testu.

```

public static class EmployeeObjectMother {
    public static IEnumerable<Employee> CreateEmployees() {
        yield return new Employee() {
            Id = 1, Name = "Scott", HireDate=new DateTime(2002, 1, 1)
        };
        yield return new Employee() {
            Id = 2, Name = "Poonam", HireDate=new DateTime(2001, 1, 1)
        };
        yield return new Employee() {
            Id = 3, Name = "Simon", HireDate=new DateTime(2008, 1, 1)
        };
    }
    // ... more fake data for different scenarios
}

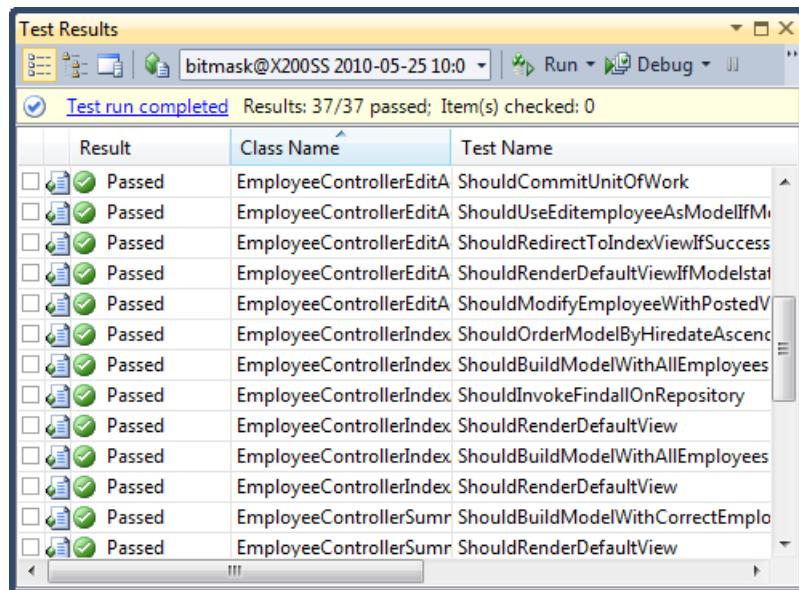
```

Możemy użyć EmployeeControllerTestBase jako klasa bazowa dla liczby świetlnymi testu (patrz rysunek 3). Każdy warunki początkowe testu przetestuje akcji określonego kontrolera. Na przykład jeden warunki początkowe testu koncentruje się na testowanie Akcja Utwórz używane podczas żądania HTTP GET (Aby wyświetlić widok na potrzeby tworzenia pracownika) i różnych warunków początkowych koncentruje się na Akcja Utwórz używana w żądaniu POST protokołu HTTP (do uwzględnienia informacji przesyłanych przez użytkownikowi utworzenie pracownika). Każdej klasy pochodnej odpowiada tylko ustalenia wymagane w tym kontekście określonego oraz w celu zapewnienia potwierdzenia potrzebne, aby zweryfikować wyniki dla kontekst określonego testu.



Rysunek 3.

Styl nazewnictwa Konwencji i testowania przedstawionych w tym miejscu nie jest wymagana dla kodu sprawdzalnego działa zgodnie — wystarczy jedno z podejść. Rysunek 4 przedstawia testów przeprowadzanych codziennie w Resharper mózgów Jet test runner wtyczki dla programu Visual Studio 2010.



Rysunek 4

Za pomocą klasy bazowej do obsługi kodu udostępnionego Instalatora testów jednostkowych dla każdej akcji kontrolera są małe i łatwe do zapisu. Testy będą wykonywane szybko (ponieważ będziemy działają w pamięci operacji) i nie powinna zakończyć się niepowodzeniem ze względu na niepowiązanych infrastruktury lub środowiskiem naturalnym (ponieważ mamy już izolowany jednostki w ramach testu).

```
[TestClass]
public class EmployeeControllerCreateActionPostTests
    : EmployeeControllerTestBase {
    [TestMethod]
    public void ShouldAddNewEmployeeToRepository() {
        _controller.Create(_newEmployee);
        Assert.IsTrue(_repository.Contains(_newEmployee));
    }
    [TestMethod]
    public void ShouldCommitUnitOfWork() {
        _controller.Create(_newEmployee);
        Assert.IsTrue(_unitOfWork.Committed);
    }
    // ... more tests

    Employee _newEmployee = new Employee() {
        Name = "NEW EMPLOYEE",
        HireDate = new System.DateTime(2010, 1, 1)
    };
}
```

W tych testach klasa bazowa wykonuje większość pracy Instalatora. Należy pamiętać, że tworzy konstruktora klasy bazowej, repozytorium w pamięci, fałszywych jednostkę pracy i wystąpienia klasy EmployeeController. Klasa testowa pochodzi z tej klasy bazowej i koncentruje się na szczegółowe informacje na temat testowania metody Create. W tym przypadku szczegółowe informacje na temat gotować "rozmieścić, działanie i asercja" czynności, pokazywany w jakiejkolwiek jednostce z badania procedury:

- Utwórz obiekt Nowy_pracownik symulowanie danych przychodzących.
- Wywołaj akcję Utwórz EmployeeController i przekazać Nowy_pracownik.
- Sprawdź, czy akcja Utwórz tworzy oczekiwanych wyników (pracownika pojawia się w repozytorium).

Co utworzyliśmy pozwala nam na wszystkie akcje EmployeeController testu. Na przykład podczas pisania testów dla akcji indeksu kontrolera pracowników firma Microsoft może dziedziczyć z klasy bazowej testu do ustanowienia tej samej podstawowej instalacji na potrzeby testów. Klasa bazowa utworzy ponownie repozytorium w pamięci, fałszywych jednostkę pracy i wystąpienia EmployeeController. Testy dla akcji indeksu wystarczy skupić się na wywoływaniu akcji indeksu i testowanie jakość modelu akcji zwraca.

```
[TestClass]
public class EmployeeControllerIndexActionTests
    : EmployeeControllerTestBase {
    [TestMethod]
    public void ShouldBuildModelWithAllEmployees() {
        var result = _controller.Index();
        var model = result.ViewData.Model
            as IEnumerable<Employee>;
        Assert.IsTrue(model.Count() == _employeeData.Count);
    }
    [TestMethod]
    public void ShouldOrderModelByHiredateAscending() {
        var result = _controller.Index();
        var model = result.ViewData.Model
            as IEnumerable<Employee>;
        Assert.IsTrue(model.SequenceEqual(
            _employeeData.OrderBy(e => e.HireDate)));
    }
    // ...
}
```

Tworzymy z substytutami w pamięci nie jest rozróżniana ukierunkowane na testowanie *stanu* oprogramowania. Na przykład podczas testowania Akcja Utwórz, chcemy, aby sprawdzić stan repozytorium, po wykonaniu akcji tworzenia — repozytorium przechowywania nowych pracownika?

```
[TestMethod]
public void ShouldAddNewEmployeeToRepository() {
    _controller.Create(_newEmployee);
    Assert.IsTrue(_repository.Contains(_newEmployee));
}
```

Później przyjrzymy interakcji na podstawie badania. Testowanie interakcji na podstawie zostanie wyświetcone pytanie, jeśli testowany kod wywoływany właściwe metody na naszych obiektów i przekazywane poprawnych parametrów. Teraz przejdziemy okładki innego wzorzec projektowy — obciążenia z opóźnieniem.

Wczesne ładowanie i powolne ładowanie

W pewnym momencie w sieci web platformy ASP.NET MVC aplikacji, którą firma Microsoft może chcieć wyświetlić informacje dotyczące pracowników i obejmują pracownika skojarzone karty godzin. Na przykład możemy mieć karty czas wyświetlania podsumowania, zawierający nazwisko pracownika oraz łącznej liczbę kart czasu systemu. Istnieje kilka rozwiązań, które możemy wykonać, aby zaimplementować tę funkcję.

Rzut

Jedno z podejść łatwy do utworzenia podsumowania jest do konstruowania modelu dedykowanych informacje, które ma być wyświetlane w widoku. W tym scenariuszu modelu może wyglądać następująco.

```
public class EmployeeSummaryViewModel {
    public string Name { get; set; }
    public int TotalTimeCards { get; set; }
}
```

Należy pamiętać, że EmployeeSummaryViewModel nie jest jednostką — innymi słowy nie jest coś, co jest potrzebne do utrwalenia w bazie danych. Tylko będziemy używać tej klasy do mieszania danych w widoku w silnie typizowany sposób. Model widoku przypomina danych przenieść obiekt (DTO), ponieważ zawiera ona nie zachowanie (nie metody) — tylko właściwości. Właściwości będą przechowywane dane, które trzeba przenieść. Jest łatwy do utworzenia wystąpienia tego modelu widoku za pomocą operatora rzutowania standard firmy LINQ — wybierz operator.

```
public ViewResult Summary(int id) {
    var model = _unitOfWork.Employees
        .Where(e => e.Id == id)
        .Select(e => new EmployeeSummaryViewModel
        {
            Name = e.Name,
            TotalTimeCards = e.TimeCards.Count()
        })
        .Single();
    return View(model);
}
```

Istnieją dwa istotne funkcje do powyższego kodu. Najpierw — kod jest łatwy do testowania, ponieważ jest wciąż łatwe do obserwowania i izolować. Wybierz operator działa równie dobrze względem naszych elementów sztucznych w pamięci, co względem rzeczywistych jednostek pracy.

```
[TestClass]
public class EmployeeControllerSummaryActionTests
    : EmployeeControllerTestBase {
    [TestMethod]
    public void ShouldBuildModelWithCorrectEmployeeSummary() {
        var id = 1;
        var result = _controller.Summary(id);
        var model = result.ViewData.Model as EmployeeSummaryViewModel;
        Assert.IsTrue(model.TotalTimeCards == 3);
    }
    // ...
}
```

Druga funkcja istotne jest, jak kod umożliwia EF4 do wygenerowania pojedynczego, wydajny kwerendy można złożyć pracowników i karty z czasu informacji ze sobą. Firma Microsoft załadowane informacje dotyczące pracowników i informacje dotyczące karty czasu do tego samego obiektu bez korzystania z żadnych specjalnych interfejsów API. Kod wyrażone jedynie informacje, których wymaga przy użyciu standardowych operatorów LINQ, które działają względem źródła danych w pamięci, a także zdalnych źródeł danych. EF4 był w stanie dokonać translacji drzew wyrażeń, generowane przez zapytanie LINQ i C# kompilatora do pojedynczych i wydajne zapytania T-SQL.

```

SELECT
[Limit1].[Id] AS [Id],
[Limit1].[Name] AS [Name],
[Limit1].[C1] AS [C1]
FROM (SELECT TOP (2)
[Project1].[Id] AS [Id],
[Project1].[Name] AS [Name],
[Project1].[C1] AS [C1]
FROM (SELECT
[Extent1].[Id] AS [Id],
[Extent1].[Name] AS [Name],
(SELECT COUNT(1) AS [A1]
FROM [dbo].[TimeCards] AS [Extent2]
WHERE [Extent1].[Id] =
[Extent2].[EmployeeTimeCard_TimeCard_Id]) AS [C1]
FROM [dbo].[Employees] AS [Extent1]
WHERE [Extent1].[Id] = @p_linq_0
) AS [Project1]
) AS [Limit1]

```

Brak innym razem, gdy firma Microsoft nie chcesz pracować z modelu widoku lub obiekt DTO, ale z rzeczywistych jednostek. Gdy wiemy, że potrzebujemy pracownika i karty godzin pracownika, firma Microsoft eagerly ładowanie powiązanych danych w sposób dyskretny kod i wydajne.

Jawne wcześnie ładowanie

Jeśli chcemy eagerly załadować informacji o powiązanych jednostek potrzebujemy mechanizmu logiki biznesowej (lub w tym scenariuszu kontroler akcji logiki) aby wyrazić chęć repozytorium. Obiekt ObjectQuery EF4<T> klasa definiuje metodę Include w taki sposób, aby określić pokrewnych obiektów do pobrania podczas wykonywania kwerendy. Należy pamiętać, EF4 ObjectContext udostępnia jednostek za pomocą konkretnego obiektu ObjectSet<T> klasy, która dziedziczy z ObjectQuery<T>. Jeśli firma Microsoft była używana w obiekcie ObjectSet<T> odwołań w naszej akcji kontrolera, napiszemy następujący kod, aby określić eager obciążenia informacje dotyczące karty czas dla każdego pracownika.

```

_employees.Include("TimeCards")
.Where(e => e.HireDate.Year > 2009);

```

Jednak ponieważ próbujemy zapewnienie naszego kodu sprawdzalnego działa zgodnie możemy się nie udostępnianie obiektu ObjectSet<T> z poza rzeczywistych jednostką pracy klasy. Zamiast tego Polegamy IObjectSet<T> interfejs, który ułatwia fałszywe, ale IObjectSet<T> nie definiuje metodę Include. Zaletą korzystania z LINQ jest, możemy utworzyć własną operator Include.

```

public static class QueryableExtensions {
    public static IQueryables<T> Include<T>
        (this IQueryables<T> sequence, string path) {
        var objectQuery = sequence as ObjectQuery<T>;
        if(objectQuery != null)
        {
            return objectQuery.Include(path);
        }
        return sequence;
    }
}

```

Zwrć uwagę, ten operator Include jest zdefiniowany jako metody rozszerzenia dla elementu IQueryables<T> zamiast IObjectSet<T>. To daje możliwość korzystania z metody z szerszego zakresu możliwych typów, w tym IQueryables<T>, IObjectSet<T>, ObjectQuery<T>, a obiekt ObjectSet<T>. W przypadku sekwencji źródłowej nie jest oryginalnym ObjectQuery EF4<T>, to nie ma żadnych szkód, wykonywane, i Include operator jest pusta. Jeśli

podstawowe sekwencji jest `ObjectQuery<T>` (lub pochodzić od `ObjectQuery<T>`), EF4 będą Zobacz nasze wymagania w zakresie dodatkowych danych i formułowanie odpowiednie SQL Zapytanie.

Za pomocą tego nowego operatora w miejscu możemy jawnie żądać eager obciążenia informacje dotyczące karty czasu z repozytorium.

```
public ViewResult Index() {
    var model = _unitOfWork.Employees
        .Include("TimeCards")
        .OrderBy(e => e.HireDate);
    return View(model);
}
```

Po uruchomieniu testów rzeczywistego obiektu `ObjectContext`, ten kod tworzy następujące pojedynczego zapytania. Zapytanie zbiera wystarczającą ilość informacji z bazy danych w jednym podróży do zmaterializowania obiektów pracowników i całkowicie wypełnić ich właściwości kart.

```
SELECT
[Project1].[Id] AS [Id],
[Project1].[Name] AS [Name],
[Project1].[HireDate] AS [HireDate],
[Project1].[C1] AS [C1],
[Project1].[Id1] AS [Id1],
[Project1].[Hours] AS [Hours],
[Project1].[EffectiveDate] AS [EffectiveDate],
[Project1].[EmployeeTimeCard_TimeCard_Id] AS [EmployeeTimeCard_TimeCard_Id]
FROM ( SELECT
    [Extent1].[Id] AS [Id],
    [Extent1].[Name] AS [Name],
    [Extent1].[HireDate] AS [HireDate],
    [Extent2].[Id] AS [Id1],
    [Extent2].[Hours] AS [Hours],
    [Extent2].[EffectiveDate] AS [EffectiveDate],
    [Extent2].[EmployeeTimeCard_TimeCard_Id] AS
        [EmployeeTimeCard_TimeCard_Id],
    CASE WHEN ([Extent2].[Id] IS NULL) THEN CAST(NULL AS int)
    ELSE 1 END AS [C1]
    FROM [dbo].[Employees] AS [Extent1]
    LEFT OUTER JOIN [dbo].[TimeCards] AS [Extent2] ON [Extent1].[Id] = [Extent2].
[EmployeeTimeCard_TimeCard_Id]
    ) AS [Project1]
ORDER BY [Project1].[HireDate] ASC,
[Project1].[Id] ASC, [Project1].[C1] ASC
```

Mamy znakomitą wiadomość jest całkowicie sprawdzalnego działa zgodnie pozostaje kod wewnętrz metody akcji. Nie potrzebujemy Podaj wszelkie dodatkowe funkcje dla naszych elementów sztucznych na obsługuje operatora `Include`. Złe wiadomości jest boss Said WE had użycie operatora `Include` wewnętrz kodu, Chcieliśmy, aby zapewnić trwałość zakresu. Jest to podstawowy przykład typu skutków ubocznych, które będą potrzebne do wzięcia pod uwagę podczas kompilowania kodu sprawdzalnego działa zgodnie. Istnieją terminy, niezbędne, aby umożliwić przeciek wątpliwości trwałości poza abstrakcją repozytorium, aby spełnić cele dotyczące wydajności.

Alternatywa dla wcześnie ładowanie jest powolne ładowanie. Powolne ładowanie oznacza, że robimy *nie* naszych firm kod potrzebny do jawnie poinformować o konieczności powiązane dane. Zamiast tego stosujemy nasze jednostek w aplikacji, a jeśli dodatkowe dane potrzebne Entity Framework załaduje dane na żądanie.

Ładowanie z opóźnieniem

To ułatwia Wyobraź sobie scenariusz, w których nie można określić, co będzie potrzebne dane dany fragment logiki biznesowej. Firma Microsoft znać logiki wymaga obiektu pracowników, ale firma Microsoft może gałąź do wykonywania różnych ścieżek, gdzie niektóre z tych ścieżek wymagają informacji o karcie czas od pracownika, a niektóre nie. Scenariusze, takie jak to są idealne dla niejawnego powolne ładowanie ponieważ magiczny sposób

wyświetlania danych na zgodnie z potrzebami.

Powolne ładowanie, znany także jako odroczone ładowanie, umieść pewne wymagania na naszych obiektach jednostki. POCOs z nieznajomością trwałości wartością true, nie będzie twarzy wszelkie wymagania z warstwy trwałości, ale nieznajomością trwałości wartością true, jest praktycznie niemożliwe do osiągnięcia. Zamiast tego mierzymy nieznajomością trwałości w stopniach względnych. Byłoby niefortunne potrzbowałyśmy dziedziczącą z klasy bazowej zorientowanej na trwałości lub użyć wyspecjalizowanej kolekcji, aby osiągnąć powolne ładowanie w POCOs. Na szczęście EF4 ma płynniejsza rozwiązania.

Praktyczne wykryć

Używanie obiektów POCO, EF4 można dynamicznie generować proxy środowiska uruchomieniowego dla jednostki. Te serwery proxy niewidocznie opakować POCOs zmateriałizowany i zapewnić uzyskanie dodatkowe usługi przechwycenie każdej właściwości i ustawienie operację do wykonania dodatkowej pracy. Takie usługi jest funkcja ładowania z opóźnieniem, którą firma Microsoft szuka. Inna usługa jest wydajny mechanizm, który może zapisać zmianie wartości właściwości jednostki, program śledzenia zmian. Lista zmian jest używana przez obiekt ObjectContext podczas metodę SaveChanges, aby utrważyć wszystkie jednostki modyfikacji, przy użyciu polecenia aktualizacji.

Dla tych serwerów proxy do pracy jednak potrzebują sposób do połączenia do właściwości get i ustawiania wykonywanych względem jednostki a serwerami proxy osiągnięcie tego celu przez zastąpienie wirtualnych elementów członkowskich. W związku z tym jeśli chcemy masz niejawnej powolne ładowanie i wydajny śledzenia zmian należy przejść z powrotem do naszych POCO definicje klas i oznacz właściwości jako wirtualny.

```
public class Employee {
    public virtual int Id { get; set; }
    public virtual string Name { get; set; }
    public virtual DateTime HireDate { get; set; }
    public virtual ICollection<TimeCard> TimeCards { get; set; }
}
```

Firma Microsoft może nadal wskazuje, że jednostki pracowników jest przede wszystkim trwałości zakresu. Jedynym wymaganiem jest, aby użyć wirtualnych elementów członkowskich i nie ma to wpływu na testowanie kodu. Firma Microsoft nie muszą pochodzić z żadnych specjalnych klasy bazowej lub nawet za pomocą specjalnych kolekcji przeznaczone do ładowania z opóźnieniem. Tak jak pokazano kod, każda klasa implementacji ICollection<T> jest dostępna na potrzeby przechowywania powiązanych jednostek.

Istnieje również jeden drobnej zmiany, które należy wprowadzić w naszych jednostki pracy. Powolne ładowanie jest *poza domyślnie*, pracując bezpośrednio z obiektem ObjectContext. Jest właściwością, możemy ustawić właściwości ContextOptions umożliwia odroczone ładowanie i możemy ustawić tę właściwość w naszym rzeczywistych jednostkę pracy, jeśli chcemy Włącz powolne ładowanie wszędzie.

```
public class SqlUnitOfWork : IUnitOfWork {
    public SqlUnitOfWork() {
        // ...
        _context = new ObjectContext(connectionString);
        _context.ContextOptions.LazyLoadingEnabled = true;
    }
    // ...
}
```

Przy użyciu niejawnego ładowania z opóźnieniem, włączone, kod aplikacji może użyć pracownika pracownika związków oraz kart czas pozostając bliżej świadomości ilości pracy wymaganej dla platformy EF załadować dodatkowe dane.

```

var employee = _unitOfWork.Employees
    .Single(e => e.Id == id);
foreach (var card in employee.TimeCards) {
    // ...
}

```

Powolne ładowanie sprawia, że kod aplikacji jest łatwiejsze do zapisu, a proxy magic kod nie będzie całkowicie sprawdzalnego działa zgodnie. Substytuty w pamięci jednostki pracy po prostu wstępnie ładowania fałszywych jednostki z powiązane dane, gdy potrzebne podczas testu.

W tym momencie będziesz Włącz naszej uwagi od tworzenia repozytoriów za pomocą `IObjectSet<T>` i przyjrzyj się elementy abstrakcji, aby ukryć wszystkie znaki Framework trwałości.

Niestandardowe repozytoria

Umieszczeniem możemy najpierw jednostki pracy wzorcu projektowym w tym artykule zamieszczone przykładowy kod dla jak może wyglądać jednostkę pracy. Umożliwia ponowne przedstawia tę koncepcję oryginalny, używając pracowników i pracowników karta godzin scenariusza, który możemy masz doświadczenie w pracy z.

```

public interface IUnitOfWork {
    IRepository<Employee> Employees { get; }
    IRepository<TimeCard> TimeCards { get; }
    void Commit();
}

```

Główną różnicą między tej jednostki pracy i jednostki pracy utworzonych w ostatniej sekcji jest jak tej jednostki pracy nie używa żadnych abstrakcje platformy EF4 (nie ma żadnych `IObjectSet<T>`). `IObjectSet<T>` działa również jako interfejs repozytorium, ale interfejs API udostępnia ona doskonale mogą być niewyrównane naszej aplikacji potrzebom. W tym podejściu nadchodzących firma Microsoft będzie reprezentować repozytoriów przy użyciu niestandardowych `IRepository<T>` abstrakcji.

Wielu programistów, którzy postępuj zgodnie z projektowania opartego na testach, projektowania opartego na zachowanie i projektowanie metodologii opartego na domenach Preferuj `IRepository<T>` podejścia z kilku powodów. Najpierw `IRepository<T>` interfejs reprezentuje "" warstwa przeciwdogradacyjna. Zgodnie z opisem w Eric Evans w książce projektowania opartego na domenie warstwy przeciwdogradacyjnej przechowuje kod domeny od infrastruktury interfejsów API, np. trwałość interfejsu API. Po drugie deweloperzy mogą tworzyć metody do repozytorium, które potrzeb dokładnie aplikacji (jak wykryte podczas pisania testów). Na przykład może być często potrzebujemy zlokalizować pojedynczą jednostkę przy użyciu wartości Identyfikatora, abyśmy mogli dodać metodę `FindById` interfejsu repozytorium. Nasze `IRepository<T>` definicja będzie wyglądać podobnie do poniższego.

```

public interface IRepository<T>
    where T : class, IEntity {
    IQueryable<T> FindAll();
    IQueryable<T> FindWhere(Expression<Func<T, bool>> predicate);
    T FindById(int id);
    void Add(T newEntity);
    void Remove(T entity);
}

```

Zwróć uwagę, firma Microsoft będzie ponownie upuścić, aby za pomocą element `IQueryable<T>` interfejsu do udostępnienia kolekcji jednostek. Element `IQueryable<T>` umożliwia drzew wyrażeń LINQ przepływać do dostawcy EF4 i dostawcy holistycznego widoku zapytania. Drugą opcją będzie zwracać `IEnumerable<T>`, co oznacza, że dostawcy EF4 LINQ będą widzieć tylko wyrażenia utworzone wewnątrz repozytorium. Wszelkie grupowania, kolejność i projekcji wykonywane poza repozytorium nie będzie składa do polecenia SQL wysyłane do bazy danych, która może obniżyć wydajność. Z drugiej strony, repozytorium, zwracając tylko `IEnumerable<T>`

wyniki będą nigdy nie Cię zaskoczyć, za pomocą nowego polecenia SQL. W obu przypadkach efekt będzie działać, a oba podejścia nadal sprawdzalnego działa zgodnie.

Jest prosta zapewnić pojedynczą implementację `IRepository<T>` interfejsa, za pomocą typów ogólnych i EF4 API obiektu `ObjectContext`.

```
public class SqlRepository<T> : IRepository<T>
    where T : class, IEntity {
    public SqlRepository(ObjectContext context) {
        _objectSet = context.CreateObjectSet<T>();
    }
    public IQueryable<T> FindAll() {
        return _objectSet;
    }
    public IQueryable<T> FindWhere(
        Expression<Func<T, bool>> predicate) {
        return _objectSet.Where(predicate);
    }
    public T FindById(int id) {
        return _objectSet.Single(o => o.Id == id);
    }
    public void Add(T newEntity) {
        _objectSet.AddObject(newEntity);
    }
    public void Remove(T entity) {
        _objectSet.DeleteObject(entity);
    }
    protected ObjectSet<T> _objectSet;
}
```

`IRepository<T>` podejście zapewnia nam niektóre dodatkową kontrolę nad naszego zapytania, ponieważ klient musi wywołać metodę, aby uzyskać dostęp do jednostki. Wewnątrz metody możemy to zapewnić dodatkowe czynności kontrolne i operatorów LINQ do wymuszania ograniczeń aplikacji. Zwróć uwagę, że interfejs ma dwa ograniczenia dla parametru typu ogólnego. Pierwszy ograniczeniem jest to klasa zmiany barwy wad, wymagane przez obiekt `ObjectSet<T>`, i drugi ograniczenie wymusza naszych jednostek do zaimplementowania `IEntity` — abstrakcję utworzony dla aplikacji. Interfejs `IEntity` wymusza podmiotów odczytywalną właściwość `Id`, a następnie możemy użyć tej właściwości w metodzie `FindById`. `IEntity` jest zdefiniowana z następującym kodem.

```
public interface IEntity {
    int Id { get; }
}
```

`IEntity` może zostać uznana za mały naruszenie nieznajomości trwałości, ponieważ naszych jednostki są wymagane do zaimplementowania interfejsu. Należy pamiętać nieznajomości trwałości dotyczy wady i zalety i dla wielu funkcji `FindById` będzie przeważają ograniczenia nałożone przez interfejs. Interfejs nie ma wpływu na testowania.

Utworzenie wystąpienia na żywo `IRepository<T>` wymaga obiektu `ObjectContext` EF4, dlatego konkretnej jednostki pracy wdrożenia należy zarządzać procesu tworzenia wystąpienia.

```

public class SqlUnitOfWork : IUnitOfWork {
    public SqlUnitOfWork() {
        var connectionString =
            ConfigurationManager
                .ConnectionStrings[ConnectionStringName]
                .ConnectionString;

        _context = new ObjectContext(connectionString);
        _context.ContextOptions.LazyLoadingEnabled = true;
    }

    public IRepository<Employee> Employees {
        get {
            if (_employees == null) {
                _employees = new SqlRepository<Employee>(_context);
            }
            return _employees;
        }
    }

    public IRepository<TimeCard> TimeCards {
        get {
            if (_timeCards == null) {
                _timeCards = new SqlRepository<TimeCard>(_context);
            }
            return _timeCards;
        }
    }

    public void Commit() {
        _context.SaveChanges();
    }

    SqlRepository<Employee> _employees = null;
    SqlRepository<TimeCard> _timeCards = null;
    readonly ObjectContext _context;
    const string ConnectionStringName = "EmployeeDataModelContainer";
}

```

Przy użyciu niestandardowych repozytorium

Z pomocą naszych niestandardowe repozytorium nie jest znacznie różnią się od przy użyciu repozytorium, w oparciu o `IObjectSet<T>`. Zamiast bezpośrednio do właściwości z zastosowaniem operatorów LINQ, najpierw musisz wywołać za pomocą jednego z repozytorium metod do pobrania element `IQueryable<T>` odwołania.

```

public ViewResult Index() {
    var model = _repository.FindAll()
        .Include("TimeCards")
        .OrderBy(e => e.HireDate);
    return View(model);
}

```

Zwróć uwagę, że niestandardowy operator `Include`, wcześniej zaimplementowane będzie działać bez zmian. Metoda `FindByld` z repozytorium usuwa zduplikowane logiki z akcji próby pobrania pojedynczej jednostki.

```

public ViewResult Details(int id) {
    var model = _repository.FindByld(id);
    return View(model);
}

```

Nie ma znaczące różnic w testowania dwa podejścia, w których firma Microsoft zostały zbadane. Zapewniamy fałszywych implementacje `IRepository<T>` przez utworzenie klas konkretnych wspierana przez zestaw

`HashSet<pracowników>` — podobnie jak zrobiliśmy w ostatniej sekcji. Jednak niektórzy deweloperzy wolą używać makiety obiektów i testowanie struktur obiektu zamiast tworzenia elementów sztucznych. Zapoznamy się przy użyciu mocks do testowania naszej implementacji i omówiono różnice między mocks i sztucznych elementów w następnej sekcji.

Testowanie za pomocą Mocks

Istnieją różne metody do tworzenia wywołań jakie Martina Fowlera "test podwójnego". Test podwójnego (na przykład stunt filmu double) jest obiektem tworzonych "występować w" rzeczywiste, obiekty w środowisku produkcyjnym podczas testów. Repozytoriów w pamięci, którą utworzyliśmy są testu wartości podwójnej precyzji dla repozytoriów, komunikujące się z programu SQL Server. Pokazaliśmy już, jak za pomocą tych podwaja testów podczas testów jednostkowych izolowania kodu i zachować testów przeprowadzanych codziennie przez szybkie.

Badanie wartości podwójnej precyzji, którą utworzyliśmy ma implementacje rzeczywistych, praca. W tle każdego z nich przechowuje kolekcję konkretnych obiektów i będą oni dodawać i usuwać obiekty z tej kolekcji, jak możemy manipulować repozytorium podczas testu. Niektórzy deweloperzy, takich jak tworzenie ich podwaja się test temu — przy użyciu rzeczywistego kodu i implementacje pracy. Te testu wartości podwójnej precyzji są tak zwany *elementów sztucznych*. Mają one implementacje pracy, ale nie są prawdziwe, do użytku produkcyjnego.

Repozytorium fałszywych faktycznie nie zapisu w bazie danych. Serwer SMTP fałszywych faktycznie nie Wyślij wiadomość e-mail za pośrednictwem sieci.

Mocks w porównaniu z elementów sztucznych

Istnieje inny typ testu double nazywane *testowanie*. Chociaż elementów sztucznych implementacje pracy, mocks są dostarczane z implementacją. Za pomocą framework makiety obiektu możemy konstruowania tych makiety obiektów w czasie wykonywania i używać ich jako wartości podwójnej precyzji testu. W tej sekcji będziemy używać "open source" pozorowanie framework Moq. Poniżej przedstawiono prosty przykład użycia Moq umożliwia dynamiczne tworzenie test podwójnego repozytorium pracownika.

```
Mock< IRepository<Employee>> mock =
    new Mock< IRepository<Employee>>();
 IRepository<Employee> repository = mock.Object;
 repository.Add(new Employee());
 var employee = repository.FindById(1);
```

Poprosimy Moq dla `IRepository<pracowników>` implementacji i tworzy jeden dynamicznie. Firma Microsoft może uzyskać dostęp do obiekt implementujący `IRepository<pracowników>`, uzyskując dostęp do właściwości obiektu pozorny `<T>` obiektu. Jest ten obiekt wewnętrzny, które możemy przekazać do naszych kontrolerów, a nie będą wiedzieli, jeśli jest to test podwójnego lub rzeczywistego repozytorium. Firma Microsoft wywoływać metody na obiekt, tak samo, jak firma Microsoft będzie wywoływać metody na obiekt z rzeczywistej implementacji.

Możesz się zastanawiać, co makiety repozytorium zrobić po możemy wywołać metody Add. Ponieważ nie ma żadnej implementacji za makiety obiektu, Dodaj nic nie robi. Brak konkretnego kolekcji w tle, takie jak mieliśmy z substytutami, którą napisaliśmy, więc jest odrzucana pracownika. Jak wygląda wartość zwracaną `FindById`? W tym przypadku makiety obiektu jest jedyną czynnością, którą wykonania, która jest zwracana wartość domyślną. Ponieważ firma Microsoft jest zwracany typ odwołania (pracownika), wartość zwracana jest wartość null.

Mocks może dźwiękowych bezwartościowe; Istnieją jednak dwa więcej funkcji mocks, które jeszcze nie Omówiliśmy. Po pierwsze Moq framework rejestruje wszystkie wywołania makiety obiektu. W dalszej części kodu możemy zadawać Moq, czy każdy użytkownik wywołał metodę Add, czy każdy użytkownik wywołał metodę `FindById`. Zobaczmy, pokażemy, jak możemy użyć tej funkcji "czarne pole" Rejestrowanie w testach.

Druga funkcja doskonałe jest, jak możemy użyć Moq program makiety obiektu z oczekiwaniem. Oczekiwanie informuje makiety obiektu, jak reagować na interakcji z danym. Na przykład możemy zaprogramować Oczekiwanie w naszym pozornym i stwierdzenie, aby zwrócić obiekt pracowników, gdy ktoś wywołuje `FindById`. Środowisko Moq wykorzystuje interfejs API konfiguracji i wyrażeń lambda z programem tego oczekiwania.

```
[TestMethod]
public void MockSample() {
    Mock< IRepository< Employee >> mock =
        new Mock< IRepository< Employee >>();
    mock.Setup(m => m.FindById(5))
        .Returns(new Employee { Id = 5 });
    IRepository< Employee > repository = mock.Object;
    var employee = repository.FindById(5);
    Assert.IsTrue(employee.Id == 5);
}
```

W tym przykładzie prosimy Moq dynamicznie Tworzenie repozytorium, a następnie będziemy programować repozytorium z oczekiwaniemi. Oczekuje informuje makiety obiekt do zwrotu nowy obiekt pracowników z wartością identyfikatora 5, gdy ktoś wywołuje metodę FindById, przekazując wartość 5. Ten test zakończy się pomyślnie, a firma Microsoft nie trzeba tworzyć pełną implementację do IRepository fałszywych<T>.

Spróbujmy ponownie testy, którą napisaliśmy wcześniej i poprawkami, a ich do używania mocks zamiast elementów sztucznych. Podobnie jak wcześniej, użyjemy klasę bazową można skonfigurować wspólne elementy infrastruktury potrzebnych dla wszystkich kontrolera testów.

```
public class EmployeeControllerTestBase {
    public EmployeeControllerTestBase() {
        _employeeData = EmployeeObjectMother.CreateEmployees()
            .AsQueryable();
        _repository = new Mock< IRepository< Employee >>();
        _unitOfWork = new Mock< IUnitOfWork >();
        _unitOfWork.Setup(u => u.Employees)
            .Returns(_repository.Object);
        _controller = new EmployeeController(_unitOfWork.Object);
    }

    protected IQueryable< Employee > _employeeData;
    protected Mock< IUnitOfWork > _unitOfWork;
    protected EmployeeController _controller;
    protected Mock< IRepository< Employee >> _repository;
}
```

Kod ustawienia pozostają takie same. Zamiast korzystać z elementów sztucznych, użyjemy Moq do konstruowania obiektów makiety. Klasa bazowa organizuje makiety jednostka pracy do zwrócenia makiety repozytorium, gdy kod wywołuje właściwość pracowników. Pozostała konfiguracja makiety odbędzie się wewnątrz świetlnymi testów przeznaczonych dla każdego konkretnego scenariusza. Na przykład warunki początkowe testu dla akcji indeksu skonfiguruje makiety repozytorium, aby zwrócić listę pracowników, gdy akcja wywołuje metodę FindAll makiety repozytorium.

```

[TestClass]
public class EmployeeControllerIndexActionTests
    : EmployeeControllerTestBase {
    public EmployeeControllerIndexActionTests() {
        _repository.Setup(r => r.FindAll())
            .Returns(_employeeData);
    }
    // ... tests
    [TestMethod]
    public void ShouldBuildModelWithAllEmployees() {
        var result = _controller.Index();
        var model = result.ViewData.Model
            as IEnumerable<Employee>;
        Assert.IsTrue(model.Count() == _employeeData.Count());
    }
    // ... and more tests
}

```

Z wyjątkiem oczekiwania Nasze testy wyglądają podobnie do testów, które Musieliszyśmy przed. Jednak dzięki możliwości rejestrowania makiety Framework możemy podejśćie, testowanie pod kątem różnych. Omówimy to Nowa perspektywa w następnej sekcji.

Stan i testowanie interakcji

Istnieją różne techniki, które można użyć do testowania oprogramowania za pomocą makiety obiektów. Jednym z podejść jest stanu na podstawie badań, czyli co możemy zostały wykonane w tym dokumencie do tej pory. Stan oparty testowania potwierdzenia sprawia, że o stanie tego oprogramowania. W ostatni test firma Microsoft wywoływanie metody akcji kontrolera, a wprowadzone potwierdzenie o modelu, należy go tworzyć. Poniżej przedstawiono kilka przykładów stanu testów:

- Sprawdź, czy repozytorium zawiera nowy obiekt pracownika po wykonaniu Utwórz.
- Sprawdź, czy model zawiera listę wszystkich pracowników, po wykonaniu indeksu.
- Sprawdź, czy po wykonaniu Usuwanie repozytorium nie zawiera danego pracownika.

Innym podejściem zostanie wyświetlony z makiety obiektów jest sprawdzenie *interakcje*. Gdy stan na podstawie testowania potwierdzenia sprawia, że o stanie obiektów, interakcji na podstawie testowania potwierdzenia sprawia, że dotyczące sposobu interakcji obiektów. Na przykład:

- Sprawdź, czy kontroler wywołuje metody Add z repozytorium, gdy wykonuje Utwórz.
- Sprawdź, czy kontroler wywołuje metodę FindAll z repozytorium, gdy indeks.
- Sprawdź, czy kontroler wywołuje jednostki szkoły wywołano metody Commit można zapisać zmian, gdy wykonuje edycji.

Często testowania interakcji wymaga mniejszej ilości danych testu, ponieważ firma Microsoft nie są wywiercenie wewnętrz kolekcji i weryfikowanie liczby. Na przykład gdy wiemy, że akcja szczegóły wywołuje metodę FindById repozytorium przy użyciu poprawnej wartości - następnie akcji jest prawdopodobnie działa prawidłowo. Bez konfigurowania żadnych danych testowych do zwrócenia z FindById, możemy sprawdzić to zachowanie.

```

[TestClass]
public class EmployeeControllerDetailsActionTests
    : EmployeeControllerTestBase {
    // ...
    [TestMethod]
    public void ShouldInvokeRepositoryToFindEmployee() {
        var result = _controller.Details(_detailsId);
        _repository.Verify(r => r.FindById(_detailsId));
    }
    int _detailsId = 1;
}

```

Tylko ustawienia wymagane w powyższe warunki początkowe testu jest Instalator dostarczonych przez klasę bazową. Gdy firma Microsoft wywołanie akcji kontrolera, Moq zarejestruje interakcje makiety repozytorium. Za pomocą interfejsu API Sprawdź Moq, możemy zadawać Moq. Jeśli kontroler wywołana FindById przy użyciu prawidłowego wartość Identyfikatora. Jeśli kontrolera nie wywołała metodę lub wywołano metodę z wartością parametru nieoczekiwany, sprawdź, czy metoda zgłosi wyjątek, a test zakończy się niepowodzeniem.

Oto inny przykład, aby sprawdzić, czy akcja Utwórz wywołuje zatwierdzenia dla bieżącej jednostki pracy.

```
[TestMethod]
public void ShouldCommitUnitOfWork() {
    _controller.Create(_newEmployee);
    _unitOfWork.Verify(u => u.Commit());
}
```

Jeden zagrożenia z testowaniem interakcji jest tendencja za pośrednictwem określić interakcje. Możliwość makiety obiektu do rejestrowania i sprawdź, czy każdy interakcja z makiety obiektu nie oznacza testu starać się zweryfikować każdej interakcji. Niektóre interakcje są szczegóły dotyczące implementacji i interakcji należy zweryfikować tylko *wymagane* do zaspokojenia bieżącego testu.

Wybór między mocks lub elementów sztucznych dużym stopniu zależy od systemu, który testujesz i osobistej (lub zespołu kart) preferencje. Obiekty makiety może znacznie zmniejszyć ilość kodu, musisz wdrożyć test wartości podwójnej precyzji, ale nie wszyscy jest komfortowo, jednocześnie programowania oczekiwania i weryfikowanie interakcje.

Wnioski

W tym dokumencie firma Microsoft została przedstawiona różne podejścia do tworzenia kodu sprawdzalnego działa zgodnie z podczas korzystania z programu Entity Framework ADO.NET dla funkcji trwałości danych. Firma Microsoft mogą korzystać z wbudowanymi abstrakcjami takich jak IObjectSet<T>, lub utworzyć własne elementy abstrakcji takich jak IRepository<T>. W obu przypadkach te elementy abstrakcji konsumentom stale trwały dzięki obsłudze POCO w ADO.NET Entity Framework 4.0, zakresu i wysoce testowalna. Dodatkowe funkcje EF4, takich jak niejawnego powolne ładowanie umożliwiają usługi biznesowe kodu do pracy bez konieczności martwienia się o szczegółowe informacje o relacyjnego magazynu danych. Na koniec abstrakcji, tworzonej przez nas są łatwe do makiety lub fałszywe wewnętrznych testów jednostkowych i możemy użyć tych testów wartości podwójnej precyzji do osiągnięcia szybkiego uruchamiania, wysoce izolowane i niezawodne testy.

Dodatkowe zasoby

- Robert C. Martin, " [zasady pojedynczej odpowiedzialności](#)"
- Martina Fowler [katalog wzorców z wzorców architektury aplikacji przedsiębiorstwa](#)
- Griffin Caprio " [wstrzykiwanie zależności](#)"
- Grupa dyskusyjna danych, " [Instruktaż: testów opartych na tworzenie aplikacji przy użyciu programu Entity Framework 4.0](#)".
- Grupa dyskusyjna danych, " [wzorców przy użyciu repozytorium i jednostki pracy za pomocą programu Entity Framework 4.0](#)"
- Dave Astels " [wprowadzenie BDD](#)"
- Aaron Jensen " [wprowadzenie do specyfikacji maszyny](#)"
- Eric Lee " [BDD narzędzia MSTest](#)"
- Eric Evans, " [projektowania opartego na domenach](#)"
- Martina Fowler " [Mocks nie są wycinków](#)"
- Martina Fowler " [Test podwójnego](#)"
- Jeremy Miller, Dyrektor ds " [stanu i testowanie interakcji](#)"
- [Moq](#)

Biografia

Scott Allen jest członek personelu technicznego w firmie Pluralsight i założycielem OdeToCode.com. 15 lat Wytwarzanie oprogramowania komercyjnego Scott pracował nad rozwiązańa związane z 8-bitową urządzeń osadzonych do wysoce skalowalnych aplikacji sieci web platformy ASP.NET. Możesz docierać do Scotta na swoim blogu w OdeToCode lub Matta na Twitterze <http://twitter.com/OdeToCode> .

Tworzenie modelu

27.09.2018 • 5 minutes to read • [Edit Online](#)

Szczegółowe informacje dotyczące sposobu mapowania właściwości i klasy aplikacji do bazy danych, tabele i kolumny są przechowywane w modelu platformy EF. Istnieją dwa główne sposoby tworzenia modelu platformy EF:

- **Za pomocą funkcji Code First:** Deweloper pisze kod, aby określić model. EF generuje modeli i mapowania na środowisko uruchomieniowe oparte na klasach jednostki i modelu dodatkowych konfiguracji dostarczoną przez dewelopera.
- **Za pomocą projektanta EF:** Deweloper Rysuje linie, aby określić model przy użyciu projektanta EF i pola. Wynikowy modelu są przechowywane jako kod XML w pliku z rozszerzeniem EDMX. Obiekty domeny aplikacji są zwykle generowane automatycznie na podstawie modelu koncepcyjnego.

Przepływy pracy EF

Obie metody może służyć do docelowych istniejącą bazę danych lub Utwórz nową bazę danych, co 4 różnych przepływów pracy. Dowiedzieć się o tym, które w jednym jest najbardziej odpowiedni dla Ciebie:

	CHCĘ NAPISAĆ KOD...	CHCĘ UŻYĆ PROJEKTANTA...
Będę tworzyć nową bazę danych	Użyj Code First do zdefiniowania modelu w kodzie, a następnie wygenerować bazę danych.	Użyj pierwszy Model do zdefiniowania modelu za pomocą linii i pola, a następnie wygenerować bazę danych.
Chcę korzystać z istniejącej bazy danych	Użyj Code First do utworzenia modelu na podstawie kodu, który mapuje do istniejącej bazy danych.	Użyj Database First do tworzenia modeli linii i pola, który jest mapowany do istniejącej bazy danych.

Obejrzyj film wideo: jakie EF przepływu pracy należy używać?

Ten krótki film wideo wyjaśnia różnice i jak znaleźć ten, który jest odpowiedni dla Ciebie.

Osoba prezentująca: [Rowan Miller](#)



[WMV](#) | [MP4](#) | [WMV \(ZIP\)](#)

Jeśli po obejrzeniu filmu, które nadal nie czujesz przy wyborze rozwiązania, jeśli chcesz użyć projektanta EF Code First Naucz jednocześnie.

Wygląd kulisy

Niezależnie od tego, czy używasz Code First i projektancie platformy EF modelu platformy EF zawsze ma kilka składników:

- Obiektów domeny lub jednostki aplikacji typy samodzielne. To jest często nazywany warstwy obiektu
- Model koncepcyjny składający się z typami encji specyficznego dla domeny i relacje, za pomocą [modelu Entity Data Model](#). Ta warstwa sytuację najczęściej nazywa się literą "C" dla *koncepcyjny*.
- Model magazynu reprezentujących tabele, kolumny i relacje, zgodnie z definicją w bazie danych. Ta warstwa

sytuację najczęściej nazywa się przy użyciu nowszej "S", dla *magazynu*.

- Mapowanie między modelu koncepcyjnego i schemat bazy danych. To mapowanie jest często nazywany mapowania "C-S".

Aparat mapowania firmy EF wykorzystuje "C-S" Mapowanie do przekształcania operacji wykonywanych względem jednostek — takie jak tworzenie, odczytu, aktualizacji i usuwania - w do równoważne operacji dla tabel w bazie danych.

Mapowanie między modelu koncepcyjnego i obiekty aplikacji często nazywa się "O-C" mapowanie. W porównaniu do mapowania "C-S", "O-C" jest mapowanie jeden do jednego i niejawne: jednostki, właściwości i relacje zdefiniowane w modelu koncepcyjnym muszą być zgodne, kształty i typy obiektów platformy .NET. Z EF4 i poza warstwy obiektów może składać się z prostych obiektów za pomocą właściwości bez wszelkie zależności od platformy EF. Te są zwykle określane jako zwykły stary obiektów CLR (POCO) i wykonywane jest mapowanie typów i właściwości na nazwę pasującą Konwencji. Wcześniej w wersji 3.5 EF wystąpiły określone ograniczenia dla warstwy obiektów, takich jak jednostki konieczności pochodzi od klasy EntityObject i konieczności wykonania EF atrybutów, aby zaimplementować mapowanie "O-C".

Najpierw kod do nowej bazy danych

27.09.2018 • 18 minutes to read • [Edit Online](#)

W tym przewodniku krok po kroku i wideo zawierają wprowadzenie do rozwiązania deweloperskiego Code First dla nowej bazy danych. Ten scenariusz obejmuje przeznaczonych dla bazy danych, która nie istnieje i utworzy Code First lub pustą bazę danych tej Code First doda nowe tabele, aby. Kod umożliwia najpierw zdefiniować model za pomocą języka C# lub klasy VB.Net. Opcjonalnie można wykonać dodatkowe czynności konfiguracyjne przy użyciu atrybutów klas i właściwości lub za pomocą interfejsu API fluent.

Obejrzyj wideo

To wideo zawiera wprowadzenie do rozwiązania deweloperskiego Code First dla nowej bazy danych. Ten scenariusz obejmuje przeznaczonych dla bazy danych, która nie istnieje i utworzy Code First lub pustą bazę danych tej Code First doda nowe tabele, aby. Najpierw kod pozwala na zdefiniowanie modelu przy użyciu klas języka C# lub VB.Net. Opcjonalnie można wykonać dodatkowe czynności konfiguracyjne przy użyciu atrybutów klas i właściwości lub za pomocą interfejsu API fluent.

Osoba prezentująca: Rowan Miller

Film wideo: [WMV](#) | [MP4](#) | [WMV \(ZIP\)](#)

Wymagania wstępne

Musisz mieć co najmniej programu Visual Studio 2010 lub Visual Studio 2012 są zainstalowane w tym przewodniku.

Jeśli używasz programu Visual Studio 2010, należy również mieć [NuGet](#) zainstalowane.

1. Tworzenie aplikacji

Aby zachować ich prostotę zamierzamy utworzyć aplikację podstawowa konsola, która używa Code First do dostępu do danych.

- Otwórz program Visual Studio
- **Plik —> New -> projekt...**
- Wybierz **Windows** menu po lewej stronie i **aplikacji konsoli**
- Wprowadź **CodeFirstNewDatabaseSample** jako nazwę
- Wybierz **OK**

2. Tworzenie modelu

Czynnością jest zdefiniowanie bardzo prosty model przy użyciu klas. Wystarczy zdefiniować je w pliku Program.cs, ale w rzeczywistej aplikacji, w Twoich zajęciach się będzie podzielić na osobne pliki i potencjalnie oddzielnego projektu.

Poniżej definicji klasy programu w pliku Program.cs Dodaj następujące dwie klasy.

```

public class Blog
{
    public int BlogId { get; set; }
    public string Name { get; set; }

    public virtual List<Post> Posts { get; set; }
}

public class Post
{
    public int PostId { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }

    public int BlogId { get; set; }
    public virtual Blog Blog { get; set; }
}

```

Można zauważyć, że wprowadzamy dwie właściwości nawigacji (Blog.Posts i Post.Blog) wirtualnego. Dzięki temu funkcja powolne ładowanie programu Entity Framework. Powolne ładowanie oznacza, że zawartość te właściwości zostaną automatycznie załadowane z bazy danych przy próbie uzyskiwać do nich dostęp.

3. Utwórz kontekst

Teraz nadszedł czas, aby zdefiniować pochodnej kontekst, który reprezentuje sesję z bazą danych, dzięki czemu nam zapytania i Zapisz dane. Definiujemy kontekstu, który pochodzi z System.Data.Entity.DbContext i udostępnia wpisane DbSet< TEntity > dla każdej klasy w naszym modelu.

Zaczynamy teraz użyć typów z programu Entity Framework, dlatego musimy Dodaj pakiet NuGet platformy EntityFramework.

- **Project —> Zarządzaj pakietami NuGet...** Uwaga: Jeśli nie masz **Zarządzaj pakietami NuGet...** opcja, należy zainstalować [najnowszej wersji pakietu NuGet](#)
- Wybierz **Online** kartę
- Wybierz **EntityFramework** pakietu
- Kliknij przycisk **instalacji**

Dodaj instrukcję using poufności informacji dotyczące System.Data.Entity w górnej części pliku Program.cs.

```
using System.Data.Entity;
```

Poniższe klasy wpisu w pliku Program.cs Dodaj następującym kontekstem pochodnych.

```

public class BloggingContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }
    public DbSet<Post> Posts { get; set; }
}

```

Oto Pełna lista Program.cs powinien teraz zawartość.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Data.Entity;

namespace CodeFirstNewDatabaseSample
{
    class Program
    {
        static void Main(string[] args)
        {
        }
    }

    public class Blog
    {
        public int BlogId { get; set; }
        public string Name { get; set; }

        public virtual List<Post> Posts { get; set; }
    }

    public class Post
    {
        public int PostId { get; set; }
        public string Title { get; set; }
        public string Content { get; set; }

        public int BlogId { get; set; }
        public virtual Blog Blog { get; set; }
    }

    public class BloggingContext : DbContext
    {
        public DbSet<Blog> Blogs { get; set; }
        public DbSet<Post> Posts { get; set; }
    }
}
```

To już cały kod, której potrzebujesz do rozpoczęcia przechowywania i pobierania danych. Oczywiście istnieje znacznej dzieje się w tle, a firma Microsoft będzie Przyjrzyj się, w momencie, ale pierwszym umożliwia sprawdzenie w działaniu.

4. Odczytywanie i zapisywanie danych

Implementuje metody Main w pliku Program.cs, jak pokazano poniżej. Ten kod tworzy nowe wystąpienie nasz kontekst i używa go do wstawiania nowego bloga. Następnie używa zapytania LINQ, aby pobrać wszystkie blogi z bazy danych uporządkowana w kolejności alfabetycznej według tytułu.

```

class Program
{
    static void Main(string[] args)
    {
        using (var db = new BloggingContext())
        {
            // Create and save a new Blog
            Console.Write("Enter a name for a new Blog: ");
            var name = Console.ReadLine();

            var blog = new Blog { Name = name };
            db.Blogs.Add(blog);
            db.SaveChanges();

            // Display all Blogs from the database
            var query = from b in db.Blogs
                        orderby b.Name
                        select b;

            Console.WriteLine("All blogs in the database:");
            foreach (var item in query)
            {
                Console.WriteLine(item.Name);
            }

            Console.WriteLine("Press any key to exit...");
            Console.ReadKey();
        }
    }
}

```

Możesz teraz uruchomić aplikację i ją przetestować.

```

Enter a name for a new Blog: ADO.NET Blog
All blogs in the database:
ADO.NET Blog
Press any key to exit...

```

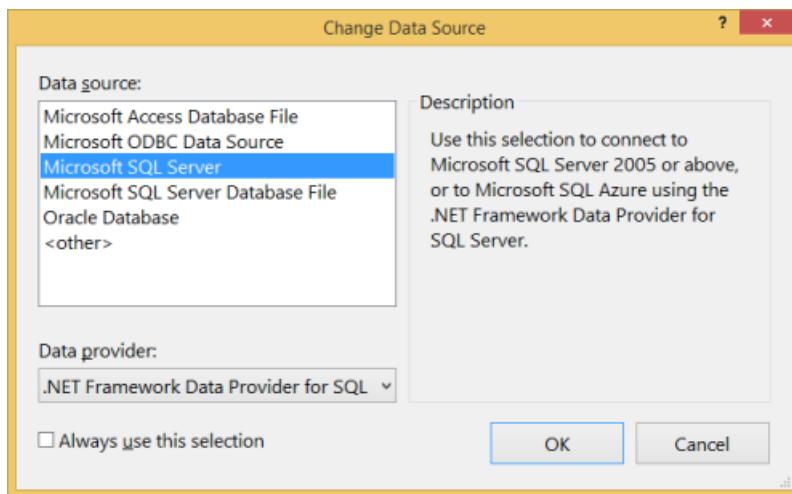
Gdzie są moje dane?

Zgodnie z Konwencją DbContext utworzył bazę danych dla Ciebie.

- Jeśli lokalne wystąpienie programu SQL Express jest dostępny (instalowany domyślnie w programie Visual Studio 2010) następnie Code First została utworzona baza danych w tym wystąpieniu
- Jeśli program SQL Express nie jest dostępny, a następnie Code First będzie próbują i użyć [LocalDB](#) (instalowany domyślnie w programie Visual Studio 2012)
- Baza danych jest nazwana w pełni kwalifikowaną nazwą pochodnego kontekst, w tym przypadku, który jest [**CodeFirstNewDatabaseSample.BloggingContext**](#)

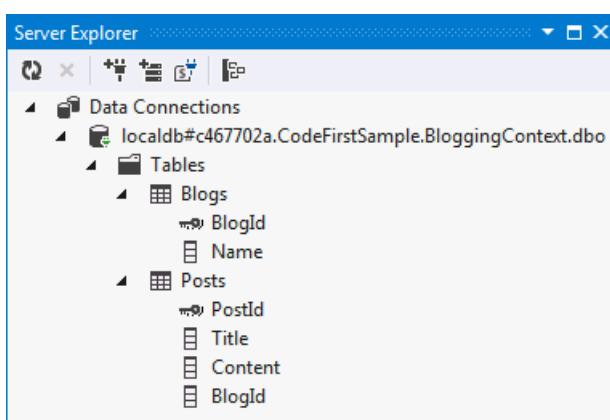
Są to po prostu domyślnych Konwencji istnieją różne sposoby, aby zmienić bazę danych, która używa Code First, więcej informacji znajduje się w **jak DbContext wykrywa Model i połączenie z bazą danych** tematu. Możesz nawiązać połączenie tej bazy danych za pomocą Eksploratora serwera w programie Visual Studio

- **Widok —> Eksploratora serwera**
- Kliknij prawym przyciskiem myszy **połączeń danych** i wybierz **Dodaj połączenie...**
- Jeśli nie jest połączona z bazą danych za pomocą Eksploratora serwera przed, musisz wybrać programu Microsoft SQL Server jako źródło danych



- Łączenie się z LocalDB lub SQL Express, w zależności od tego, który z nich został zainstalowany

Firma Microsoft jest teraz Zbadaj schematu utworzonego Code First.



Które klasy do uwzględnienia w modelu, analizując właściwości DbSet, które zdefiniowaliśmy opracowaną w DbContext. Następnie używa domyślnego zestawu Code First konwencje do określenia nazwy tabel i kolumn, określanie typów danych, znajdowanie, kluczy podstawowych, itp. W dalszej części tego przewodnika, omówimy sposób można zastąpić tych konwencji.

5. Obsługa zmiany modelu

Teraz nadszedł czas, aby wprowadzić pewne zmiany na nasz model, jeśli możemy wprowadzić te zmiany, należy również zaktualizować schemat bazy danych. W tym celu użyjemy do użycia funkcję migracje Code First ani migracji w skrócie.

Migracja pozwala mieć uporządkowany zestaw kroków, które opisują sposób uaktualnienia (i obniżenia poziomu) naszych schemat bazy danych. Każdy z tych kroków nazywana migracji, zawiera pewne kod, który opisano zmiany, które mają być stosowane.

Pierwszym krokiem jest włączenie migracje Code First dla naszych BloggingContext.

- Narzędzia —> Menedżer pakietów biblioteki -> Konsola Menedżera pakietów**
- Uruchom **Enable-Migrations** polecenia w konsoli Menedżera pakietów
- Nowy folder migracji został dodany do naszych projektu, który zawiera dwa elementy:
 - Configuration.cs** — ten plik zawiera ustawienia, używających migracje migracji BloggingContext. Nie trzeba zmieniać niczego w ramach tego przewodnika, ale poniżej przedstawiono, w którym można określić przestrzeń nazw zmieniają się dane inicjatora, dostawcy rejestru dla innych baz danych, czy migracje są generowane w itp.

- <**Sygnatura czasowa**>_InitialCreate.cs — jest to pierwszy migracji, reprezentuje zmiany, które zostały już zastosowane do bazy danych, zrób to przed pustej bazą danych na taki, który zawiera tabele blogów i wpisów . Mimo że umożliwialiśmy Code First automatycznie utworzyć te tabele, skoro mamy wybranych do migracji, które zostały przekonwertowane do migracji. Kod najpierw również zarejestrował w naszej bazie danych lokalnych, czy migracja została już zainstalowana. Sygnatury czasowej na nazwę pliku służy do ustalania kolejności celów.

Teraz możemy wprowadzić zmiany w naszym modelu, Dodaj właściwość adres Url do klasy Blog:

```
public class Blog
{
    public int BlogId { get; set; }
    public string Name { get; set; }
    public string Url { get; set; }

    public virtual List<Post> Posts { get; set; }
}
```

- Uruchom **AddUrl migracji Dodaj** polecenia w konsoli Menedżera pakietów. Polecenie Dodaj migracji sprawdza zmiany od ostatniego migracji i scaffolds nowej migracji ze wszystkimi zmianami, które znajdują się. Firma Microsoft może nazwij migracje; w takim przypadku wywołania migracji "AddUrl". Utworzony szkielet kodu mówi, że musimy dodać kolumnę adres Url, który może pomieścić ciąg danych, do dbo. Blogi tabeli. Jeśli to konieczne, można będziemy edytować utworzony szkielet kodu, ale nie jest to wymagane w tym przypadku.

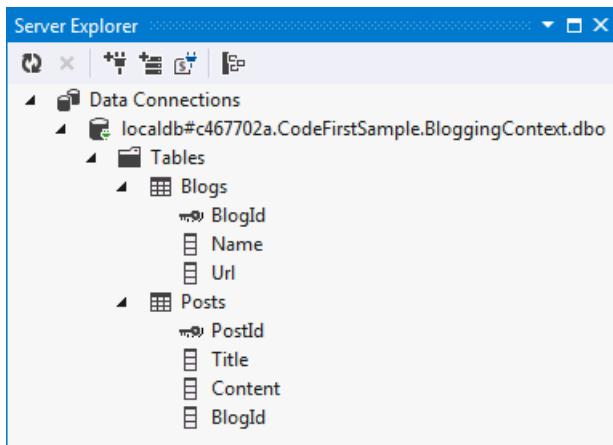
```
namespace CodeFirstNewDatabaseSample.Migrations
{
    using System;
    using System.Data.Entity.Migrations;

    public partial class AddUrl : DbMigration
    {
        public override void Up()
        {
            AddColumn("dbo.Blogs", "Url", c => c.String());
        }

        public override void Down()
        {
            DropColumn("dbo.Blogs", "Url");
        }
    }
}
```

- Uruchom **Update-Database** polecenia w konsoli Menedżera pakietów. To polecenie będzie dotyczyć wszelkie oczekujące migracji bazy danych. Nasze migracji InitialCreate została już zainstalowana, migracje będą po prostu zastosować naszej nowej migracji AddUrl. Porada: Możesz użyć — **pełne** Przełącz podczas wywoływania Update-Database, aby zobaczyć, SQL Server, który jest wykonywana w bazie danych.

Nowa kolumna adres Url jest dodany do blogów tabeli w bazie danych:



6. Adnotacje danych

Do tej pory zostało właśnie umożliwialiśmy EF odnajdywanie modelu przy użyciu jego domyślnych Konwencji, ale będą występować sytuacje, gdy nie są zgodne z konwencjami naszych zajęć i musimy wykonać dalszą część konfigurowania. Dostępne są dwie opcje dla tego; omówimy adnotacje danych w tej sekcji, a następnie wygodnego interfejsu API w następnej sekcji.

- Dodajmy klasy użytkownika w naszym modelu

```
public class User
{
    public string Username { get; set; }
    public string DisplayName { get; set; }
}
```

- Należy również dodać zestaw do nasz kontekst pochodne

```
public class BloggingContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }
    public DbSet<Post> Posts { get; set; }
    public DbSet<User> Users { get; set; }
}
```

- Jeśli firma Microsoft podjęto próbę dodania migracji otrzymamy wynik powiedzenie błęd "obiektu *EntityType*"User"nie ma zdefiniowanego klucza. Klucz jest definiowany dla tego obiektu *EntityType*." ponieważ EF nie ma możliwości, wiedząc, że nazwa użytkownika powinna mieć klucz podstawowy dla użytkownika.
- Zamierzamy korzystanie z adnotacji danych teraz musimy dodać, przy użyciu instrukcji w górnej części pliku Program.cs

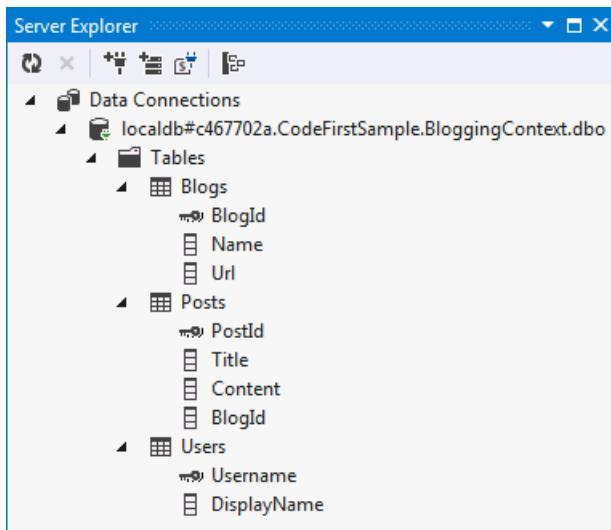
```
using System.ComponentModel.DataAnnotations;
```

- Teraz dodawać adnotacje do właściwości nazwy użytkownika do identyfikowania, czy jest klucz podstawowy

```
public class User
{
    [Key]
    public string Username { get; set; }
    public string DisplayName { get; set; }
}
```

- Użyj **AddUser migracji** Dodaj polecenia do tworzenia szkieletu migracji, aby zastosować te zmiany do bazy danych
- Uruchom **Update-Database** polecenie, aby zastosować nową migrację do bazy danych

Teraz zostanie dodana nowa tabela, do bazy danych:



Pełna lista adnotacje obsługiwane przez EF jest:

- [KeyAttribute](#)
- [StringLengthAttribute](#)
- [MaxLengthAttribute](#)
- [ConcurrencyCheckAttribute](#)
- [RequiredAttribute](#)
- [TimestampAttribute](#)
- [ComplexTypeAttribute](#)
- [ColumnAttribute](#)
- [TableAttribute](#)
- [InversePropertyAttribute](#)
- [ForeignKeyAttribute](#)
- [DatabaseGeneratedAttribute](#)
- [NotMappedAttribute](#)

7. Interfejs Fluent API

W poprzedniej sekcji przyjrzaliśmy się przy użyciu adnotacji danych do uzupełnienia lub zastąpić, co zostało wykryte przez Konwencję. Inny sposób, aby skonfigurować model jest za pośrednictwem interfejsu API fluent Code First.

Większość konfiguracji modelu można zrobić za pomocą prostych danych adnotacji. Interfejs fluent API jest bardziej zaawansowany sposób określania Konfiguracja modelu, który obejmuje wszystko, co dodatkowo zrobić adnotacje danych na niektórych bardziej zaawansowanych konfiguracji nie jest możliwe przy użyciu adnotacji danych. Adnotacje danych i wygodnego interfejsu API mogą być używane razem.

Dostęp do interfejsu API fluent zastąpienia metody OnModelCreating w DbContext. Założymy, że Chcieliśmy, aby zmienić nazwę kolumny, która User.DisplayName znajduje się w celu wyświetlenia_nazwy.

- Zastąp metodę OnModelCreating na BloggingContext następującym kodem

```

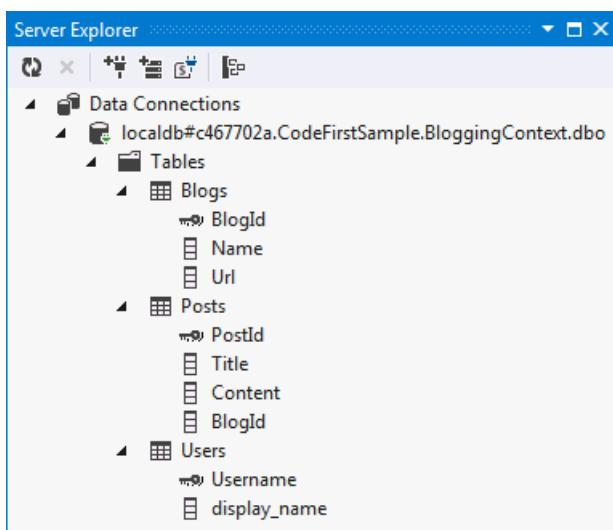
public class BloggingContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }
    public DbSet<Post> Posts { get; set; }
    public DbSet<User> Users { get; set; }

    protected override void OnModelCreating(DbModelBuilder modelBuilder)
    {
        modelBuilder.Entity<User>()
            .Property(u => u.DisplayName)
            .HasColumnName("display_name");
    }
}

```

- Użyj **ChangeDisplayName migracji Dodaj** polecenia do tworzenia szkieletu migracji, aby zastosować te zmiany do bazy danych.
- Uruchom **Update-Database** polecenie, aby zastosować nową migrację do bazy danych.

W kolumnie Nazwa wyświetlna została zmieniona na wyświetlanie_nazwy:



Podsumowanie

W tym przewodniku przyjrzaliśmy się rozwiązań deweloperskiego Code First przy użyciu nowej bazy danych. Firma Microsoft zdefiniowany model przy użyciu klasy, a następnie ten model umożliwia tworzenie bazy danych i przechowywać i pobierać dane. Gdy baza danych została utworzona użyliśmy migracje Code First można zmienić schematu jako nasz model ewoluował. Widzieliśmy również sposób konfigurowania model przy użyciu adnotacji danych i interfejsu API Fluent.

Najpierw kod istniejącej bazy danych

13.09.2018 • 9 minutes to read • [Edit Online](#)

W tym przewodniku krok po kroku i wideo zawierają wprowadzenie do rozwiązania deweloperskiego Code First przeznaczonych dla istniejącej bazy danych. Kod umożliwia najpierw zdefiniować model za pomocą języka C# lub klasy VB.Net. Opcjonalnie dodatkowej konfiguracji można wykonać przy użyciu atrybutów klas i właściwości lub za pomocą interfejsu API fluent.

Obejrzyj wideo

To wideo jest [teraz dostępna w witrynie Channel 9](#).

Wymagania wstępne

Musisz mieć **programu Visual Studio 2012** lub **programu Visual Studio 2013** zainstalowany w celu przeprowadzenia tego instruktażu.

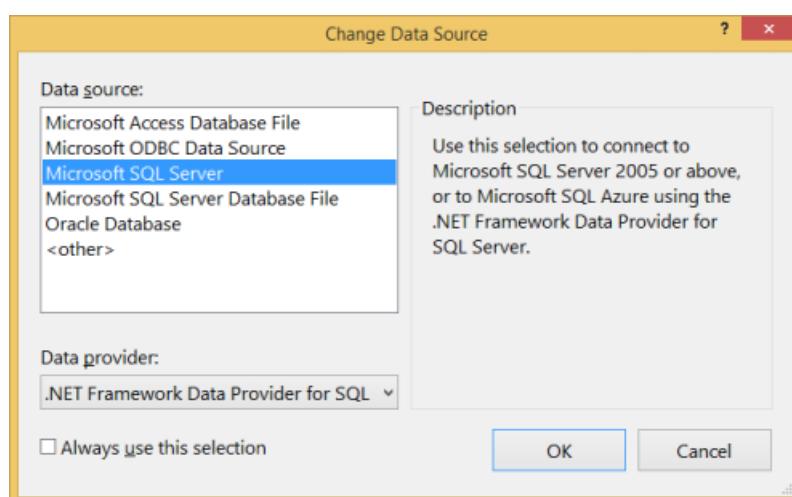
Należy również wersja **6.1** (lub nowszym) z **Entity Framework Tools for Visual Studio** zainstalowane. Zobacz [uzyskać Entity Framework](#) informacji na temat instalowania najnowszej wersji narzędzi Entity Framework Tools.

1. Utwórz istniejącej bazy danych

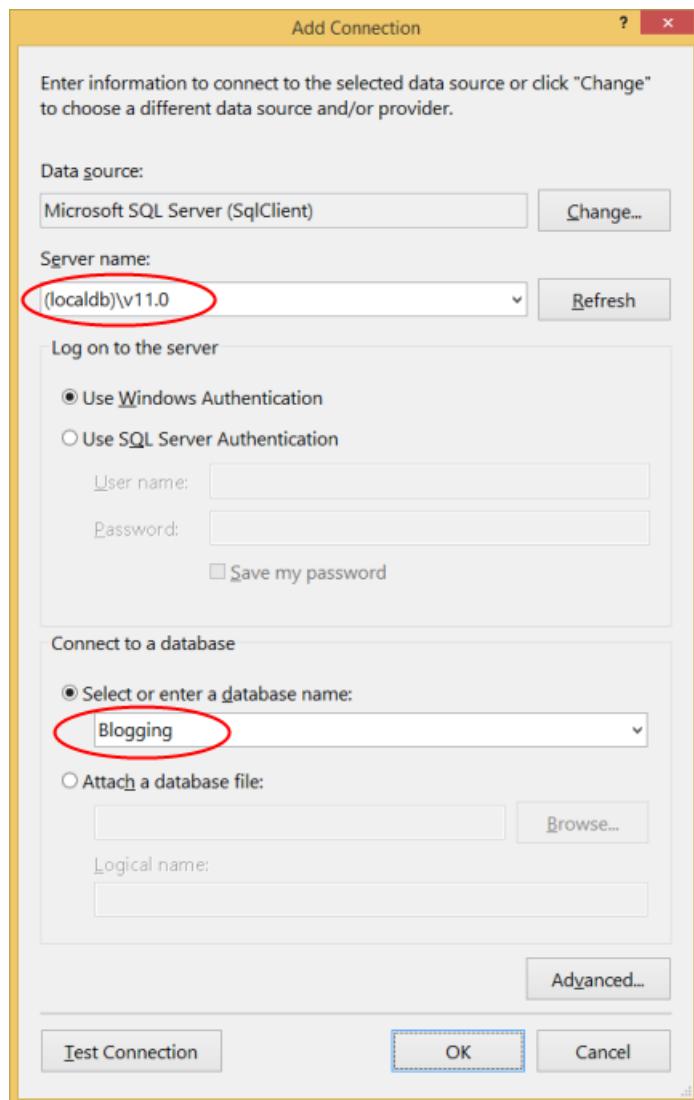
Zazwyczaj przeznaczonych do istniejącej bazy danych, zostanie już utworzony, ale w tym przewodniku należy utworzyć bazy danych w celu uzyskania dostępu.

Rozpoczniemy i wygenerować bazę danych.

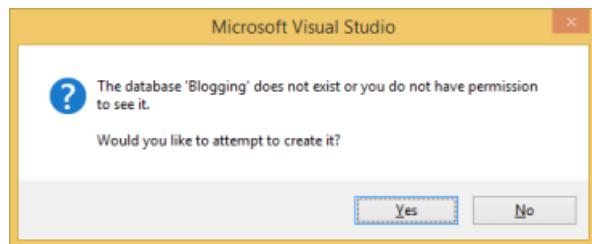
- Otwórz program Visual Studio
- **Widok —> Eksploratora serwera**
- Kliknij prawym przyciskiem myszy **połączeń danych** -> **Dodaj połaczenie...**
- Jeśli nie jest połączona z bazą danych z **Eksploratora serwera** zanim będzie konieczne wybranie **programu Microsoft SQL Server** jako źródło danych



- Połącz się z wystąpieniem usługi LocalDB, a następnie wprowadź **do obsługi blogów** jako nazwa bazy danych



- Wybierz **OK** i uzyskasz, jeśli chcesz utworzyć nową bazę danych, wybierz opcję **tak**



- Nowe bazy danych będą teraz wyświetlane w Eksploratorze serwera, kliknij prawym przyciskiem myszy na nim i wybierz **nowe zapytanie**
- Skopiuj następujące instrukcje SQL do nowego zapytania, a następnie kliknij prawym przyciskiem myszy, zapytania i wybierz pozycję **wykonania**

```

CREATE TABLE [dbo].[Blogs] (
    [BlogId] INT IDENTITY (1, 1) NOT NULL,
    [Name] NVARCHAR (200) NULL,
    [Url] NVARCHAR (200) NULL,
    CONSTRAINT [PK_dbo.Blogs] PRIMARY KEY CLUSTERED ([BlogId] ASC)
);

CREATE TABLE [dbo].[Posts] (
    [PostId] INT IDENTITY (1, 1) NOT NULL,
    [Title] NVARCHAR (200) NULL,
    [Content] NTEXT NULL,
    [BlogId] INT NOT NULL,
    CONSTRAINT [PK_dbo.Posts] PRIMARY KEY CLUSTERED ([PostId] ASC),
    CONSTRAINT [FK_dbo.Posts_dbo.Blogs_BlogId] FOREIGN KEY ([BlogId]) REFERENCES [dbo].[Blogs] ([BlogId]) ON
DELETE CASCADE
);

INSERT INTO [dbo].[Blogs] ([Name],[Url])
VALUES ('The Visual Studio Blog', 'http://blogs.msdn.com/visualstudio/')

INSERT INTO [dbo].[Blogs] ([Name],[Url])
VALUES ('.NET Framework Blog', 'http://blogs.msdn.com/dotnet/')

```

2. Tworzenie aplikacji

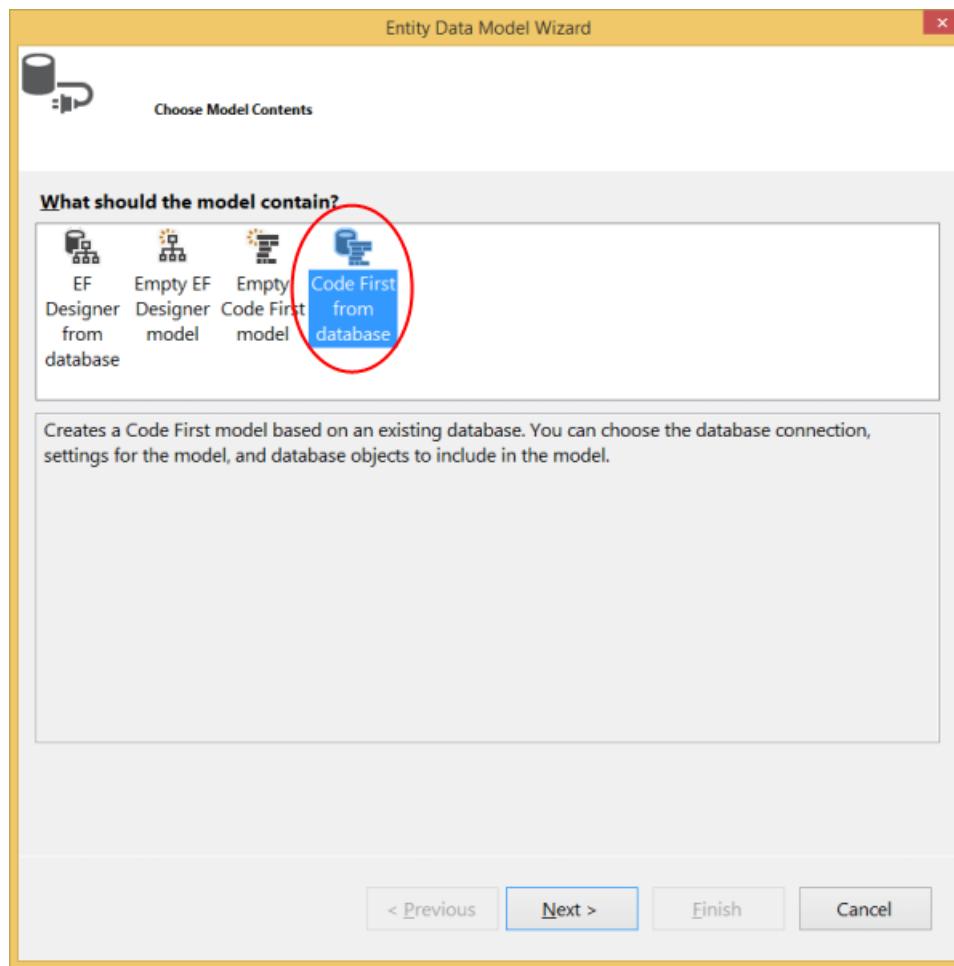
Aby zachować ich prostotę zamierzamy utworzyć aplikację podstawowa konsola, która używa Code First do dostępu do danych:

- Otwórz program Visual Studio
- **Plik —> New -> projektu...**
- Wybierz **Windows** menu po lewej stronie i **aplikacji konsoli**
- Wprowadź **CodeFirstExistingDatabaseSample** jako nazwę
- Wybierz **OK**

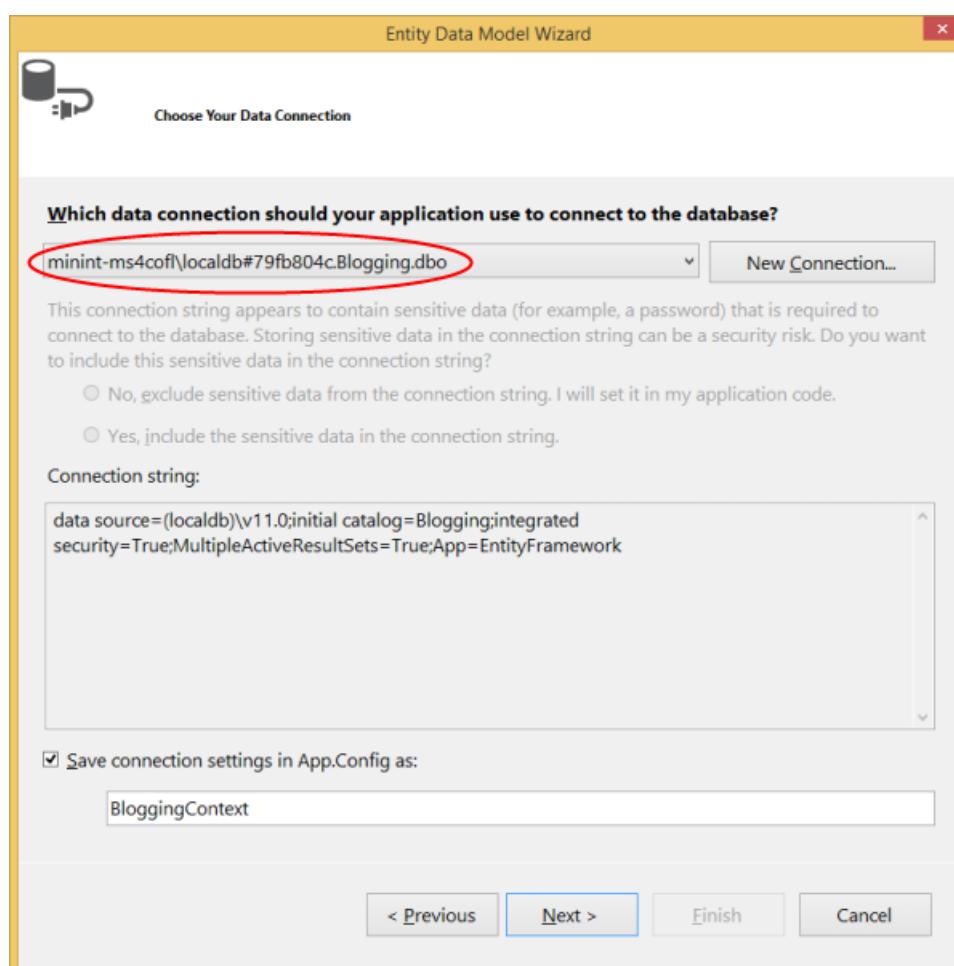
3. Model odtwarzania

Będziemy korzystać z narzędzi Entity Framework Tools for Visual Studio ułatwia nam dotarcie kodu początkowego do mapowania na bazie danych. Te narzędzia po prostu generują kod, który można także wpisać ręcznie, jeśli użytkownik sobie tego życzy.

- **Projekt —> Dodaj nowy element...**
- Wybierz **danych** menu po lewej stronie i następnie **ADO.NET Entity Data Model**
- Wprowadź **BloggingContext** jako nazwę i kliknij przycisk **OK**
- Spowoduje to uruchomienie **Kreator modelu Entity Data Model**
- Wybierz **Code First z bazy danych** i kliknij przycisk **dalej**

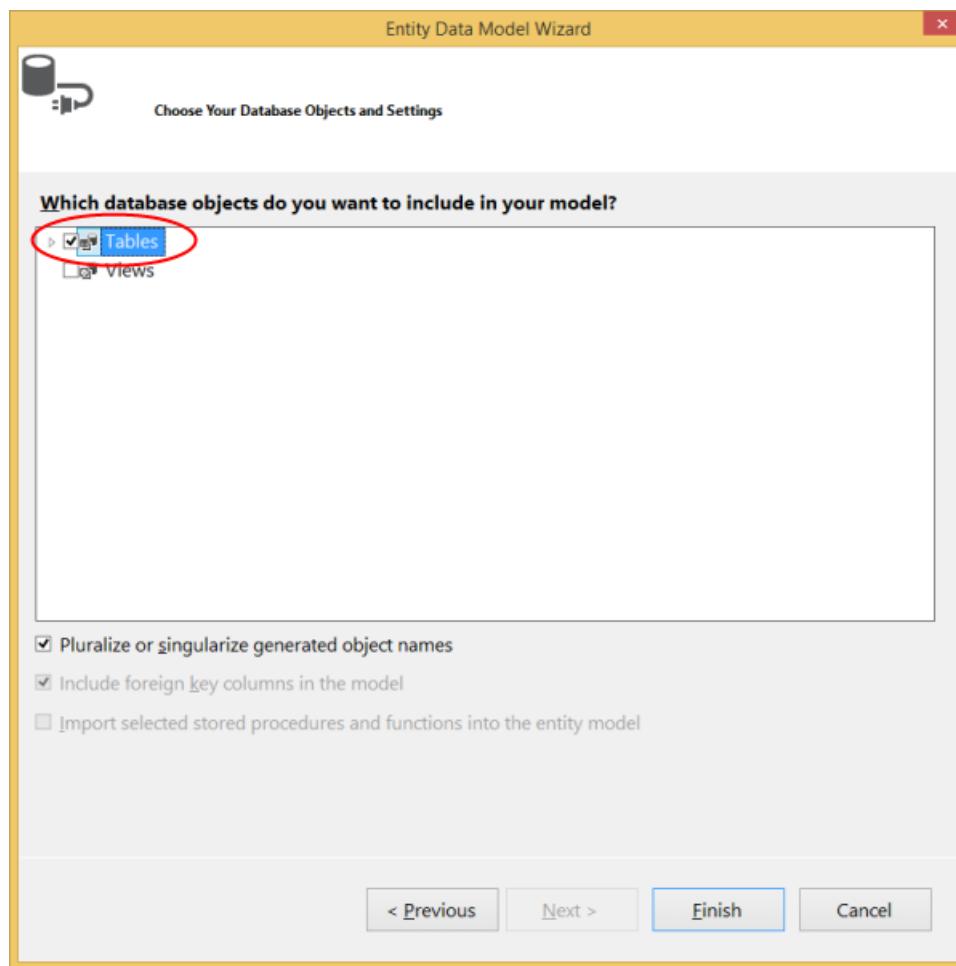


- Wybierz połączenie do bazy danych utworzonej w pierwszej sekcji, a następnie kliknij przycisk **dalej**



- Kliknij pole wyboru obok pozycji **tabel** importowania wszystkich tabel, a następnie kliknij przycisk

Zakończ



Po procesu odtwarzania wykonaniu określonej liczby elementów będzie zostały dodane do projektu, możemy przyjrzeć się elementy dodane.

Plik konfiguracji

Plik App.config został dodany do projektu, ten plik zawiera parametry połączenia z istniejącą bazą danych.

```
<connectionStrings>
  <add
    name="BloggingContext"
    connectionString="data source=(localdb)\mssqllocaldb;initial catalog=Blogging;integrated
    security=True;MultipleActiveResultSets=True;App=EntityFramework"
    providerName="System.Data.SqlClient" />
</connectionStrings>
```

Można zauważyc za niektore inne ustawienia w pliku konfiguracji, są ustawienia domyslne programu EF, które informują Code First, gdzie można utworzyć bazy danych. Ponieważ firma Microsoft jest mapowany do istniejącej bazy danych, te ustawienia zostaną zignorowane w naszej aplikacji.

Kontekst pochodne

A **BloggingContext** klasy został dodany do projektu. Kontekst reprezentuje sesję z bazą danych, dzięki czemu nam zapytania i Zapisz dane. Udostępnia kontekst **DbSet< TEntity >** dla każdego typu w naszym modelu.

Można także zauważyc, że domyślny konstruktor wywołuje konstruktora podstawowego przy użyciu **nazwa =** składni. Oznacza to Code First, że parametry połączenia do użycia dla tego kontekstu powinny być załadowane z pliku konfiguracji.

```

public partial class BloggingContext : DbContext
{
    public BloggingContext()
        : base("name=BloggingContext")
    {
    }

    public virtual DbSet<Blog> Blogs { get; set; }
    public virtual DbSet<Post> Posts { get; set; }

    protected override void OnModelCreating(DbModelBuilder modelBuilder)
    {
    }
}

```

Należy zawsze używać **nazwa** = składni, w przypadku korzystania z parametrów połączenia w pliku konfiguracji. Daje to gwarancję, że jeśli nie ma parametrów połączenia następnie Entity Framework zgłosi zamiast tworzenia nowej bazy danych przez Konwencję.

Klasy modelu

Na koniec **Blog** i **wpis** klasy również zostały dodane do projektu. Są to klasy domeny, które tworzą nasz model. Zobaczysz adnotacje danych stosowane do klas do określenia konfiguracji, której konwencje Code First nie będzie wyrównane z struktury istniejącej bazy danych. Na przykład, zobaczysz **StringLength** adnotację **Blog.Name** i **Blog.Url** ponieważ mają maksymalną długość **200** w bazie danych (Code First domyślna ma używać długość maksymalną obsługiwana przez dostawcę bazy danych — **nvarchar(max)** w programie SQL Server).

```

public partial class Blog
{
    public Blog()
    {
        Posts = new HashSet<Post>();
    }

    public int BlogId { get; set; }

    [StringLength(200)]
    public string Name { get; set; }

    [StringLength(200)]
    public string Url { get; set; }

    public virtual ICollection<Post> Posts { get; set; }
}

```

4. Odczytywanie i zapisywanie danych

Teraz, gdy model, nadszedł czas na potrzeby dostępu do niektórych danych. Implementowanie **Main** method in Class metoda **Program.cs** jak pokazano poniżej. Ten kod tworzy nowe wystąpienie klasy nasz kontekst, a następnie używa go, aby wstawić nowy **blogu**. Następnie wykorzystuje zapytania LINQ można pobrać wszystkich **blogi** z bazy danych uporządkowana w kolejności alfabetycznej według **tytuł**.

```
class Program
{
    static void Main(string[] args)
    {
        using (var db = new BloggingContext())
        {
            // Create and save a new Blog
            Console.WriteLine("Enter a name for a new Blog: ");
            var name = Console.ReadLine();

            var blog = new Blog { Name = name };
            db.Blogs.Add(blog);
            db.SaveChanges();

            // Display all Blogs from the database
            var query = from b in db.Blogs
                        orderby b.Name
                        select b;

            Console.WriteLine("All blogs in the database:");
            foreach (var item in query)
            {
                Console.WriteLine(item.Name);
            }

            Console.WriteLine("Press any key to exit...");
            Console.ReadKey();
        }
    }
}
```

Możesz teraz uruchomić aplikację i ją przetestować.

```
Enter a name for a new Blog: ADO.NET Blog
All blogs in the database:
.NET Framework Blog
ADO.NET Blog
The Visual Studio Blog
Press any key to exit...
```

Co zrobić, jeśli Moja baza danych zmiany?

Code First Kreatora bazy danych jest przeznaczona do generowania początkowego zestawu punktu klas, które można dostosować i modyfikować. Zmiany schematu bazy danych można ręcznie edytować klasy lub wykonania innego odtwarzania, aby zastąpić klasy.

Za pomocą migracje Code First istniejącą bazę danych

Jeśli chcesz użyć migracje Code First z istniejącej bazy danych, zobacz [migracje Code First istniejącą bazę danych](#).

Podsumowanie

W tym przewodniku przyjrzaliśmy się rozwiązań deweloperskiego Code First przy użyciu istniejącej bazy danych. Użyliśmy narzędzi Entity Framework Tools dla programu Visual Studio do odtworzenia zestawu klas, które są mapowane do bazy danych i może służyć do przechowywania i pobierania danych.

Adnotacje danych na pierwszym kodu

03.11.2018 • 27 minutes to read • [Edit Online](#)

NOTE

EF4.1 począwszy tylko — funkcje, interfejsów API itp. z opisem na tej stronie zostały wprowadzone w programie Entity Framework 4.1. Jeśli używasz starszej wersji, niektóre lub wszystkie z tych informacji nie ma zastosowania.

Zawartość na tej stronie są zaczerpnięte z artykułu pierwotnie napisane przez Julie Lerman (<<http://thedatafarm.com>>).

Entity Framework Code First pozwala na używanie własnych klas domeny do reprezentowania modelu, który zależy od platformy EF do wykonywania zapytań, zmień śledzenie i aktualizowanie funkcji. Kod najpierw wykorzystuje wzorzec programowania, określone jako Konwencja za pośrednictwem konfiguracji. Najpierw kod będzie założono, że Twoich zajęciach zgodne z konwencjami Entity Framework, w takim przypadku będą działać automatycznie informacje o tym, jak wykonać to zadanie. Jednak jeśli Twoich zajęciach nie wykonuj tych konwencji, masz możliwość dodawania konfiguracji do swojej klasy zapewniające EF niezbędne informacje.

Kod daje najpierw dodaj te konfiguracje do swoich klas na dwa sposoby. Jeden używa prostego atrybuty o nazwie DataAnnotations, a drugi używa Code First Fluent interfejsu API, który zapewnia sposób, aby opisać konfiguracje obowiązkowo, w kodzie.

Ten artykuł koncentruje się na konfigurowanie Twoich zajęciach — wyróżnianie najczęściej wymagane konfiguracje, za pomocą DataAnnotations (w przestrzeni nazw System.ComponentModel.DataAnnotations). DataAnnotations również są zrozumiałe przez kilka aplikacji .NET, takich jak ASP.NET MVC, która umożliwia tych aplikacji korzystać z tej samej adnotacji dla walidacji po stronie klienta.

Model

Zademonstruję DataAnnotations pierwszy kodu przy użyciu prostego pary klas: Blog i Post.

```
public class Blog
{
    public int Id { get; set; }
    public string Title { get; set; }
    public string BloggerName { get; set; }
    public virtual ICollection<Post> Posts { get; set; }
}

public class Post
{
    public int Id { get; set; }
    public string Title { get; set; }
    public DateTime DateCreated { get; set; }
    public string Content { get; set; }
    public int BlogId { get; set; }
    public ICollection<Comment> Comments { get; set; }
}
```

Jak są one klasy blogu i wpis wygodnie Konwencją pierwszy kodu i wymagają nie ulepszeń, aby włączyć zgodności EF. Jednak również umożliwia adnotacje zapewnienie EF więcej informacji na temat klas i bazy danych, które mapują.

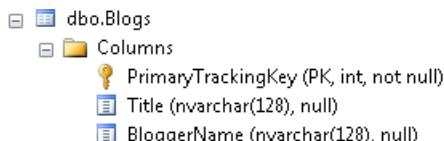
Key

Entity Framework opiera się na każdej jednostki o wartości klucza, który służy do śledzenia jednostek. Jeden Konwencji Code First jest niejawne właściwości klucza; Kod najpierw sprawdza właściwości o nazwie "Id" lub kombinacji nazwy klasy i "Id", takich jak "BlogId". Ta właściwość będzie zmapowana do kolumny klucza podstawowego w bazie danych.

Klasy blogu i wpis stosują taką Konwencję. Co zrobić, jeśli ich nie? Co zrobić, jeśli użyto nazwy w blogu *PrimaryTrackingKey* w zamian lub nawet *foo*? Jeśli kod najpierw nie może znaleźć właściwość, która pasuje do niniejszej Konwencji spowoduje zgłoszenie wyjątku, ze względu na wymagania programu Entity Framework, musi mieć właściwość klucza. Można użyć klucza adnotacji, aby określić, które właściwości, które ma być używany jako EntityKey.

```
public class Blog
{
    [Key]
    public int PrimaryTrackingKey { get; set; }
    public string Title { get; set; }
    public string BloggerName { get; set; }
    public virtual ICollection<Post> Posts { get; set; }
}
```

Jeśli najpierw przy użyciu kodu jest funkcją generowanie bazy danych, tabela blogu będzie miała kolumny klucza podstawowego o nazwie *PrimaryTrackingKey*, który również jest zdefiniowany jako tożsamość domyślnie.



```
dbo.Blogs
  Columns
    PrimaryTrackingKey (PK, int, not null)
    Title (nvarchar(128), null)
    BloggerName (nvarchar(128), null)
```

Klucze złożone

Entity Framework obsługuje kluczy złożonych — klucze podstawowe, które składają się z więcej niż jednej właściwości. Na przykład może mieć klasy usługi Passport, którego klucz podstawowy jest kombinacją *PassportNumber* i *IssuingCountry*.

```
public class Passport
{
    [Key]
    public int PassportNumber { get; set; }
    [Key]
    public string IssuingCountry { get; set; }
    public DateTime Issued { get; set; }
    public DateTime Expires { get; set; }
}
```

Podjęto próbę użycia klasy powyżej w modelu platformy EF mogłoby spowodować `InvalidOperationException`:

Nie można określić złożonego podstawowego klucza porządkowanie dla typu "Paszport". Użyj metody HasKey lub ColumnAttribute, aby określić zamówienia złożone kluczy podstawowych.

Aby można było używać kluczy złożonych, platformy Entity Framework wymaga do definiowania porządku właściwości klucza. Można to zrobić przy użyciu adnotacji kolumny do określania kolejności.

NOTE

Wartość kolejności jest względna (a nie na podstawie indeksu), dzięki czemu można używać dowolnej wartości. Na przykład 100 do 200 byłoby dopuszczalne zamiast 1 i 2.

```
public class Passport
{
    [Key]
    [Column(Order=1)]
    public int PassportNumber { get; set; }
    [Key]
    [Column(Order = 2)]
    public string IssuingCountry { get; set; }
    public DateTime Issued { get; set; }
    public DateTime Expires { get; set; }
}
```

Jeśli jednostki z kluczy obcych złożonych, należy określić w tej samej kolumnie, porządkowanie, który był używany dla odpowiednich właściwości klucza podstawowego.

Tylko względną kolejność w ramach właściwości klucza obcego muszą być takie same, dokładne wartości, które są przypisane do **kolejność** nie muszą być zgodne. Na przykład w następującej klasy 3 i 4 może służyć zamiast 1 i 2.

```
public class PassportStamp
{
    [Key]
    public int StampId { get; set; }
    public DateTime Stamped { get; set; }
    public string StampingCountry { get; set; }

    [ForeignKey("Passport")]
    [Column(Order = 1)]
    public int PassportNumber { get; set; }

    [ForeignKey("Passport")]
    [Column(Order = 2)]
    public string IssuingCountry { get; set; }

    public Passport Passport { get; set; }
}
```

Wymagane

Wymagane adnotacji informuje EF, czy określona właściwość jest wymagana.

Dodawanie wymaganych do właściwości Title wymusi EF (i MVC), aby upewnić się, że właściwość zawiera dane.

```
[Required]
public string Title { get; set; }
```

Nie dodatkowych bez zmiany kodu lub języka znaczników w aplikacji, aplikacji MVC przeprowadzi weryfikację po stronie klienta, nawet dynamiczne tworzenie komunikat przy użyciu nazwy właściwości i adnotacji.

Create

Blog

Title
 The Title field is required.

BloggerName
 Julie

Wymagany atrybut wpłynie również na wygenerowanej bazy danych, wprowadzając mapowanej właściwości niedopuszczającej. Należy zauważać, że pole tytułu została zmieniona na "nie ma wartości null".

NOTE

W niektórych przypadkach może nie być możliwe dla kolumny w bazie danych, może nie dopuszczać wartości null, nawet jeśli właściwość jest wymagana. Na przykład, gdy przy użyciu danych strategii TPH dziedziczenia dla wielu typów są przechowywane w jednej tabeli. Jeśli typ pochodny obejmuje wymagana właściwość kolumny nie można dokonać innych niż null, ponieważ nie wszystkie typy w hierarchii mają ta właściwość.

```
dbo.Blogs
  Columns
    PrimaryTrackingKey (PK, int, not null)
    Title (nvarchar(128), not null)
    BloggerName (nvarchar(128), null)
```

Element MaxLength i MinLength

Atrybuty MaxLength i MinLength umożliwiają określenie dodatkowych właściwości sprawdzania poprawności, tak jak w przypadku wymagany.

Oto BloggerName o wymagania dotyczące długości. W przykładzie pokazano również sposób łączenia atrybutów.

```
[MaxLength(10),MinLength(5)]
public string BloggerName { get; set; }
```

Adnotacja MaxLength wpłynie na bazie danych przez ustawienie właściwości długości do 10.

```
Columns
  PrimaryTrackingKey (PK, int, not null)
  Title (nvarchar(128), not null)
  BloggerName (nvarchar(10), null)
```

Adnotacja po stronie klienta w MVC i EF 4.1 po stronie serwera adnotacji zarówno podlegają weryfikacji ponownie dynamicznie tworzenie komunikat o błędzie: "pole BloggerName musi być typu string lub tablicy o maksymalnej długości"10". Ten komunikat jest nieco długi. Wiele adnotacji umożliwiają określenie komunikat o błędzie z atrybutem komunikat o błędzie.

```
[MaxLength(10, ErrorMessage="BloggerName must be 10 characters or less"),MinLength(5)]
public string BloggerName { get; set; }
```

Można również określić komunikat o błędzie w wymaganych adnotacjach.

Create

Blog

Title

BloggerName
 BloggerName must be 10 characters or less

NotMapped

Kod Konwencji pierwsze decyduje o tym, że dla każdej właściwości, który jest obsługiwany typ danych jest reprezentowana w bazie danych. Ale to nie jest zawsze w przypadku aplikacji. Na przykład może mieć właściwości w klasie blogów, która tworzy kod, w oparciu o pola tytułu i BloggerName. Tej właściwości można tworzyć dynamicznie i nie musi być przechowywany. Możesz oznaczyć wszystkie właściwości, które nie są mapowane do bazy danych z adnotacją NotMapped, np. Ta właściwość BlogCode.

```
[NotMapped]
public string BlogCode
{
    get
    {
        return Title.Substring(0, 1) + ":" + BloggerName.Substring(0, 1);
    }
}
```

ComplexType

Nie jest niczym niezwykłym opisują jednostek domeny przez zestaw klas, a następnie warstwy tych klas do opisania całą jednostkę. Może na przykład dodać klasę o nazwie BlogDetails do modelu.

```
public class BlogDetails
{
    public DateTime? DateCreated { get; set; }

    [MaxLength(250)]
    public string Description { get; set; }
}
```

Należy zauważyć, że BlogDetails nie ma żadnych typów właściwości klucza. W przypadku projektowania opartego na domenach BlogDetails nazywa się obiektu wartości. Entity Framework odnosi się do obiektów wartości jako typy złożone. Typy złożone nie mogą być śledzone własnych.

Jednak jako właściwość w klasie blogu BlogDetails będzie śledzona jako część obiektu blogu. Aby najpierw umożliwia to rozpoznawanie kodu należy oznaczyć klasę BlogDetails jako ComplexType.

```
[ComplexType]
public class BlogDetails
{
    public DateTime? DateCreated { get; set; }

    [MaxLength(250)]
    public string Description { get; set; }
}
```

Teraz można dodać właściwość w klasie blogu do reprezentowania BlogDetails na blogu.

```
public BlogDetails BlogDetail { get; set; }
```

W bazie danych w tabeli blogu będzie zawierać wszystkie właściwości blogu, w tym właściwości zawarte w jego właściwość BlogDetail. Domyślnie każdej z nich jest poprzedzone nazwą typu złożonego BlogDetail.

```
dbo.Blogs
Columns
PrimaryTrackingKey (PK, int, not null)
Title (nvarchar(128), not null)
BloggerName (nvarchar(10), null)
BlogDetail_DateCreated (datetime, null)
BlogDetail_Description (nvarchar(250), null)
```

Inny interesujący Uwaga jest mimo, że właściwość DateCreated została zdefiniowana jako nieprzyjmujące wartości daty/godziny w klasie, pole odpowiedniej bazy danych dopuszcza wartości null. Należy użyć wymaganych adnotacji, jeśli chcesz mieć wpływ na schemat bazy danych.

ConcurrencyCheck

Adnotacja ConcurrencyCheck pozwala Flagą jedną lub więcej właściwości, które ma być używany dla współbieżności sprawdzanie w bazie danych, gdy użytkownik edytuje lub usunięcie jednostki. Jeśli masz doświadczenie w pracy z projektantem EF, jest to zgodne z ustawienie właściwości przed na stałe.

Zobaczmy, jak działa ConcurrencyCheck, dodając ją do właściwości BloggerName.

```
[ConcurrencyCheck, MaxLength(10, ErrorMessage="BloggerName must be 10 characters or less"), MinLength(5)]
public string BloggerName { get; set; }
```

Po wywołaniu funkcji SaveChanges ze względu na adnotacji ConcurrencyCheck pola BloggerName oryginalna wartość tej właściwości będzie używany w aktualizacji. Polecenie będzie podejmować próby zlokalizowania prawidłowego wiersza, filtrując nie tylko na wartości klucza, ale także w oryginalnej wartości BloggerName. Poniżej przedstawiono krytycznych części polecenia UPDATE wysyłane do bazy danych, gdzie można zobaczyć, polecenie spowoduje zaktualizowanie wiersza, który ma PrimaryTrackingKey jest 1 i BloggerName "Julie", który podczas oryginalnej wartości w tym blogu został pobrany z bazy danych.

```
where ([PrimaryTrackingKey] = @4) and ([BloggerName] = @5))
@4=1,@5=N'Julie'
```

Jeśli ktoś zmienił nazwę blogger blogu w tym samym czasie, ta aktualizacja zakończy się niepowodzeniem, a otrzymasz DbUpdateConcurrencyException, który należy do obsługi.

Znacznik czasu:

Jest to bardziej powszechnie, aby używać pól rowversion lub sygnatura czasowa sprawdzania współbieżności. Jednak zamiast przy użyciu adnotacji ConcurrencyCheck, można użyć bardziej szczegółowe adnotacji sygnatury czasowej, tak długo, jak typ właściwości to tablica bajtów. Kod najpierw traktują właściwości sygnatury czasowej niż jako właściwości ConcurrencyCheck, ale jej będzie również upewnił się, że pole bazy danych, która najpierw generuje kod innych niż null. Może mieć tylko jedną właściwość sygnatury czasowej w danej klasie.

Dodawanie następującej właściwości do klasy Blog:

```
[Timestamp]  
public Byte[] TimeStamp { get; set; }
```

wyniki w kodzie, najpierw tworząc kolumnę sygnatur czasowych nie dopuszcza wartości null w tabeli bazy danych.

```
dbo.Blogs  
  Columns  
    PrimaryTrackingKey (PK, int, not null)  
    Title (nvarchar(128), not null)  
    BloggerName (nvarchar(10), null)  
    TimeStamp (timestamp, not null)  
    BlogDetail_DateCreated (datetime, null)  
    BlogDetail_Description (nvarchar(250), null)
```

Tabel i kolumn

Pozwalasz Code First utworzyć bazę danych, można zmienić nazwy tabel i kolumn, które, która zostanie utworzona. Umożliwia także Code First przy użyciu istniejącej bazy danych. Ale nie zawsze jest przypadek, że nazwy klas i właściwości w domenie pasują do nazw tabel i kolumn w bazie danych.

Moje klasy ma nazwę blogu i zgodnie z Konwencją kod najpierw przyjęto założenie, że to będzie zmapowana do tabeli o nazwie blogów. Jeśli tak nie jest atrybutem tabeli można określić nazwę tabeli. Tutaj na przykład adnotacja jest określającą, czy nazwa tabeli jest InternalBlogs.

```
[Table("InternalBlogs")]  
public class Blog
```

Adnotacja kolumny jest więcej doświadczeniem podczas określania atrybutów zamapowanych kolumn. Można zaznaczyć nazwę, typ danych lub nawet kolejność, w której kolumna pojawia się w tabeli. Oto przykład atrybutu kolumny.

```
[Column("BlogDescription", TypeName="ntext")]  
public String Description {get;set;}
```

Nie należy mylić w kolumnie Nazwa atrybutu o DataAnnotation typu danych. Typ danych jest adnotacja używane dla interfejsu użytkownika i jest ignorowana przez rozwiązanie Code First.

Oto tabeli po jest zostały ponownie wygenerowane. Nazwa tabeli została zmieniony na InternalBlogs i opis kolumny z typu złożonego jest teraz BlogDescription. Ponieważ nazwa została określona w adnotacji, kod najpierw nie będzie używać konwencji początkowych nazwa kolumny o nazwie typu złożonego.

└─	└─	dbo.InternalBlogs
└─	└─	Columns
└─	└─	PrimaryTrackingKey (PK, int, not null)
└─	└─	Title (nvarchar(128), not null)
└─	└─	BloggerName (nvarchar(10), null)
└─	└─	TimeStamp (timestamp, not null)
└─	└─	BlogDetail_DateCreated (datetime, null)
└─	└─	BlogDescription (ntext, null)

DatabaseGenerated

Funkcje bazy danych ważna jest możliwość mająć być obliczane właściwości. Jeśli masz mapowania klas Code First tabel zawierających kolumny obliczane, nie ma programu Entity Framework, aby spróbować zaktualizować te kolumny. Jednak EF w celu uzyskania tych wartości z bazy danych, po zostały wstawione lub zaktualizowane dane. Do tych właściwości w klasie wraz z wyliczenia obliczane, można użyć adnotacji DatabaseGenerated. Inne typy wyliczeniowe to: Brak i tożsamości.

```
[DatabaseGenerated(DatabaseGeneratedOption.Computed)]
public DateTime DateCreated { get; set; }
```

Można użyć bazy danych, generowany dla kolumn bajtów lub sygnatura czasowa, gdy kod najpierw generuje bazy danych, w przeciwnym razie należy używać tylko to po wskazaniu istniejących baz danych, ponieważ kod najpierw nie będzie można określić formułę dla kolumny obliczanej.

Już wspomniano powyżej, domyślnie, właściwość klucza, która jest liczbą całkowitą staną się klucza tożsamości w bazie danych. Który będzie taka sama jak ustawienie DatabaseGenerated DatabaseGeneratedOption.Identity. Jeśli nie chcesz go jako klucza tożsamości, można ustawić wartości do DatabaseGeneratedOption.None.

Indeks

NOTE

EF6.1 począwszy tylko - atrybutu indeksu została wprowadzona w programie Entity Framework 6.1. Informacje przedstawione w tej sekcji nie ma zastosowania, jeśli używasz starszej wersji.

Można utworzyć indeksu w co najmniej jedną kolumnę, za pomocą **IndexAttribute**. Dodawanie atrybutu do jednej lub więcej właściwości będzie powodować EF utworzyć odpowiedni indeks w bazie danych, podczas tworzenia bazy danych lub tworzenia szkieletu odpowiednich **CreateIndex** wywołuje się, jeśli używasz migracje Code First.

Na przykład, poniższy kod będzie skutkować indeksu tworzona **ocena** kolumny **wpisy** tabeli w bazie danych.

```
public class Post
{
    public int Id { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }
    [Index]
    public int Rating { get; set; }
    public int BlogId { get; set; }
}
```

Domyślnie, będą miały nazwę nadaną indeks **IX_<nazwa właściwości>** (IX_klasyfikacji w powyższym

przykładzie). Mimo że można określić również nazwę indeksu. W poniższym przykładzie określono indeks powinien zostać nazwany **PostRatingIndex**.

```
[Index("PostRatingIndex")]
public int Rating { get; set; }
```

Indeksy są domyślnie nie jest unikatowa, ale można użyć **IsUnique** o nazwie parametru, aby określić, że indeks powinien być unikatowy. Poniższy przykład przedstawia unikatowego indeksu na **użytkownika** przez nazwę logowania.

```
public class User
{
    public int UserId { get; set; }

    [Index(IsUnique = true)]
    [StringLength(200)]
    public string Username { get; set; }

    public string DisplayName { get; set; }
}
```

Indeksy wielu kolumn

Indeksy, obejmujące wiele kolumn są określone przy użyciu tej samej nazwie w wielu adnotacjach indeksu dla danej tabeli. Podczas tworzenia indeksów wielokolumnowych, należy określić kolejność kolumn w indeksie. Na przykład, poniższy kod tworzy indeks wielokolumnowy na **ocena** i **BlogId** o nazwie **IX_BlogAndRating**.

BlogId jest w pierwszej kolumnie indeksu i **ocena** drugi.

```
public class Post
{
    public int Id { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }
    [Index("IX_BlogIdAndRating", 2)]
    public int Rating { get; set; }
    [Index("IX_BlogIdAndRating", 1)]
    public int BlogId { get; set; }
}
```

Relacji atrybutów: InverseProperty i klucza obcego

NOTE

Ta strona zawiera informacje o konfigurowaniu relacji w modelu Code First przy użyciu adnotacji danych. Aby uzyskać ogólne informacje dotyczące relacji w EF i uzyskiwania dostępu i manipulowania danymi za pomocą relacji, zobacz [relacje & właściwości nawigacji](#). *

Kod Konwencji pierwsze zajmie się najbardziej typowe relacje w modelu, ale istnieją przypadki, gdzie potrzebuje pomocy.

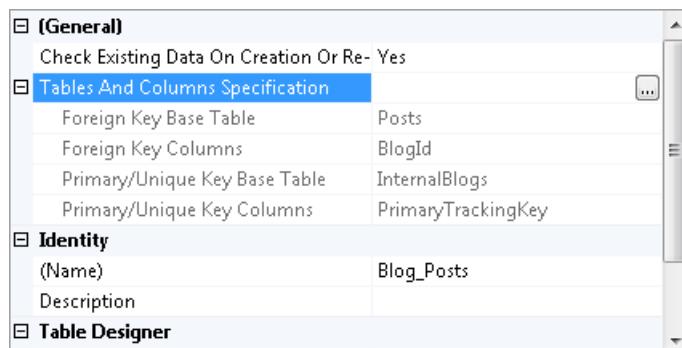
Zmiana nazwy właściwości klucza klasy blogu utworzone problem z jego relacji do wpisu.

Podczas generowania bazy danych, kod najpierw widzi właściwość **BlogId** w klasie **Post** i rozpoznaje je, zgodnie z Konwencją, czy jest on zgodny Nazwa klasy oraz "Id" jako klucza obcego do klasy blogu. Ale nie ma właściwości **BlogId** w klasie blogu. Rozwiązania dla tego jest utworzenie właściwości nawigacji we wpisie i użyj

DataAnnotation obcego, aby najpierw zrozumieć sposób tworzenia relacji między dwoma klasami kodu — przy użyciu właściwości Post.BlogId — oraz jak określić ograniczeń, Baza danych.

```
public class Post
{
    public int Id { get; set; }
    public string Title { get; set; }
    public DateTime DateCreated { get; set; }
    public string Content { get; set; }
    public int BlogId { get; set; }
    [ForeignKey("BlogId")]
    public Blog Blog { get; set; }
    public ICollection<Comment> Comments { get; set; }
}
```

Ograniczenia w bazie danych przedstawiono relację między InternalBlogs.PrimaryTrackingKey i Posts.BlogId.



InverseProperty jest używany, jeśli masz wiele relacji między klasami.

W klasie wpis może być do śledzenia określonego autora wpisu w blogu oraz która ją edytowała. Poniżej przedstawiono dwie nowe właściwości nawigacji dla klasy wpisu.

```
public Person CreatedBy { get; set; }
public Person UpdatedBy { get; set; }
```

Należy również dodać w klasie osoby odwoływać się tych właściwości. Klasa osoba ma właściwości nawigacji Wstecz do wpisu, jeden dla wszystkich wpisów, napisane przez osoby i jeden dla wszystkich wpisów, aktualizowane przez tę osobę.

```
public class Person
{
    public int Id { get; set; }
    public string Name { get; set; }
    public List<Post> PostsWritten { get; set; }
    public List<Post> PostsUpdated { get; set; }
}
```

Kod najpierw jest możliwość dopasowania właściwości do dwóch klas samodzielnie. W tabeli bazy danych dla wpisów powinien mieć jeden klucz obcy dla osoby CreatedBy i jeden dla UpdatedBy osoby, ale kod najpierw utworzy cztery będą właściwości klucza obcego: osoba_identyfikator, osoba_Id1, CreatedBy_identyfikator i UpdatedBy_identyfikatora.

			dbo.Posts
			Columns
			Id (PK, int, not null)
			Title (nvarchar(128), null)
			DateCreated (datetime, not null)
			Content (nvarchar(128), null)
			BlogId (FK, int, not null)
			Person_Id (FK, int, null)
			Person_Id1 (FK, int, null)
			CreatedBy_Id (FK, int, null)
			UpdatedBy_Id (FK, int, null)

Aby rozwiązać te problemy, można użyć adnotacji `InverseProperty`, aby określić wyrównanie właściwości.

```
[InverseProperty("CreatedBy")]
public List<Post> PostsWritten { get; set; }

[InverseProperty("UpdatedBy")]
public List<Post> PostsUpdated { get; set; }
```

Ponieważ właściwość `PostsWritten` osobiście wie, to odnosi się do typu wpisu, utworzy relacji z elementem `Post.CreatedBy`. Podobnie `Post.UpdatedBy` połączy `PostsUpdated`. I kod najpierw nie utworzy ich klucze obce dodatkowych.

		dbo.Posts	
			Columns
			Id (PK, int, not null)
			Title (nvarchar(128), null)
			DateCreated (datetime, not null)
			Content (nvarchar(128), null)
			BlogId (FK, int, not null)
			CreatedBy_Id (FK, int, null)
			UpdatedBy_Id (FK, int, null)

Podsumowanie

DataAnnotations nie tylko umożliwiają opisującą weryfikacji po stronie klienta i serwera, w Twoich zajęciach pierwszy kodu, ale również umożliwiają rozszerzanie i nawet rozwiązać założeń, które kod najpierw spowoduje, że Twoje klasy oparte na konwencjach jej informacje. Za pomocą DataAnnotations można nie tylko zachęcić generowanie schematu bazy danych, ale można również mapować Twoich zajęciach pierwszy kodu do wcześniejszej istniejącej bazy danych.

Gdy są bardzo elastyczne, należy pamiętać, zapewniające DataAnnotations tylko najbardziej najczęściej potrzebne zmiany konfiguracji wprowadzone w Twoich zajęciach pierwszy kod. Aby skonfigurować Twoje klasy dla niektórych przypadków brzegowych, powinien wyglądać mechanizmu alternatywną konfigurację, Code First API Fluent.

Definiowanie DbSets

13.09.2018 • 2 minutes to read • [Edit Online](#)

Podczas programowania z użyciem kodu pierwszego przepływu pracy należy zdefiniować pochodnego typu DbContext, który reprezentuje sesję z bazą danych i udostępnia DbSet dla każdego typu w modelu. W tym temacie omówiono różne sposoby, można zdefiniować właściwości DbSet.

Kontekst DbContext z właściwościami DbSet

Często spotykana pokazano w przykładach Code First jest typu DbContext za pomocą właściwości publiczne automatyczne DbSet typów jednostek w modelu. Na przykład:

```
public class BloggingContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }
    public DbSet<Post> Posts { get; set; }
}
```

W przypadku użycia w trybie Code First, to zostanie skonfigurowany jako typy jednostek, jak również konfigurowanie innych typów, które są osiągalne z tych blogów i wpisów. Ponadto DbContext będzie automatycznie wywoływać metody ustawiającej dla każdej z tych właściwości można ustawić wystąpienie DbSet odpowiednie.

Kontekst DbContext z właściwościami IDbSet

Istnieją sytuacje, na przykład w przypadku tworzenia mocks lub elementów sztucznych, gdzie jest bardziej użyteczny zadeklarować właściwości zestawu przy użyciu interfejsu. W takich przypadkach IDbSet interfejs może służyć zamiast DbSet. Na przykład:

```
public class BloggingContext : DbContext
{
    public IDbSet<Blog> Blogs { get; set; }
    public IDbSet<Post> Posts { get; set; }
}
```

Ten kontekst działa w taki sam sposób, jak kontekst, który używa klasy DbSet dla jego właściwości zestawu.

Kontekst DbContext za pomocą właściwości tylko do odczytu zestawu

Jeśli nie chcesz ujawniać publicznej metody ustawiające dla właściwości DbSet lub IDbSet zamiast tego można utworzyć właściwości tylko do odczytu i samodzielnie utworzyć wystąpień zestawu. Na przykład:

```
public class BloggingContext : DbContext
{
    public DbSet<Blog> Blogs
    {
        get { return Set<Blog>(); }
    }

    public DbSet<Post> Posts
    {
        get { return Set<Post>(); }
    }
}
```

Należy pamiętać, że DbContext buforuje wystąpienie DbSet zwrócony z metody Set, tak, aby każda z tych właściwości zwróciło samo wystąpienie, za każdym razem, gdy jest wywoływana.

Odnajdywanie typów jednostek dla Code First działa tak samo jak postaci, w jakiej jest dla właściwości publicznej metod pobierających i ustawiających.

Pomoc techniczna dla wyliczenia — kod najpierw

27.09.2018 • 7 minutes to read • [Edit Online](#)

NOTE

EF5 począwszy tylko — funkcje, interfejsów API itp. z opisem na tej stronie zostały wprowadzone w programie Entity Framework 5. Jeśli używasz starszej wersji, niektóre lub wszystkie informacje, nie ma zastosowania.

W tym przewodniku krok po kroku i wideo pokazuje, jak za pomocą typach wyliczeniowych Entity Framework Code First. Ilustruje też sposób używania typów wyliczeniowych w zapytaniu programu LINQ.

W tym przewodniku będzie używać Code First, aby utworzyć nową bazę danych, ale można również użyć [Code First, aby zmapować istniejącą bazę danych](#).

Obsługa typu wyliczeniowego została wprowadzona w Entity Framework 5. Aby korzystać z nowych funkcji, takich jak typy wyliczeniowe, typy danych przestrzennych i funkcji z wartościami przechowywanymi w tabeli, należy wskazać .NET Framework 4.5. Program Visual Studio 2012 jest przeznaczony dla platformy .NET 4.5 domyślnie.

W programie Entity Framework, wyliczenie może mieć następujące typy bazowe: **bajtów, Int16, Int32, Int64**, lub **SByte**.

Obejrzyj wideo

To wideo pokazuje, jak za pomocą typach wyliczeniowych Entity Framework Code First. Ilustruje też sposób używania typów wyliczeniowych w zapytaniu programu LINQ.

Osoba prezentująca: Julia Kornich

Film wideo: [WMV](#) | [MP4](#) | [WMV \(ZIP\)](#)

Wymagania wstępne

Musisz być w wersji programu Visual Studio 2012 Ultimate, Premium, Professional i Web Express zainstalowany w celu przeprowadzenia tego instruktażu.

Konfigurowanie projektu

1. Otwórz program Visual Studio 2012
2. Na **pliku** menu wskaż **New**, a następnie kliknij przycisk **projektu**
3. W okienku po lewej stronie kliknij **Visual C#**, a następnie wybierz pozycję **konsoli** szablonu
4. Wprowadź **EnumCodeFirst** jako nazwę projektu i kliknij przycisk **OK**

Definiowanie nowego modelu za pomocą funkcji Code First

Korzystając z rozwiązania deweloperskiego Code First zwykle Rozpocznij od pisania klas .NET Framework, które definiują model koncepcyjny (domena). Poniższy kod definiuje klasę działu.

Ten kod definiuje również wyliczenie DepartmentNames. Domyślnie jest wyliczenie **int** typu. Właściwość Nazwa klasy działu jest typu DepartmentNames.

Otwórz plik Program.cs i wklej poniższe definicje klas.

```
public enum DepartmentNames
{
    English,
    Math,
    Economics
}

public partial class Department
{
    public int DepartmentID { get; set; }
    public DepartmentNames Name { get; set; }
    public decimal Budget { get; set; }
}
```

Definiowanie typu pochodnego typu DbContext

Oprócz definiowania jednostek, musisz zdefiniować klasę, która pochodzi od typu DbContext i udostępnia DbSet< TEntity > właściwości. DbSet< TEntity > właściwości umożliwiają kontekstu wiedzieć, jakie typy, które mają zostać uwzględnione w modelu.

Wystąpienie typu DbContext pochodzące zarządza obiektami jednostki w czasie wykonywania, zawierającą wypełnianie obiektów przy użyciu danych z bazy danych, zmień śledzenie i przechowywanie danych w bazie danych.

Typy DbContext i DbSet są definiowane w zestawie platformy EntityFramework. Firma Microsoft doda odwołanie do tej biblioteki DLL przy użyciu pakietu EntityFramework NuGet.

1. W Eksploratorze rozwiązań kliknij prawym przyciskiem myszy nazwę projektu.
2. Wybierz **Zarządzaj pakietami NuGet...**
3. W oknie dialogowym pakiety zarządzania NuGet wybierz **Online** kartę i wybierz polecenie **EntityFramework** pakietu.
4. Kliknij przycisk **instalacji**

Należy pamiętać, że oprócz zestawu platformy EntityFramework, odwołania do zestawów System.ComponentModel.DataAnnotations i System.Data.Entity są dodawane także.

W górnej części pliku Program.cs Dodaj następującą instrukcję using:

```
using System.Data.Entity;
```

W pliku Program.cs Dodaj definicję kontekstu.

```
public partial class EnumTestContext : DbContext
{
    public DbSet<Department> Departments { get; set; }
}
```

Utrwalanie i pobieranie danych

Otwórz plik Program.cs, w którym jest zdefiniowana metoda Main. Dodaj następujący kod do funkcji Main. Ten kod dodaje nowy obiekt działu do kontekstu. Następnie zapisuje dane. Kod wykonuje też zapytanie LINQ, które zwraca działu, gdzie nazwa jest DepartmentNames.English.

```
using (var context = new EnumTestContext())
{
    context.Departments.Add(new Department { Name = DepartmentNames.English });

    context.SaveChanges();

    var department = (from d in context.Departments
                      where d.Name == DepartmentNames.English
                      select d).FirstOrDefault();

    Console.WriteLine(
        "DepartmentID: {0} Name: {1}",
        department.DepartmentID,
        department.Name);
}
```

Skompilować i uruchomić aplikację. Program generuje następujące wyniki:

```
DepartmentID: 1 Name: English
```

Wyświetlanie wygenerowanych bazy danych

Po uruchomieniu aplikacji po raz pierwszy, platformy Entity Framework tworzy bazę danych. Dysponujemy programu Visual Studio 2012, baza danych zostanie utworzony w wystąpieniu programu LocalDB. Domyślnie platforma Entity Framework nazwy bazy danych po w pełni kwalifikowanej nazwie pochodnej kontekstu (w tym przykładzie, który jest **EnumCodeFirst.EnumTestContext**). Kolejne razy, zostanie użyta istniejącej bazy danych.

Należy pamiętać, że jeśli wprowadzisz zmiany do modelu, po utworzeniu bazy danych, należy użyć migracje Code First do zaktualizowania schematu bazy danych. Zobacz [Code First dla nowej bazy danych](#) na przykład za pomocą migracji.

Aby wyświetlić i bazy danych, wykonaj następujące czynności:

1. W menu głównym programu Visual Studio 2012, wybierz **widoku -> Eksplorator obiektów SQL Server**.
2. Jeśli LocalDB, nie ma na liście serwerów, kliknij prawym przyciskiem myszy **programu SQL Server** i wybierz **dodawania serwera SQL** Użyj domyślnej **uwierzytelniania Windows** połączyć się z Wystąpienia LocalDB
3. Rozwiń węzeł LocalDB
4. Unfold **baz danych** folder, aby zobaczyć nową bazę danych, a następnie przejdź do **działu** tabeli należy pamiętać, że Code First nie tworzy tabelę, która mapuje dane na typ wyliczeniowy
5. Aby wyświetlić dane, kliknij prawym przyciskiem myszy w tabeli i wybrać **widoku danych**

Podsumowanie

W tym przewodniku zobaczyliśmy, jak za pomocą typach wyliczeniowych Entity Framework Code First.

Przestrzenne - Code pierwszy

27.09.2018 • 8 minutes to read • [Edit Online](#)

NOTE

EF5 począwszy tylko — funkcje, interfejsów API itp. z opisem na tej stronie zostały wprowadzone w programie Entity Framework 5. Jeśli używasz starszej wersji, niektóre lub wszystkie informacje, nie ma zastosowania.

Przewodnik krok po kroku i wideo pokazuje, jak do mapowania typów przestrzennych z Entity Framework Code First. Ilustruje też sposób używania zapytania LINQ można znaleźć odległość między dwiema lokalizacjami.

W tym przewodniku będzie używać Code First, aby utworzyć nową bazę danych, ale można również użyć [Code First istniejącą bazę danych](#).

Obsługa przestrzenne typu została wprowadzona w programie Entity Framework 5. Należy pamiętać, że aby korzystać z nowych funkcji, takich jak typ przestrzennych, wyliczeń i funkcji z wartościami przechowywanymi w tabeli, należy wskazać .NET Framework 4.5. Program Visual Studio 2012 jest przeznaczony dla platformy .NET 4.5 domyślnie.

Używanie typów danych przestrzennych, należy również użyć dostawcy środowiska Entity Framework, który obsługuje przestrzennych. Zobacz [Obsługa dostawców dla typów przestrzennych](#) Aby uzyskać więcej informacji.

Istnieją dwa typy danych przestrzennych głównego: geometry i położenia geograficznego. Typ danych Geografia przechowuje dane ellipsoidal (na przykład współrzędne geograficzne GPS koordynuje). Typ danych Geometria reprezentuje euklidesowa współrzędnych (płaski).

Obejrzyj wideo

To wideo pokazuje, jak do mapowania typów przestrzennych z Entity Framework Code First. Ilustruje też sposób używania zapytania LINQ można znaleźć odległość między dwiema lokalizacjami.

Osoba prezentująca: Julia Kornich

Film wideo: [WMV](#) | [MP4](#) | [WMV \(ZIP\)](#)

Wymagania wstępne

Musisz być w wersji programu Visual Studio 2012 Ultimate, Premium, Professional i Web Express zainstalowany w celu przeprowadzenia tego instruktażu.

Konfigurowanie projektu

1. Otwórz program Visual Studio 2012
2. Na **pliku** menu wskaz **New**, a następnie kliknij przycisk **projektu**
3. W okienku po lewej stronie kliknij **Visual C#**, a następnie wybierz pozycję **konsoli** szablonu
4. Wprowadź **SpatialCodeFirst** jako nazwę projektu i kliknij przycisk **OK**

Definiowanie nowego modelu za pomocą funkcji Code First

Korzystając z rozwiązania deweloperskiego Code First zwykle Rozpocznij od pisania klas .NET Framework, które definiują model koncepcyjny (domena). Poniższy kod definiuje klasę University.

Uniwersytecie ma właściwość lokalizacji, typu DbGeography. Aby użyć typu DbGeography, należy dodać odwołanie do zestawu System.Data.Entity, a także dodać System.Data.Spatial za pomocą instrukcji.

Otwórz plik Program.cs i wklej następujące instrukcje using na początku pliku:

```
using System.Data.Spatial;
```

Dodaj następującą definicję klasy University do pliku Program.cs.

```
public class University
{
    public int UniversityID { get; set; }
    public string Name { get; set; }
    public DbGeography Location { get; set; }
}
```

Definiowanie typu pochodnego typu DbContext

Oprócz definiowania jednostek, musisz zdefiniować klasę, która pochodzi od typu DbContext i udostępnia DbSet< TEntity > właściwości. DbSet< TEntity > właściwości umożliwiają kontekstu wiedzieć, jakie typy, które mają zostać uwzględnione w modelu.

Wystąpienie typu DbContext pochodzące zarządza obiektami jednostki w czasie wykonywania, zawierającą wypełnianie obiektów przy użyciu danych z bazy danych, zmień śledzenie i przechowywanie danych w bazie danych.

Typy DbContext i DbSet są definiowane w zestawie platformy EntityFramework. Firma Microsoft doda odwołanie do tej biblioteki DLL przy użyciu pakietu EntityFramework NuGet.

1. W Eksploratorze rozwiązań kliknij prawym przyciskiem myszy nazwę projektu.
2. Wybierz **Zarządzaj pakietami NuGet...**
3. W oknie dialogowym pakiety zarządzania NuGet wybierz **Online** kartę i wybierz polecenie **EntityFramework** pakietu.
4. Kliknij przycisk **instalacji**

Należy zauważać, że oprócz zestawu platformy EntityFramework, odwołanie do zestawu System.ComponentModel.DataAnnotations jest także dodawane.

W górnej części pliku Program.cs Dodaj następującą instrukcję using:

```
using System.Data.Entity;
```

W pliku Program.cs Dodaj definicję kontekstu.

```
public partial class UniversityContext : DbContext
{
    public DbSet<University> Universities { get; set; }
}
```

Utrwalanie i pobieranie danych

Otwórz plik Program.cs, w którym jest zdefiniowana metoda Main. Dodaj następujący kod do funkcji Main.

Ten kod dodaje dwa nowe obiekty University do kontekstu. Właściwości przestrzenne są inicjowane za pomocą

metody `DbGeography.FromText`. Punkt lokalizacji geograficznej, reprezentowane jako `WellKnownText` jest przekazywany do metody. Kod następnie zapisuje dane. Następnie zapytania LINQ, która zwraca obiekt `University`, gdzie jego lokalizacja jest najbardziej zbliżony do określonej lokalizacji jest tworzony i wykonywane.

```
using (var context = new UniversityContext ())
{
    context.Universities.Add(new University()
    {
        Name = "Graphic Design Institute",
        Location = DbGeography.FromText("POINT(-122.336106 47.605049)"),
    });

    context.Universities.Add(new University()
    {
        Name = "School of Fine Art",
        Location = DbGeography.FromText("POINT(-122.335197 47.646711)"),
    });

    context.SaveChanges();

    var myLocation = DbGeography.FromText("POINT(-122.296623 47.640405)");

    var university = (from u in context.Universities
                      orderby u.Location.Distance(myLocation)
                      select u).FirstOrDefault();

    Console.WriteLine(
        "The closest University to you is: {0}.",
        university.Name);
}
```

Skompilować i uruchomić aplikację. Program generuje następujące wyniki:

```
The closest University to you is: School of Fine Art.
```

Wyświetlanie wygenerowanych bazy danych

Po uruchomieniu aplikacji po raz pierwszy, platformy Entity Framework tworzy bazę danych. Dysponujemy programu Visual Studio 2012, baza danych zostanie utworzony w wystąpieniu programu LocalDB. Domyślnie platforma Entity Framework nazwy bazy danych po w pełni kwalifikowanej nazwie pochodnej kontekstu (w tym przykładzie, który jest **SpatialCodeFirst.UniversityContext**). Kolejne razy, zostanie użyta istniejącej bazy danych.

Należy pamiętać, że jeśli wprowadzisz zmiany do modelu, po utworzeniu bazy danych, należy użyć migracje Code First do zaktualizowania schematu bazy danych. Zobacz [Code First dla nowej bazy danych](#) na przykład za pomocą migracji.

Aby wyświetlić i bazy danych, wykonaj następujące czynności:

1. W menu głównym programu Visual Studio 2012, wybierz **widoku** - > **Eksplorator obiektów SQL Server**.
2. Jeśli LocalDB, nie ma na liście serwerów, kliknij prawym przyciskiem myszy **programu SQL Server** i wybierz **dodawania serwera SQL** Użyj domyślnej **uwierzytelniania Windows** połączyć się z wystąpienia LocalDB
3. Rozwiń węzeł LocalDB
4. Unfold **baz danych** folder, aby zobaczyć nową bazę danych, a następnie przejdź do **uniwersytety** tabeli
5. Aby wyświetlić dane, kliknij prawym przyciskiem myszy w tabeli i wybrać **widoku danych**

Podsumowanie

W tym przewodniku zobaczyliśmy, jak za pomocą typów przestrzennych programu Entity Framework Code First.

Pierwszy konwencje związane z

13.09.2018 • 9 minutes to read • [Edit Online](#)

Kod najpierw umożliwia opisując modelu przy użyciu klas języka C# lub Visual Basic .NET. Wykryto kształtu podstawowego modelu przy użyciu Konwencji. Konwencje to zestawy reguł, które są używane do automatycznego konfigurowania modelu koncepcyjnego oparte na definicje klas, podczas pracy z usługą Code First. Konwencje są zdefiniowane w przestrzeni nazw System.Data.Entity.ModelConfiguration.Conventions.

Model można dodatkowo skonfigurować przy użyciu adnotacji danych lub interfejsu API fluent. Pierwszeństwo jest przyznawane konfiguracji za pomocą interfejsu API fluent adnotacje danych, a następnie Konwencji. Aby uzyskać więcej informacji, zobacz [adnotacje danych, interfejs Fluent API — relacje, interfejs Fluent API — typy & właściwości i Fluent interfejsu API za pomocą VB.NET](#).

Szczegółowa lista konwencje Code First jest dostępna w [dokumentacji interfejsu API](#). Ten temat zawiera omówienie Konwencji używanych przez rozwiązanie Code First.

Typ odnajdywania

Korzystając z rozwiązania deweloperskiego Code First zwykle Rozpoczni od pisania klas .NET Framework, które definiują model koncepcyjny (domena). Oprócz definiowania klasy, należy również umożliwić **DbContext** wiedzieć, jakie typy, które mają zostać uwzględnione w modelu. Aby to zrobić, należy zdefiniować klasy kontekstu, która pochodzi od klasy **DbContext** i udostępnia **DbSet** właściwości dla typów, które mają być częścią modelu. Kod najpierw będzie zawierać tych typów i również będzie ściągać wszystkie typy odwołania, nawet jeśli przywoływane typy są definiowane w innym zestawie.

Jeśli Twoje typy brać udziału w hierarchii dziedziczenia, jest wystarczający, aby zdefiniować **DbSet** właściwości dla klasy bazowej, a typów pochodnych zostanie automatycznie dołączony, jeśli są w tym samym zestawie jako klasa bazowa.

W poniższym przykładzie jest tylko jedna **DbSet** właściwości zdefiniowane w **SchoolEntities** klasy (**działów**). Kod najpierw używa tej właściwości do odnalezienia i pobieraj żadnych typów odwołania.

```

public class SchoolEntities : DbContext
{
    public DbSet<Department> Departments { get; set; }
}

public class Department
{
    // Primary key
    public int DepartmentID { get; set; }
    public string Name { get; set; }

    // Navigation property
    public virtual ICollection<Course> Courses { get; set; }
}

public class Course
{
    // Primary key
    public int CourseID { get; set; }

    public string Title { get; set; }
    public int Credits { get; set; }

    // Foreign key
    public int DepartmentID { get; set; }

    // Navigation properties
    public virtual Department Department { get; set; }
}

public partial class OnlineCourse : Course
{
    public string URL { get; set; }
}

public partial class OnsiteCourse : Course
{
    public string Location { get; set; }
    public string Days { get; set; }
    public System.DateTime Time { get; set; }
}

```

Aby wyłączyć typ z modelu, należy użyć **NotMapped** atrybutu lub **DbModelBuilder.Ignore** wygodnego interfejsu API.

```
modelBuilder.Ignore<Department>();
```

Konwencja klucza podstawowego

Kod najpierw wnioskuje, że właściwość jest kluczem podstawowym, czy właściwości klasy ma nazwę "ID" (bez uwzględniania wielkości liter), a następnie nazwę klasy "ID". Jeśli typ właściwość klucza podstawowego jest wartością liczbową lub identyfikator GUID zostanie on skonfigurowany jako kolumny tożsamości.

```

public class Department
{
    // Primary key
    public int DepartmentID { get; set; }

    . . .

}

```

Konwencja relacji

Platformy Entity Framework właściwości nawigacji Podaj sposób przechodzenia relację między dwoma typami encji. Każdy obiekt może mieć właściwości nawigacji dla każdej relacji, w których uczestniczy. Właściwości nawigacji pozwala na przechodzenie relacji i zarządzanie nimi w obu kierunkach, zwracając obiekt odwołania (Jeśli liczebność jest jedną lub zero lub jeden) lub kolekcji (Jeśli liczebność to wiele). Kod najpierw wnioskuje relacje na podstawie właściwości nawigacji zdefiniowany dla typów.

Oprócz właściwości nawigacji zaleca się, że zawsze zapisz właściwości klucza obcego na typy, które reprezentują obiekty zależne. Dowolną właściwość przy użyciu tego samego typu danych jako główną właściwość klucza podstawowego i nazwę, która jest zgodna z jedną z następujących formatów reprezentuje klucz obcy dla relacji: "<nazwy właściwości nawigacji><podmiotu zabezpieczeń Nazwa właściwości klucza podstawowego>','<Nazwa klasy jednostki><nazwa właściwość klucza podstawowego>", lub"<nazwa głównej właściwości klucza podstawowego>". Jeśli nie zostaną znalezione wiele dopasowań pierwszeństwo jest przyznawane w kolejności podanej powyżej. Wykrywanie klucza obcego nie jest uwzględniana wielkość liter. Po wykryciu właściwości klucza obcego Code First wnioskuje liczebność relacji, w oparciu o dopuszczania wartości Null klucza obcego. Jeśli właściwość ma wartość null następnie relacji jest zarejestrowany jako opcjonalna. w przeciwnym razie relacja jest zarejestrowany zgodnie z wymaganiami.

Jeśli klucz obcy dla jednostki zależne nie dopuszcza wartości null, następnie Code First ustawi usuwanie kaskadowe relacji. Jeśli klucz obcy dla jednostki zależne ma wartość null, Code First nie ustawi usuwanie kaskadowe relacji i gdy podmiot zabezpieczeń został usunięty klucz obcy zostanie ustawniona na wartość null. Liczebność i cascade Usuń zachowanie wykryta przez Konwencję może zostać przesłonięta przez przy użyciu interfejsu API fluent.

W poniższym przykładzie właściwości nawigacji i klucza obcego są używane do definiowania relacji między klasami dział i kursów.

```
public class Department
{
    // Primary key
    public int DepartmentID { get; set; }
    public string Name { get; set; }

    // Navigation property
    public virtual ICollection<Course> Courses { get; set; }
}

public class Course
{
    // Primary key
    public int CourseID { get; set; }

    public string Title { get; set; }
    public int Credits { get; set; }

    // Foreign key
    public int DepartmentID { get; set; }

    // Navigation properties
    public virtual Department Department { get; set; }
}
```

NOTE

Jeśli masz wiele relacji między tymi samymi typami (na przykład, założmy, że należy zdefiniować **osoby** i **książki** klas, gdzie **osoby** klasa zawiera **ReviewedBooks** i **AuthoredBooks** właściwości nawigacji i **książki** klasa zawiera **Autor** i **Weryfikacja** właściwości nawigacji), musisz ręcznie skonfigurować relacje przy użyciu adnotacji danych lub interfejsu API fluent. Aby uzyskać więcej informacji, zobacz [adnotacje danych — relacje](#) i [interfejs Fluent API — relacje](#).

Typy złożone Konwencji

Gdy Code First odnajduje definicję klasy, w którym klucz podstawowy, nie można wywnioskować, a nie klucza podstawowego jest zarejestrowana przy użyciu adnotacji danych lub interfejsu API fluent, ten typ jest automatycznie zarejestrowany jako typ złożony. Wykrywanie typu złożonego wymaga również, że typ nie ma właściwości, które odwołują się typów jednostek i nie odwołuje się właściwość kolekcji innego typu. Biorąc pod uwagę następujące definicje klas Code First może wywnioskować, **szczegóły** jest typem złożonym, ponieważ ma ona bez klucza podstawowego.

```
public partial class OnsiteCourse : Course
{
    public OnsiteCourse()
    {
        Details = new Details();
    }

    public Details Details { get; set; }
}

public class Details
{
    public System.DateTime Time { get; set; }
    public string Location { get; set; }
    public string Days { get; set; }
}
```

Konwencja ciąg połączenia

Aby dowiedzieć się o z konwencjami, czy kontekst DbContext używa do odnajdywania połączenie zobacz [połączenia i modele](#).

Usuwanie Konwencji

Możesz usunąć dowolne konwencje zdefiniowane w przestrzeni nazw `System.Data.Entity.ModelConfiguration.Conventions`. Poniższy przykład usuwa **PluralizingTableNameConvention**.

```
public class SchoolEntities : DbContext
{
    . .

    protected override void OnModelCreating(DbModelBuilder modelBuilder)
    {
        // Configure Code First to ignore PluralizingTableName convention
        // If you keep this convention, the generated tables
        // will have pluralized names.
        modelBuilder.Conventions.Remove<PluralizingTableNameConvention>();
    }
}
```

Konwencje niestandardowe

Konwencje niestandardowe są obsługiwane w EF6 i nowszych wersjach. Aby uzyskać więcej informacji, zobacz [konwencje pierwszy kod niestandardowy](#).

Konwencje pierwszy kod niestandardowy

13.09.2018 • 19 minutes to read • [Edit Online](#)

NOTE

EF6 począwszy tylko — funkcje, interfejsów API itp. z opisem na tej stronie zostały wprowadzone w programie Entity Framework 6. Jeśli używasz starszej wersji, niektóre lub wszystkie informacje, nie ma zastosowania.

Przy użyciu najpierw kod modelu jest obliczana z klas przy użyciu zestawu Konwencji. Wartość domyślna [pierwszy konwencje związane z](#) określa elementy, takie jak, których właściwość staje się klucz podstawowy jednostki, nazwa tabeli mapuje jednostki i jakie dokładności i skali dziesiętna kolumna ma domyślnie.

Czasami te domyślne Konwencje nie są idealne dla modelu, a trzeba pracować wokół nich przez skonfigurowanie wielu pojedynczych jednostek przy użyciu adnotacji danych lub interfejsu API Fluent. Niestandardowe pierwszy konwencje związane z umożliwiają definiowanie własnych Konwencji odpowiadającym, które zapewniają domyślne wartości dla modelu. W tym przewodniku omówimy różnego rodzaju konwencje niestandardowych oraz sposób tworzenia każdego z nich.

Konwencje opartych na modelu

Ta strona obejmuje interfejs API DbModelBuilder konwencje niestandardowych. Ten interfejs API powinny być wystarczające do tworzenia większość konwencje niestandardowych. Jednak istnieje także możliwość tworzenia opartych na modelu Konwencji — konwencje, które manipulują końcowego modelu, po jego utworzeniu — Obsługa zaawansowanych scenariuszy. Aby uzyskać więcej informacji, zobacz [opartych na modelu Konwencji](#).

Nasz Model

Zacznijmy od definiowania prosty model, który możemy użyć za pomocą naszych Konwencji. Dodaj następujące klasy do projektu.

```

using System;
using System.Collections.Generic;
using System.Data.Entity;
using System.Linq;

public class ProductContext : DbContext
{
    static ProductContext()
    {
        Database.SetInitializer(new DropCreateDatabaseIfModelChanges<ProductContext>());
    }

    public DbSet<Product> Products { get; set; }
}

public class Product
{
    public int Key { get; set; }
    public string Name { get; set; }
    public decimal? Price { get; set; }
    public DateTime? ReleaseDate { get; set; }
    public ProductCategory Category { get; set; }
}

public class ProductCategory
{
    public int Key { get; set; }
    public string Name { get; set; }
    public List<Product> Products { get; set; }
}

```

Wprowadzenie do niestandardowych Konwencji

Napiszmy Konwencji, który konfiguruje żadnej właściwości o nazwie klucza jako klucza podstawowego dla tego typu jednostki.

Konwencje są włączone w Konstruktorze modelu, którego dostęp można uzyskać poprzez zastąpienie OnModelCreating w kontekście. Aktualizacja klasy ProductContext w następujący sposób:

```

public class ProductContext : DbContext
{
    static ProductContext()
    {
        Database.SetInitializer(new DropCreateDatabaseIfModelChanges<ProductContext>());
    }

    public DbSet<Product> Products { get; set; }

    protected override void OnModelCreating(DbModelBuilder modelBuilder)
    {
        modelBuilder.Properties()
            .Where(p => p.Name == "Key")
            .Configure(p => p.IsKey());
    }
}

```

Teraz, będą wszystkich właściwości w naszym modelu o nazwie klucza skonfigurowany jako klucz podstawowy jednostki, niezależnie od jej części.

Również zapytała Konwencji naszego dokładniejszą przez filtrowanie według typu właściwości, który będziemy do skonfigurowania:

```
modelBuilder.Properties<int>()
    .Where(p => p.Name == "Key")
    .Configure(p => p.IsKey());
```

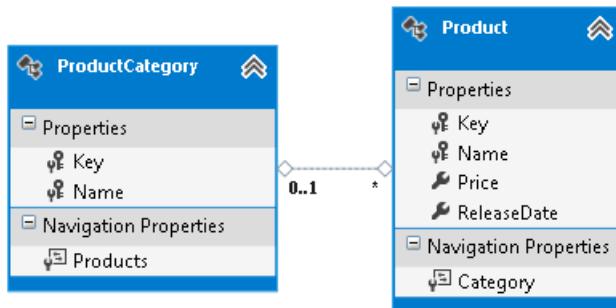
To spowoduje skonfigurowanie wszystkich właściwości o nazwie klucza jako podstawowego klucza ich jednostki, ale tylko wtedy, gdy są one liczbą całkowitą.

Funkcją interesującą metody IsKey jest dodatek. Co oznacza, że jeśli wywołujesz IsKey na wiele właściwości, a wszystkie staną się częścią klucza złożonego. Jedno zastrzeżenie: to jest, czy podczas określania wielu właściwości klucza należy także określić, zamówienie tych właściwości. Można to zrobić, wywołując HasColumnOrder metody, takie jak poniżej:

```
modelBuilder.Properties<int>()
    .Where(x => x.Name == "Key")
    .Configure(x => x.IsKey().HasColumnOrder(1));

modelBuilder.Properties()
    .Where(x => x.Name == "Name")
    .Configure(x => x.IsKey().HasColumnOrder(2));
```

Ten kod konfiguruje typy w naszym modelu ma klucz złożony składający się z nazwy kolumny klucza int i ciąg. Jeśli firma Microsoft umożliwia wyświetlenie modelu w Projektancie wyglądała następująco:



Inny przykład Konwencji właściwość to skonfigurować wszystkie właściwości daty/godziny w swój model do mapowania typu datetime2 w programie SQL Server zamiast daty/godziny. Można to osiągnąć przy użyciu następujących czynności:

```
modelBuilder.Properties<DateTime>()
    .Configure(c => c.HasColumnType("datetime2"));
```

Klasy Konwencji

Innym sposobem definiowania Konwencji jest użycie klasy Konwencji do hermetyzacji z Konwencji. Korzystając z klasy Konwencji, a następnie utworzyć typ, który dziedziczy z klasy Konwencji w przestrzeni nazw System.Data.Entity.ModelConfiguration.Conventions.

Mogemy utworzyć klasę Konwencji z Konwencją datetime2, który wcześniej pokazaliśmy, wykonując następujące czynności:

```
public class DateTime2Convention : Convention
{
    public DateTime2Convention()
    {
        this.Properties<DateTime>()
            .Configure(c => c.HasColumnType("datetime2"));
    }
}
```

Aby poinformować EF, użyj tej Konwencji, dodaj go do kolekcji konwencje w OnModelCreating, który jeśli wykonywano wraz z tym przewodnikiem będzie wyglądać następująco:

```
protected override void OnModelCreating(DbModelBuilder modelBuilder)
{
    modelBuilder.Properties<int>()
        .Where(p => p.Name.EndsWith("Key"))
        .Configure(p => p.IsKey());

    modelBuilder.Conventions.Add(new DateTime2Convention());
}
```

Jak widać, że wystąpienie Konwencji naszego możemy dodać do kolekcji Konwencji. Dziedziczenie z Konwencji oferuje wygodny sposób grupowania i udostępnianie konwencje przez zespoły lub projekty. Na przykład można mieć biblioteki klas w języku wspólny zbiór konwencji, że projekty wszystkie Twojej organizacji użyj.

Atrybuty niestandardowe

Innym zastosowaniem doskonałe Konwencji jest aby włączyć nowe atrybuty, które będą używane podczas konfigurowania modelu. Na przykład Utwórz atrybut, który możemy użyć, aby oznaczyć właściwości ciągu jako innego niż Unicode.

```
[AttributeUsage(AttributeTargets.Property, AllowMultiple = false)]
public class NonUnicode : Attribute
{}
```

Teraz Utwórzmy Konwencji zastosowaniu tego atrybutu w naszym modelu:

```
modelBuilder.Properties()
    .Where(x => x.GetCustomAttributes(false).OfType<NonUnicode>().Any())
    .Configure(c => cIsUnicode(false));
```

Z niniejszej Konwencji firma Microsoft można dodać atrybutu NonUnicode do żadnego z naszych właściwości ciągów, które oznacza, że kolumna w bazie danych będą przechowywane jako varchar zamiast nvarchar.

Jedną z rzeczy uwag dotyczących niniejszej Konwencji jest to, że jeśli atrybut NonUnicode zostanie umieszczony na coś innego niż właściwość ciągu, a następnie go spowoduje zgłoszenie wyjątku. Dzieje się tak, ponieważ nie można skonfigurowaćIsUnicode na dowolnego typu innego niż ciąg. Jeśli tak się stanie, następnie można wprowadzić swoje Konwencji bardziej szczegółowe tak, aby go odfiltrowuje wszystkie elementy, które nie jest ciąg.

Chociaż powyżej Konwencji działa w przypadku definiowania atrybutów niestandardowych, które ma innego interfejsu API, które mogą być znacznie łatwiejsze do użycia, szczególnie gdy zachodzi potrzeba użycia właściwości z klasy atrybutów.

W tym przykładzie użyjemy aktualizacja naszych atrybutu i zmień ją na atrybutIsUnicode, więc wygląda

następująco:

```
[AttributeUsage(AttributeTargets.Property, AllowMultiple = false)]
internal class IsUnicode : Attribute
{
    public bool Unicode { get; set; }

    public IsUnicode(bool isUnicode)
    {
        Unicode = isUnicode;
    }
}
```

Gdy będziemy już mieć, firma Microsoft można ustawić typu wartość logiczna na naszych atrybut Konwencji stwierdzić, czy właściwość powinna być Unicode. Firma Microsoft może zrobić w Konwencji, które mamy już uzyskując ClrProperty klasy konfiguracji następująco:

```
modelBuilder.Properties()
    .Where(x => x.GetCustomAttributes(false).OfType<IsUnicode>().Any())
    .Configure(c => cIsUnicode(c.Clr PropertyInfo.GetCustomAttribute<IsUnicode>().Unicode));
```

Jest to dość proste, ale jest bardziej zwięzły sposób realizacji tego celu za pomocą Having metoda konwencje interfejsu API. Having metoda ma parametr typu Func<PropertyInfo, T> który akceptuje PropertyInfo taka sama jak Where metody, ale oczekuje się, aby zwrócić obiekt. Jeśli zwracany obiekt ma wartość null, a następnie właściwość nie zostanie skonfigurowany, co oznacza można odfiltrować właściwości z nią tak samo jak miejsca, ale różni się w tym będzie również przechwytywanie zwróconego obiektu i przekazać go do metody konfiguracji. Działa to podobnie do poniższych:

```
modelBuilder.Properties()
    .Having(x =>x.GetCustomAttributes(false).OfType<IsUnicode>().FirstOrDefault())
    .Configure((config, att) => configIsUnicode(att.Unicode));
```

Atrybuty niestandardowe nie są Jedyne przypadek, kiedy używać Having metody przydaje się dowolnego miejsca, wymagających przeglądanie informacji o coś, co możesz filtrować podczas konfigurowania właściwości lub typów.

Konfigurowanie typów

Do tej pory wszystkie nasze konwencje zostały dla właściwości, ale istnieje inny obszar konwencje interfejsu API na temat konfigurowania typów w modelu. Proces jest podobny do Konwencji, które zauważono pory, ale opcje konfigurowania wewnętrznych będzie znajdować się w jednostce zamiast właściwości poziomu.

Jedną z rzeczy, które konwencje poziomu typu mogą być bardzo przydatne podczas ulegnie zmianie tabeli konwencji nazewnictwa, aby mapować do istniejącego schematu, która różni się od domyślnej EF lub Utwórz nową bazę danych z różnych konwencji nazewnictwa. W tym celu najpierw należy metodę, która może zaakceptować TypeInfo dla typu w naszym modelu i zwraca nazwę tabeli dla tego typu, co należy:

```
private string GetTableName(Type type)
{
    var result = Regex.Replace(type.Name, "[A-Z]", m => m.Value[0] + "_" + m.Value[1]);

    return result.ToLower();
}
```

Ta metoda przyjmuje typ i zwraca串, który używa małych znaków podkreślenia zamiast CamelCase. W naszym

modelu oznacza to, że klasa ProductCategory zostaną zmapowane do tabeli o nazwie produktu_kategorii zamiast oddzielały kategorie Productcategory.

Gdy będziemy już mieć tej metody można nazywamy je w Konwencji następująco:

```
modelBuilder.Types()
    .Configure(c => c.ToTable(GetTableName(c.ClrType)));
```

Ta konwencja konfiguruje każdy typ w naszym modelu do mapowania nazwy tabeli, który jest zwracany z metody naszych GetTableName. Ta konwencja jest odpowiednikiem wywołania metody ToTable dla każdej jednostki w modelu używając interfejsu API Fluent.

Jedno należy zwrócić uwagę na to jest, że po wywołaniu ToTable EF potrwa ciąg, który jest udostępniany jako nazwę tabeli dokładnie, bez jakichkolwiek pluralizacji, który normalnie jak podczas określania nazwy tabeli. To dlatego nazwa tabeli z Konwencji naszego produktu_kategorii zamiast produktu_kategorii. Możemy rozwiązać, w Konwencji naszego poprzez wywołanie usługi pluralizacja określić główną przyczynę.

W poniższym kodzie użyto [rozpoznawania zależności](#) funkcja, dodany do programu EF6 można pobrać usługi pluralizacja, który był używany EF i naszych Nazwa tabeli w liczbie mnogiej.

```
private string GetTableName(Type type)
{
    var pluralizationService = DbConfiguration.DependencyResolver.GetService<IPluralizationService>();

    var result = pluralizationService.Pluralize(type.Name);

    result = Regex.Replace(result, "[A-Z]", m => m.Value[0] + "_" + m.Value[1]);

    return result.ToLower();
}
```

NOTE

Ogólny wersję GetService jest metodą rozszerzenia w przestrzeni nazw System.Data.Entity.Infrastructure.DependencyResolution, musisz dodać za pomocą instrukcji do kontekstu w taki sposób, aby można było go używać.

ToTable i dziedziczenie

Innym ważnym aspektem ToTable jest to, że jeśli jawnie mapujesz typ danej tabeli, a następnie można zmienić strategię mapowania, która platforma EF użyje. Jeśli wywołasz ToTable dla każdego typu w hierarchii dziedziczenia, przekazując nazwę typu jako nazwę tabeli, tak jak opisano powyżej, następnie zostanie zmieniony strategii mapowania Tabela wg hierarchii (TPH) domyślny Tabela wg typu (TPT). Najlepszym sposobem, aby opisać to jest whith konkretny przykład:

```
public class Employee
{
    public int Id { get; set; }
    public string Name { get; set; }
}

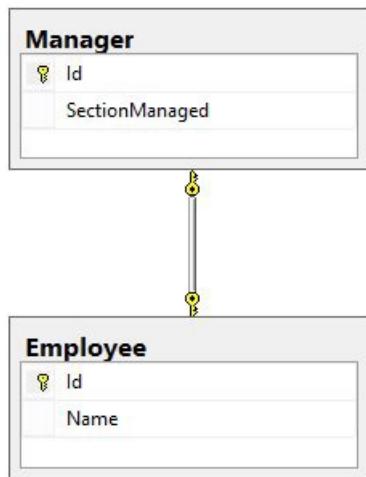
public class Manager : Employee
{
    public string SectionManaged { get; set; }
}
```

Domyślnie pracownika i Menedżera są mapowane na tej samej tabeli (pracownicy) w bazie danych. Tabela będzie

zawierać zarówno pracowników i menedżerów, a kolumna dyskryminatora, który poinformuje, jakiego typu wystąpienia są przechowywane w każdym wierszu. Jest to mapowanie TPH, ponieważ ma jedną tabelę dla hierarchii. Jednak jeśli wywołasz ToTable na obu klasse następuje każdego typu będzie zamiast tego można zamapować na własną tabelę, znany także jako TPT, ponieważ każdy typ ma własną tabelę.

```
modelBuilder.Types()
    .Configure(c=>c.ToTable(c.ClrType.Name));
```

Powyższy kod będzie zmapowana do struktury tabeli, która wygląda podobnie do poniższego:



Można tego uniknąć, a obsługa TPH domyślnego mapowania na kilka sposobów:

1. Wywołaj ToTable z taką samą nazwą tabeli, dla każdego typu w hierarchii.
2. Wywołaj ToTable tylko w klasie bazowej, hierarchii, w tym przykładzie, która byłaby pracownika.

Kolejność wykonywania

Konwencje działają w sposób ostatniego wins, taki sam jak interfejs Fluent API. Oznacza to, że jeśli piszesz dwóch Konwencji skonfigurowanych na tej samej opcji tej właściwości, a następnie ostatni z nich do wykonania usługi wins. Na przykład w poniższym kodzie maksymalna długość wszystkich ciągów ma wartość 500, ale możemy następnie skonfiguruj wszystkie właściwości w modelu, który ma mieć maksymalną długość równą 250 o nazwie Name.

```
modelBuilder.Properties<string>()
    .Configure(c => c.HasMaxLength(500));

modelBuilder.Properties<string>()
    .Where(x => x.Name == "Name")
    .Configure(c => c.HasMaxLength(250));
```

Ponieważ Konwencji, aby ustawić maksymalną długość do 250 po ten, który ustawia wszystkie ciągi na 500, wszystkie właściwości o nazwie Name w naszym modelu będą mieć MaxLength 250 podczas innych ciągów, takich jak opisy, będzie wynosić 500. Za pomocą Konwencji w ten sposób oznacza, że może zapewnić Konwencję ogólną dla typów lub właściwości w modelu i następnie zastąpić te je dla podzbiorów, które różnią się.

Fluent API i adnotacje danych może również zastąpić Konwencji w szczególnych przypadkach. W naszym powyższym przykładzie Jeśli firma Microsoft gdyby użyto Fluent API, aby ustawić maksymalną długość właściwości następnie może mieć testujemy go przed lub po Konwencji, ponieważ dokładniejszą Fluent API wygra nad bardziej ogólnymi Konwencji konfiguracji.

Konwencje wbudowane

Ponieważ konwencje niestandardowe mogą mieć wpływ domyślnych Konwencji Code First, może być przydatne do dodania konwencje do uruchomienia przed lub po innym Konwencji. W tym celu można użyć AddBefore i AddAfter metod zbierania konwencje na Twoje pochodnego typu DbContext. Poniższy kod zwiększałoby klasy Konwencji utworzony wcześniej tak, aby było uruchamiane przed wbudowanej w Konwencji klucza odnajdywania.

```
modelBuilder.Conventions.AddBefore<IdKeyDiscoveryConvention>(new DateTime2Convention());
```

Ma to być najbardziej przydatne podczas dodawania konwencje, które mają zostać uruchomione przed lub po wbudowanej Konwencji, lista wbudowanej konwencje można znaleźć tutaj: [Namespace System.Data.Entity.ModelConfiguration.Conventions](#).

Można również usunąć konwencje, które mają być stosowane do modelu. Aby usunąć z Konwencją, należy użyć metody Remove. Oto przykład usuwania PluralizingTableNameConvention.

```
protected override void OnModelCreating(DbModelBuilder modelBuilder)
{
    modelBuilder.Conventions.Remove<PluralizingTableNameConvention>();
}
```

Konwencje opartych na modelu

27.09.2018 • 9 minutes to read • [Edit Online](#)

NOTE

EF6 począwszy tylko — funkcje, interfejsów API itp. z opisem na tej stronie zostały wprowadzone w programie Entity Framework 6. Jeśli używasz starszej wersji, niektóre lub wszystkie informacje, nie ma zastosowania.

Model oparty konwencje są zaawansowane metody oparte na Konwencji model konfiguracji. W przypadku większości scenariuszy [niestandardowego kodu pierwszego Konwencji interfejsu API na DbModelBuilder](#) powinny być używane. Opis interfejsu API DbModelBuilder konwencje zaleca się przed rozpoczęciem korzystania z modelu na podstawie Konwencji.

Konwencje na podstawie modelu pozwala na tworzenie konwencje, które wpływają na właściwości i tabel, które nie są konfigurowane za pomocą standardowych konwencji. Przykładem tego są dyskryminatora kolumn w tabeli na hierarchii modeli i skojarzenia niezależnie od kolumny.

Tworzenie z Konwencją

Pierwszym krokiem w tworzeniu Konwencji model oparty jest wybór, gdy w potoku Konwencji musi zostać zastosowana do modelu. Istnieją dwa typy modelu Konwencji, Store (S-Space) i dotycząca pojęć (C-Space). Konwencja C + spacja jest stosowany do modelu, który aplikacja zostanie skompilowana Konwencji S miejsca jest stosowane do wersji modelu, który reprezentuje bazy danych i elementy kontrolki, takie jak jak automatycznie generowanych kolumny mają nazwy.

Konwencja modelu to klasa, która rozszerza `IConceptualModelConvention` lub `IStoreModelConvention`. Te interfejsy, oba akceptują typ ogólny, który może być typu `MetadataItem`, która jest używana do filtrowania typu danych, który dotyczy Konwencji.

Dodawanie Konwencję

Konwencje modelu zostaną dodane w taki sam sposób, jak regularne konwencje klasy. W **OnModelCreating** metody, Dodaj Konwencji do listy konwencje dla modelu.

```
using System.Data.Entity;
using System.Data.Entity.Core.Metadata.Edm;
using System.Data.Entity.Infrastructure;
using System.Data.Entity.ModelConfiguration.Conventions;

public class BlogContext : DbContext
{
    public DbSet<Post> Posts { get; set; }
    public DbSet<Comment> Comments { get; set; }

    protected override void OnModelCreating(DbModelBuilder modelBuilder)
    {
        modelBuilder.Conventions.Add<MyModelBasedConvention>();
    }
}
```

Można również dodać Konwencję względem innego Konwencja, za pomocą `Conventions.AddBefore<>` lub `Conventions.AddAfter<>` metody. Aby uzyskać więcej informacji dotyczących Konwencji, których dotyczy

platformy Entity Framework, zobacz sekcję Uwagi.

```
protected override void OnModelCreating(DbModelBuilder modelBuilder)
{
    modelBuilder.Conventions.AddAfter<IdKeyDiscoveryConvention>(new MyModelBasedConvention());
}
```

Przykład: Dyskryminatora modelu Konwencji

Zmianianie nazw kolumn, generowane przez EF jest przykładem coś, co nie można zrobić z innymi konwencjami interfejsów API. Jest to sytuacja w przypadku, gdy za pomocą modelu Konwencji jest jedynym rozwiązaniem.

Przykładowy sposób konfigurowania wygenerowanych kolumn przy użyciu konwencji model oparty jest dostosowanie sposobu, w kolumny dyskryminatora. Poniżej znajduje się przykład Konwencji prosty model na podstawie zmienia nazwę każdej kolumny w modelu o nazwie "Dyskryminatora" na "Obiektu EntityType". W tym kolumny, że deweloper po prostu o nazwie "Dyskryminatora". Ponieważ kolumna "Dyskryminatora" jest kolumną wygenerowanego musi on zostać uruchomiony w obszarze S.

```
using System.Data.Entity;
using System.Data.Entity.Core.Metadata.Edm;
using System.Data.Entity.Infrastructure;
using System.Data.Entity.ModelConfiguration.Conventions;

class DiscriminatorRenamingConvention : IStoreModelConvention<EdmProperty>
{
    public void Apply(EdmProperty property, DbModel model)
    {
        if (property.Name == "Discriminator")
        {
            property.Name = "EntityType";
        }
    }
}
```

Przykład: IA ogólne, zmiana nazwy Konwencji

Inny przykład bardziej złożonego modelu na podstawie Konwencji w akcji jest skonfigurować sposób noszą niezależnych skojarzenia (IAs). Jest to sytuacja, w którym Model konwencje są stosowane, ponieważ usługi IAs są generowane przez EF i nie są dostępne w modelu, które mogą uzyskiwać dostęp do interfejsu API DbModelBuilder.

Gdy EF generuje IA, tworzy kolumnę o nazwie EntityType_KeyName. Na przykład skojarzenie o nazwie klienta z kolumną klucza o nazwie CustomerId ta aplikacja wygeneruje kolumnę o nazwie Customer_CustomerId. Następujące paski Konwencji "_" znak poza nazwa kolumny, która jest generowana dla IA.

```

using System.Data.Entity;
using System.Data.Entity.Core.Metadata.Edm;
using System.Data.Entity.Infrastructure;
using System.Data.Entity.ModelConfiguration.Conventions;

// Provides a convention for fixing the independent association (IA) foreign key column names.
public class ForeignKeyNamingConvention : IStoreModelConvention<AssociationType>
{
    public void Apply(AssociationType association, DbModel model)
    {
        // Identify ForeignKey properties (including IAs)
        if (association.IsForeignKey)
        {
            // rename FK columns
            var constraint = association.Constraint;
            if (DoPropertiesHaveDefaultNames(constraint.FromProperties, constraint.ToRole.Name,
                constraint.ToProperties))
            {
                NormalizeForeignKeyProperties(constraint.FromProperties);
            }
            if (DoPropertiesHaveDefaultNames(constraint.ToProperties, constraint.FromRole.Name,
                constraint.FromProperties))
            {
                NormalizeForeignKeyProperties(constraint.ToProperties);
            }
        }
    }

    private bool DoPropertiesHaveDefaultNames(ReadOnlyMetadataCollection<EdmProperty> properties, string
        roleName, ReadOnlyMetadataCollection<EdmProperty> otherEndProperties)
    {
        if (properties.Count != otherEndProperties.Count)
        {
            return false;
        }

        for (int i = 0; i < properties.Count; ++i)
        {
            if (!properties[i].Name.EndsWith("_" + otherEndProperties[i].Name))
            {
                return false;
            }
        }
        return true;
    }

    private void NormalizeForeignKeyProperties(ReadOnlyMetadataCollection<EdmProperty> properties)
    {
        for (int i = 0; i < properties.Count; ++i)
        {
            int underscoreIndex = properties[i].Name.IndexOf('_');
            if (underscoreIndex > 0)
            {
                properties[i].Name = properties[i].Name.Remove(underscoreIndex, 1);
            }
        }
    }
}

```

Rozszerzanie istniejących konwencji

Jeśli należy napisać Konwencji, która jest podobna do jednego Konwencji, które platformy Entity Framework już ma zastosowanie do modelu zawsze można rozszerzyć tej Konwencji, aby uniknąć konieczności ponownego pisania jej od podstaw. Jest na przykład aby zmienić istniejący identyfikator dopasowania Konwencja, za pomocą

niestandardowego. Dodatkowa korzyść do przesłonięcia klucza Konwencji jest, że przeciążonej ma zostać wywołana tylko wtedy, gdy żaden klucz już wykryta lub jawnie skonfigurowane. Lista konwencje używanym przez program Entity Framework jest dostępny tutaj:

<http://msdn.microsoft.com/library/system.data.entity.modelconfiguration.conventions.aspx>.

```
using System.Data.Entity;
using System.Data.Entity.Core.Metadata.Edm;
using System.Data.Entity.Infrastructure;
using System.Data.Entity.ModelConfiguration.Conventions;
using System.Linq;

// Convention to detect primary key properties.
// Recognized naming patterns in order of precedence are:
// 1. 'Key'
// 2. [type name]Key
// Primary key detection is case insensitive.
public class CustomKeyDiscoveryConvention : KeyDiscoveryConvention
{
    private const string Id = "Key";

    protected override IEnumerable<EdmProperty> MatchKeyProperty(
        EntityType entityType, IEnumerable<EdmProperty> primitiveProperties)
    {
        Debug.Assert(entityType != null);
        Debug.Assert(primitiveProperties != null);

        var matches = primitiveProperties
            .Where(p => Id.Equals(p.Name, StringComparison.OrdinalIgnoreCase));

        if (!matches.Any())
        {
            matches = primitiveProperties
                .Where(p => (entityType.Name + Id).Equals(p.Name, StringComparison.OrdinalIgnoreCase));
        }

        // If the number of matches is more than one, then multiple properties matched differing only by
        // case--for example, "Key" and "key".
        if (matches.Count() > 1)
        {
            throw new InvalidOperationException("Multiple properties match the key convention");
        }

        return matches;
    }
}
```

Następnie należy dodać naszej nowej Konwencji przed istniejącą Konwencji klucza. Po dodamy CustomKeyDiscoveryConvention, możemy usunąć IdKeyDiscoveryConvention. Jeśli nie możemy usunąć istniejące IdKeyDiscoveryConvention, ta Konwencja będzie nadal pierwszeństwo Konwencji odnajdywania identyfikator, ponieważ jest uruchamiane, najpierw, ale w przypadku, gdy nie ma właściwości "key" zostanie znaleziony, zostanie uruchomiony Konwencji "id". Zobaczmy to zachowanie, ponieważ każda Konwencja widzi modelu jako zaktualizowany zgodnie z Konwencją poprzedniego (a nie na obsługiwaniu na nim niezależnie i wszystkie połączone ze sobą) powoduje, że jeśli na przykład poprzedniego Konwencji zaktualizowany nazwę kolumny, aby dopasować stanie się coś odsetek swoje niestandardowe Konwencji (jeśli wcześniej nazwę nie zainteresowania), a następnie go będą stosowane do tej kolumny.

```
public class BlogContext : DbContext
{
    public DbSet<Post> Posts { get; set; }
    public DbSet<Comment> Comments { get; set; }

    protected override void OnModelCreating(DbModelBuilder modelBuilder)
    {
        modelBuilder.Conventions.AddBefore<IdKeyDiscoveryConvention>(new CustomKeyDiscoveryConvention());
        modelBuilder.Conventions.Remove<IdKeyDiscoveryConvention>();
    }
}
```

Uwagi

Lista konwencje, które obecnie są stosowane przez program Entity Framework jest dostępna w tej dokumentacji MSDN: <http://msdn.microsoft.com/library/system.data.entity.modelconfiguration.conventions.aspx>. Ta lista jest pobierane bezpośrednio z naszego kodu źródłowego. Kod źródłowy platformy Entity Framework 6 jest dostępny w [GitHub](#) wiele z Konwencji używanych przez program Entity Framework jest dobrym punktami startowymi dla modelu niestandardowego na podstawie Konwencji.

Interfejs Fluent API — relacji

13.09.2018 • 10 minutes to read • [Edit Online](#)

NOTE

Ta strona zawiera informacje o konfigurowaniu relacji w modelu Code First przy użyciu interfejsu API fluent. Aby uzyskać ogólne informacje dotyczące relacji w EF i uzyskiwania dostępu i manipulowanie danymi za pomocą relacji, zobacz [relacje & właściwości nawigacji](#).

Podczas pracy z Code First, należy zdefiniować model przez definiowanie klas CLR Twojej domeny. Domyślnie program Entity Framework używa Code First Konwencji do mapowania klas schematu bazy danych. Jeśli używasz Code First konwencji nazewnictwa, w większości przypadków możesz polegać na Code First, aby zdefiniować relacje między tabelami, usługi na podstawie kluczy obcych i właściwości nawigacji, definiujące na klasy. Jeśli nie zgodne z konwencjami podczas definiowania klas, lub jeśli chcesz zmienić sposób pracy Konwencji, można użyć wygodnego interfejsu API lub anotacje danych do konfigurowania Twoich zajęciach, więc Code First można mapować relacje między tabelami.

Wprowadzenie

Podczas konfigurowania relacji z interfejsu API fluent, rozpoczynać wystąpienia EntityTypeConfiguration i następnie użyć do określenia typu relacji, które ta jednostka uczestniczy w HasRequired, HasOptional czy HasMany metody. Metody HasRequired i HasOptional przyjmują wyrażenie lambda reprezentujące właściwość nawigacji odwołania. Metoda HasMany pobiera Wyrażenie lambda reprezentujące właściwość nawigacji kolekcji. Następnie można skonfigurować właściwości nawigacji odwrotność przy użyciu metod WithRequired WithOptional i WithMany. Metody te mają przeciążenia, które nie przyjmują argumentów i może służyć do określenia kardynalności z tego jednokierunkowe.

Następnie można skonfigurować właściwości klucza obcego przy użyciu metody HasForeignKey. Ta metoda przyjmuje Wyrażenie lambda reprezentujące właściwość, która ma być używany jako klucza obcego.

Konfigurowanie relacji wymagana do opcjonalne (jeden do — Zero lub jeden)

Poniższy przykład umożliwia skonfigurowanie relacji jeden do zero lub jeden. OfficeAssignment ma właściwość InstructorID klucz podstawowy i klucz obcy, ponieważ nazwa właściwości nie jest zgodna z Konwencją, metoda HasKey jest używane do konfigurowania klucza podstawowego.

```
// Configure the primary key for the OfficeAssignment
modelBuilder.Entity<OfficeAssignment>()
    .HasKey(t => t.InstructorID);

// Map one-to-zero or one relationship
modelBuilder.Entity<OfficeAssignment>()
    .HasRequired(t => t.Instructor)
    .WithOptional(t => t.OfficeAssignment);
```

Konfigurowanie relacji, w którym obu końcach są wymagane (jeden do jednego)

W większości przypadków Entity Framework można wywnioskować typu jest zależna od i która podmiotu zabezpieczeń w relacji. Jednak gdy zarówno końców relacji są wymagane, lub obie strony są opcjonalne platformy Entity Framework nie może zidentyfikować zależnych od ustawień lokalnych i główną. Gdy wymagane są oba końce relacji, należy użyć WithRequiredPrincipal lub WithRequiredDependent po metodzie HasRequired. Po obu stronach relacji są opcjonalne, należy użyć WithOptionalPrincipal lub WithOptionalDependent po metodzie HasOptional.

```
// Configure the primary key for the OfficeAssignment  
modelBuilder.Entity<OfficeAssignment>()  
    .HasKey(t => t.InstructorID);  
  
modelBuilder.Entity<Instructor>()  
    .HasRequired(t => t.OfficeAssignment)  
    .WithRequiredPrincipal(t => t.Instructor);
```

Konfigurowanie relacji wiele do wielu

Poniższy kod służy do konfigurowania relacji wiele do wielu między typami kurs i instruktora. W poniższym przykładzie domyślnych Konwencji Code First są używane do tworzenia tabelę sprzężenia. W wyniku tworzenia CourseInstructor tabeli z kolumnami Course_CourseID i Instructor_InstructorID.

```
modelBuilder.Entity<Course>()  
    .HasMany(t => t.Instructors)  
    .WithMany(t => t.Courses)
```

Jeśli chcesz określić nazwę tabeli sprzężenia i nazwy kolumn w tabeli, należy wykonać dodatkowe czynności konfiguracyjne przy użyciu metody mapy. Poniższy kod generuje CourseInstructor tabeli z kolumnami CourseID i InstructorID.

```
modelBuilder.Entity<Course>()  
    .HasMany(t => t.Instructors)  
    .WithMany(t => t.Courses)  
    .Map(m =>  
    {  
        m.ToTable("CourseInstructor");  
        m.MapLeftKey("CourseID");  
        m.MapRightKey("InstructorID");  
    });
```

Konfigurowanie relacji z jedną właściwością nawigacji

(Nazywane również jednokierunkowe) jednokierunkową relacją jest, gdy właściwość nawigacji jest zdefiniowane tylko na jednym z końców relacji, a nie w obu. Zgodnie z Konwencją Code First zawsze interpretuje jednokierunkowa relacja jako jeden do wielu. Na przykład jeśli chcesz, aby bezpośredni związek pomiędzy instruktora oraz przypisane OfficeAssignment, w którym masz właściwości nawigacji na typie przez instruktorów, należy skonfigurować tę relację przy użyciu interfejsu API fluent.

```
// Configure the primary Key for the OfficeAssignment  
modelBuilder.Entity<OfficeAssignment>()  
    .HasKey(t => t.InstructorID);  
  
modelBuilder.Entity<Instructor>()  
    .HasRequired(t => t.OfficeAssignment)  
    .WithRequiredPrincipal();
```

Włączanie usuwanie kaskadowe

Usuwanie kaskadowe na relacji można skonfigurować przy użyciu metody WillCascadeOnDelete. Jeśli klucz obcy dla jednostki zależne nie dopuszcza wartości null, następnie Code First ustawia usuwanie kaskadowe relacji. Jeśli klucz obcy dla jednostki zależne ma wartość null, Code First nie ustawia usuwanie kaskadowe relacji i gdy podmiot zabezpieczeń został usunięty klucz obcy zostanie ustawiona na wartość null.

Możesz usunąć tych konwencji usuwanie kaskadowe przy użyciu:

```
modelBuilder.Conventions.Remove<OneToManyCascadeDeleteConvention>()
modelBuilder.Conventions.Remove<ManyToManyCascadeDeleteConvention>()
```

Poniższy kod konfiguruje relację jako wymaganą, a następnie wyłącza usuwanie kaskadowe.

```
modelBuilder.Entity<Course>()
    .IsRequired(t => t.Department)
    .WithMany(t => t.Courses)
    .HasForeignKey(d => d.DepartmentID)
    .WillCascadeOnDelete(false);
```

Konfigurowanie złożony klucz obcy

Jeśli klucz podstawowy dla typu dział obejmowało DepartmentID i nazwę właściwości, konfigurowania klucza podstawowego dla działu i klucz obcy dla typów kurs w następujący sposób:

```
// Composite primary key
modelBuilder.Entity<Department>()
    .HasKey(d => new { d.DepartmentID, d.Name });

// Composite foreign key
modelBuilder.Entity<Course>()
    .IsRequired(c => c.Department)
    .WithMany(d => d.Courses)
    .HasForeignKey(d => new { d.DepartmentID, d.DepartmentName });
```

Zmiana nazw klucz obcy, który nie jest zdefiniowany w modelu

Jeśli nie chcesz zdefiniować klucz obcy dla typu CLR, ale aby określić nazwę, jakie powinien mieć w bazie danych, wykonaj następujące czynności:

```
modelBuilder.Entity<Course>()
    .IsRequired(c => c.Department)
    .WithMany(t => t.Courses)
    .Map(m => m.MapKey("ChangedDepartmentID"));
```

Konfigurowanie nazwy klucza obcego, który nie jest zgodna z Konwencją pierwszy kodu

Właściwość klucza obcego w klasie kurs został wywołany SomeDepartmentID zamiast DepartmentID, należy wykonać następujące czynności, aby określić, że SomeDepartmentID jako klucza obcego:

```
modelBuilder.Entity<Course>()
    .IsRequired(c => c.Department)
    .WithMany(d => d.Courses)
    .HasForeignKey(c => c.SomeDepartmentID);
```

Model używany w przykładach

Poniższy model Code First jest używany dla przykładów na tej stronie.

```
using System.Data.Entity;
using System.Data.Entity.ModelConfiguration.Conventions;
// add a reference to System.ComponentModel.DataAnnotations DLL
using System.ComponentModel.DataAnnotations;
using System.Collections.Generic;
using System;

public class SchoolEntities : DbContext
{
    public DbSet<Course> Courses { get; set; }
    public DbSet<Department> Departments { get; set; }
    public DbSet<Instructor> Instructors { get; set; }
    public DbSet<OfficeAssignment> OfficeAssignments { get; set; }

    protected override void OnModelCreating(DbModelBuilder modelBuilder)
    {
        // Configure Code First to ignore PluralizingTableName convention
        // If you keep this convention then the generated tables will have pluralized names.
        modelBuilder.Conventions.Remove<PluralizingTableNameConvention>();
    }
}

public class Department
{
    public Department()
    {
        this.Courses = new HashSet<Course>();
    }
    // Primary key
    public int DepartmentID { get; set; }
    public string Name { get; set; }
    public decimal Budget { get; set; }
    public System.DateTime StartDate { get; set; }
    public int? Administrator { get; set; }

    // Navigation property
    public virtual ICollection<Course> Courses { get; private set; }
}

public class Course
{
    public Course()
    {
        this.Instructors = new HashSet<Instructor>();
    }
    // Primary key
    public int CourseID { get; set; }

    public string Title { get; set; }
    public int Credits { get; set; }

    // Foreign key
    public int DepartmentID { get; set; }

    // Navigation properties
    public virtual Department Department { get; set; }
```

```
    public virtual ICollection<Instructor> Instructors { get; private set; }

}

public partial class OnlineCourse : Course
{
    public string URL { get; set; }
}

public partial class OnsiteCourse : Course
{
    public OnsiteCourse()
    {
        Details = new Details();
    }

    public Details Details { get; set; }
}

public class Details
{
    public System.DateTime Time { get; set; }
    public string Location { get; set; }
    public string Days { get; set; }
}

public class Instructor
{
    public Instructor()
    {
        this.Courses = new List<Course>();
    }

    // Primary key
    public int InstructorID { get; set; }
    public string LastName { get; set; }
    public string FirstName { get; set; }
    public System.DateTime HireDate { get; set; }

    // Navigation properties
    public virtual ICollection<Course> Courses { get; private set; }
}

public class OfficeAssignment
{
    // Specifying InstructorID as a primary
    [Key()]
    public Int32 InstructorID { get; set; }

    public string Location { get; set; }

    // When Entity Framework sees Timestamp attribute
    // it configures ConcurrencyCheck and DatabaseGeneratedPattern=Computed.
    [Timestamp]
    public Byte[] Timestamp { get; set; }

    // Navigation property
    public virtual Instructor Instructor { get; set; }
}
```

Interfejs API Fluent — Konfigurowanie i mapowania właściwości i typy

28.11.2018 • 18 minutes to read • [Edit Online](#)

Podczas pracy z programem Entity Framework Code First to zachowanie domyślne jest do mapowania klas usługi POCO tabel przy użyciu zestawu konwencji wbudowanymi do programów EF. Czasami jednak użytkownik nie może lub nie powinny być zgodne z konwencjami te i zamapować jednostek na coś innego niż co zgodnie z konwencjami.

Istnieją dwa główne sposoby, można skonfigurować EF, aby użyć coś innego niż konwencje, to znaczy [adnotacje](#) lub interfejs fluent API w systemie szyfrowania plików. Adnotacje dotyczą tylko podzbiór funkcji interfejsu API fluent, więc istnieją scenariusze mapowania, które nie mogą być osiągnięte korzystanie z adnotacji. Ten artykuł jest przeznaczony do pokazania, jak skonfigurować właściwości za pomocą interfejsu API fluent.

Kod pierwszego interfejsu API fluent najczęściej odbywa się przez zastąpienie [OnModelCreating](#) metody w swojej pochodnej [DbContext](#). Poniższe przykłady są przeznaczone do przedstawiają sposób wykonywania różnych zadań przy użyciu wygodnego interfejsu API i umożliwiają skopiowanie kodu i dostosowanie go do potrzeb Twojego modelu, jeśli chcesz zobaczyć model, który może być używany jako — jest to znajduje się na końcu tego artykułu.

Ustawienia dla całego modelu

Schemat domyślny (od wersji EF6)

Uruchamianie platformy EF6 umożliwia metoda [HasDefaultSchema](#) na [DbModelBuilder](#) Określ schemat bazy danych do użycia dla wszystkich tabel, procedur składowanych, itp. To ustawienie domyślne, zostaną zastąpione dla obiektów, które jawnie skonfigurujeesz inny schemat.

```
modelBuilder.HasDefaultSchema("sales");
```

Konwencje niestandardowe (od wersji EF6)

Począwszy od tworzenia własnych konwencji odpowiadającym, aby uzupełnić te dołączone w Code First platformy EF6. Aby uzyskać więcej informacji, zobacz [konwencje pierwszy kod niestandardowy](#).

Mapowanie właściwości

[Właściwość](#) metoda służy do konfigurowania atrybutów dla każdej właściwości należących do jednostki lub typ złożony. Metoda właściwość jest używana do uzyskiwania obiektu konfiguracji dla danej właściwości. Opcje na obiekt konfiguracji są specyficzne dla typu skonfigurowana; Na przykład jest dostępne tylko na właściwości parametrów [IsUnicode](#).

Konfigurowanie klucza podstawowego

Konwencja platformy Entity Framework, kluczy podstawowych jest:

1. Klasa definiuje właściwość, której nazwa to "ID" lub "Id"
2. lub nazwa klasy, po której następuje "ID" lub "Id"

Aby jawnie ustawić właściwość, która ma być kluczem podstawowym, można użyć metody [HasKey](#). W poniższym przykładzie metoda [HasKey](#) służy do konfigurowania [InstructorID](#) klucz podstawowy dla typu [OfficeAssignment](#).

```
modelBuilder.Entity<OfficeAssignment>().HasKey(t => t.InstructorID);
```

Konfigurowanie złożony klucz podstawowy

Poniższy przykład umożliwia skonfigurowanie DepartmentID i nazwy właściwości na złożony klucz podstawowy typu działu.

```
modelBuilder.Entity<Department>().HasKey(t => new { t.DepartmentID, t.Name });
```

Wyłączanie tożsamości liczbowych kluczy podstawowych

Poniższy przykład ustawia właściwość DepartmentID

System.ComponentModel.DataAnnotations.DatabaseGeneratedOption.None, aby wskazać, że wartość nie zostanie wygenerowany przez bazę danych.

```
modelBuilder.Entity<Department>().Property(t => t.DepartmentID)
    .HasDatabaseGeneratedOption(DatabaseGeneratedOption.None);
```

Określanie maksymalnego we właściwości

W poniższym przykładzie nazwa właściwości, należy nie może przekraczać 50 znaków. Jeśli wprowadzisz wartość, która jest dłuższa niż 50 znaków, otrzymasz [DbEntityValidationException](#) wyjątku. Jeśli Code First tworzy bazę danych z tego modelu do 50 znaków zostanie ustawiony także maksymalna długość nazwy kolumny.

```
modelBuilder.Entity<Department>().Property(t => t.Name).HasMaxLength(50);
```

Konfigurowanie wymaganych właściwości

W poniższym przykładzie właściwość Name jest wymagany. Jeśli nie określiš nazwy, wystąpi wyjątek [DbEntityValidationException](#). Jeśli Code First tworzy bazę danych z tego modelu kolumny używane do przechowywania tej właściwości zwykle będzie inne niż null.

NOTE

W niektórych przypadkach może nie być możliwe dla kolumny w bazie danych, może nie dopuszczać wartości null, nawet jeśli właściwość jest wymagana. Na przykład, gdy przy użyciu danych strategii TPH dziedziczenia dla wielu typów są przechowywane w jednej tabeli. Jeśli typ pochodny obejmuje wymagana właściwość kolumny nie można dokonać innych niż null, ponieważ nie wszystkie typy w hierarchii mają ta właściwość.

```
modelBuilder.Entity<Department>().Property(t => t.Name).IsRequired();
```

Konfigurowanie indeksu na co najmniej jednej właściwości

NOTE

EF6.1 począwszy tylko - atrybutu indeksu została wprowadzona w programie Entity Framework 6.1. Informacje przedstawione w tej sekcji nie ma zastosowania, jeśli używasz starszej wersji.

Tworzenie indeksów nie jest natywnie obsługiwane przez interfejs Fluent API, ale umożliwia korzystanie z pomocy technicznej dla **IndexAttribute** za pośrednictwem interfejsu API Fluent. Atrybuty indeksu są przetwarzane przez dołączenie adnotacją modelu na model, który następnie jest przekształcane w indeksu w bazie danych później w potoku. Można ręcznie dodać te same adnotacje przy użyciu interfejsu API Fluent.

W tym celu najłatwiej można utworzyć wystąpienia **IndexAttribute** zawierający wszystkie ustawienia dla nowego indeksu. Następnie można utworzyć wystąpienia **IndexAnnotation** czyli EF określonego typu który przekonwertuje **IndexAttribute** ustawienia do adnotacji modelu, które mogą być przechowywane w modelu platformy EF. Te mogą być następnie przekazywany do **HasColumnAnnotation** metody w interfejsie API Fluent, określając nazwę **indeksu** dla adnotacji.

```
modelBuilder
    .Entity<Department>()
    .Property(t => t.Name)
    .HasColumnAnnotation("Index", new IndexAnnotation(new IndexAttribute()));
```

Aby uzyskać pełną listę ustawień dostępnych w **IndexAttribute**, zobacz *indeksu* części [adnotacje danych na pierwszym kodzie](#). W tym dostosowywania nazwę indeksu, tworzenie indeksów unikatowych i tworzenie indeksów wielokolumnowych.

Można określić wiele adnotacji indeksu na jedną właściwość, przekazując tablicę **IndexAttribute** do konstruktora **IndexAnnotation**.

```
modelBuilder
    .Entity<Department>()
    .Property(t => t.Name)
    .HasColumnAnnotation(
        "Index",
        new IndexAnnotation(new[]
        {
            new IndexAttribute("Index1"),
            new IndexAttribute("Index2") { IsUnique = true }
        }));
    ));;
```

Określanie nie można zamapować właściwość CLR z kolumną w bazie danych

Poniższy przykład pokazuje, jak określić, czy właściwość na typ CLR nie została zamapowana na kolumnę w bazie danych.

```
modelBuilder.Entity<Department>().Ignore(t => t.Budget);
```

Mapowanie właściwości CLR do określonej kolumny w bazie danych

Poniższy przykład mapuje właściwość CLR nazwę kolumny bazy danych `DepartmentName`.

```
modelBuilder.Entity<Department>()
    .Property(t => t.Name)
    .HasColumnName("DepartmentName");
```

Zmiana nazw klucz obcy, który nie jest zdefiniowany w modelu

Jeśli nie chcesz zdefiniować klucz obcy dla typu CLR, ale aby określić nazwę, jakie powinien mieć w bazie danych, wykonaj następujące czynności:

```
modelBuilder.Entity<Course>()
    .IsRequired(c => c.Department)
    .WithMany(t => t.Courses)
    .Map(m => m.MapKey("ChangedDepartmentID"));
```

Skonfigurowanie, czy właściwość ciągu obsługuje zawartość Unicode

Domyślnie ciągi są Unicode (nvarchar w programie SQL Server). Metoda `IsUnicode` służy do określenia, czy ciąg

powinien być typu varchar.

```
modelBuilder.Entity<Department>()
    .Property(t => t.Name)
    .IsUnicode(false);
```

Konfigurowanie typu danych kolumny bazy danych

[HasColumnType](#) metoda umożliwia mapowanie na różne reprezentacje tego samego typu podstawowego. Przy użyciu tej metody nie włącza wykonywania konwersji danych w czasie wykonywania. Należy pamiętać, że [IsUnicode](#) preferowanym sposobem tworzenia kolumn ustawnienie varchar, ponieważ jest niezależny od bazy danych.

```
modelBuilder.Entity<Department>()
    .Property(p => p.Name)
    .HasColumnType("varchar");
```

Konfigurowanie właściwości typu złożonego

Istnieją dwa sposoby konfigurowania właściwości skalarne w typie złożonym.

Właściwości można wywołać w [ComplexTypeConfiguration](#).

```
modelBuilder.ComplexType<Details>()
    .Property(t => t.Location)
    .HasMaxLength(20);
```

Zapisu kropkowego umożliwia również dostęp do właściwości typu złożonego.

```
modelBuilder.Entity<OnsiteCourse>()
    .Property(t => t.Details.Location)
    .HasMaxLength(20);
```

Konfigurowanie właściwości, aby służyć jako Token optymistycznej współbieżności

Aby określić, że właściwości w obiekcie reprezentuje tokenem współbieżności, można użyć atrybutu [ConcurrencyCheck](#) lub metoda [IsConcurrencyToken](#).

```
modelBuilder.Entity<OfficeAssignment>()
    .Property(t => t.Timestamp)
    .IsConcurrencyToken();
```

Metoda [IsRowVersion](#) służy również do skonfigurowania właściwości, aby być w wersji wierszy w bazie danych. Ustawianie właściwości jako wersja wiersza automatycznie skonfiguruje je jako token optymistycznej współbieżności.

```
modelBuilder.Entity<OfficeAssignment>()
    .Property(t => t.Timestamp)
    .IsRowVersion();
```

Mapowanie typu

Określanie, czy klasa jest typem złożonym

Zgodnie z Konwencją typ, który ma określony klucz podstawowy jest traktowany jako typ złożony. Istnieją

sytuacje, w którym Code First nie wykrywa typ złożony (na przykład, jeśli użytkownik ma właściwość o nazwie identyfikator, ale nie oznaczają, aby mogła być kluczem podstawowym). W takich przypadkach należy użyć interfejsu API fluent jawnie określić, że typ jest typem złożonym.

```
modelBuilder.ComplexType<Details>();
```

Określanie nie można zamapować typ CLR jednostki do tabeli w bazie danych

Poniższy przykład pokazuje, jak wykluczyć typu CLR z mapowany do tabeli w bazie danych.

```
modelBuilder.Ignore<OnlineCourse>();
```

Mapowania typ jednostki do określonej tabeli w bazie danych

Wszystkie właściwości działu zostanie zamapowane do kolumn w tabeli o nazwie t_ działu.

```
modelBuilder.Entity<Department>()
    .ToTable("t_Department");
```

Można również określić nazwę schematu następująco:

```
modelBuilder.Entity<Department>()
    .ToTable("t_Department", "school");
```

Mapowanie dziedziczenia tabela wg hierarchii (TPH)

W tym scenariuszu mapowania TPH wszystkich typów w hierarchii dziedziczenia są mapowane na pojedynczą tabelę. Kolumna dyskryminatora jest używany do identyfikowania typu każdego wiersza. Podczas tworzenia modelu przy użyciu Code First, TPH jest strategia domyślna dla typów, które uczestniczą w hierarchii dziedziczenia. Domyślnie kolumna dyskryminatora zostanie dodana do tabeli o nazwie "Dyskryminatora" i nazwę typu CLR poszczególnych typów w hierarchii jest używany dla wartości dyskryminatora. Zachowanie domyślne można modyfikować za pomocą interfejsu API fluent.

```
modelBuilder.Entity<Course>()
    .Map<Course>(m => m.Requires("Type").HasValue("Course"))
    .Map<OnsiteCourse>(m => m.Requires("Type").HasValue("OnsiteCourse"));
```

Mapowanie dziedziczenia tabela wg typu (TPT)

W tym scenariuszu mapowania TPT wszystkie typy są mapowane na poszczególnych tabel. Właściwości, które należą wyłącznie do typu podstawowego lub typu pochodnego są przechowywane w tabeli, która mapuje do tego typu. Tabele mapowane na typy pochodne również przechowują klucz obcy, który tworzy sprzężenie tabeli pochodnej z tabeli podstawowej.

```
modelBuilder.Entity<Course>().ToTable("Course");
modelBuilder.Entity<OnsiteCourse>().ToTable("OnsiteCourse");
```

Mapowanie dziedziczenia klasy tabeli na konkretny (TPC)

W tym scenariuszu mapowania TPC wszystkie typy nieabstrakcyjnej w hierarchii są mapowane na poszczególnych tabel. Tabele, które mapują do klas pochodnych nie mają relacji do tabeli, która mapuje do klasy bazowej w bazie danych. Wszystkie właściwości klasy, w tym właściwości dziedziczonych są mapowane na kolumny odpowiedniej tabeli.

Wywołaj metodę MapInheritedProperties, aby skonfigurować każdego typu pochodnego.

MapInheritedProperties ponownie mapuje wszystkie właściwości, które były dziedziczone z klasy bazowej do nowych kolumn w tabeli dla klasy pochodnej.

NOTE

Należy pamiętać, że ponieważ nie mają tabele uczestniczących w hierarchii dziedziczenia TPC klucz podstawowy będzie jednostki zduplikowane klucze podczas wstawiania do tabel, które są mapowane do podklasy, jeśli masz wartości bazy danych, wygenerowane za pomocą tego samego Inicjator właściwości identity. Aby rozwiązać ten problem, możesz określić wartość różnych inicjatora początkowej dla każdej tabeli lub wyłączyć tożsamość na właściwość klucza podstawowego. Tożsamość jest wartością domyślną dla właściwości klucza liczby całkowitej, podczas pracy z usługą Code First.

```
modelBuilder.Entity<Course>()
    .Property(c => c.CourseID)
    .HasDatabaseGeneratedOption(DatabaseGeneratedOption.None);

modelBuilder.Entity<OnsiteCourse>().Map(m =>
{
    m.MapInheritedProperties();
    m.ToTable("OnsiteCourse");
});

modelBuilder.Entity<OnlineCourse>().Map(m =>
{
    m.MapInheritedProperties();
    m.ToTable("OnlineCourse");
});
```

Mapowanie właściwości typu jednostki z wieloma tabelami w bazie danych (jednostka dzielenie)

Jednostki podział umożliwia właściwości typu jednostki, aby być rozkładane na wiele tabel. W poniższym przykładzie jednostki dział zostanie podzielona na dwie tabele: dział i DepartmentDetails. Podział jednostki używa wielu wywołań metody mapy do mapowania podzbiorów właściwości do określonej tabeli.

```
modelBuilder.Entity<Department>()
    .Map(m =>
    {
        m.Properties(t => new { t.DepartmentID, t.Name });
        m.ToTable("Department");
    })
    .Map(m =>
    {
        m.Properties(t => new { t.DepartmentID, t.Administrator, t.StartDate, t.Budget });
        m.ToTable("DepartmentDetails");
    });
}
```

Mapowanie typów jednostek do jednej tabeli w bazie danych (tabeli dzielenie)

Poniższy przykład mapuje dwóch typów jednostek, które mają klucz podstawowy do jednej tabeli.

```
modelBuilder.Entity<OfficeAssignment>()
    .HasKey(t => t.InstructorID);

modelBuilder.Entity<Instructor>()
    .HasRequired(t => t.OfficeAssignment)
    .WithRequiredPrincipal(t => t.Instructor);

modelBuilder.Entity<Instructor>()..ToTable("Instructor");

modelBuilder.Entity<OfficeAssignment>()..ToTable("Instructor");
```

Mapowanie typu jednostki do wstawiania/aktualizowania/usuwania procedur składowanych (od wersji EF6)

Uruchamianie platformy EF6 można mapować jednostki używanie procedur składowanych do wstawiania, aktualizacji i usuwania. Aby uzyskać więcej informacji, zobacz [kodu pierwszy wstawiania/aktualizowania/usuwania procedur składowanych](#).

Model używany w przykładach

Poniższy model Code First jest używany dla przykładów na tej stronie.

```
using System.Data.Entity;
using System.Data.Entity.ModelConfiguration.Conventions;
// add a reference to System.ComponentModel.DataAnnotations DLL
using System.ComponentModel.DataAnnotations;
using System.Collections.Generic;
using System;

public class SchoolEntities : DbContext
{
    public DbSet<Course> Courses { get; set; }
    public DbSet<Department> Departments { get; set; }
    public DbSet<Instructor> Instructors { get; set; }
    public DbSet<OfficeAssignment> OfficeAssignments { get; set; }

    protected override void OnModelCreating(DbModelBuilder modelBuilder)
    {
        // Configure Code First to ignore PluralizingTableName convention
        // If you keep this convention then the generated tables will have pluralized names.
        modelBuilder.Conventions.Remove<PluralizingTableNameConvention>();
    }
}

public class Department
{
    public Department()
    {
        this.Courses = new HashSet<Course>();
    }
    // Primary key
    public int DepartmentID { get; set; }
    public string Name { get; set; }
    public decimal Budget { get; set; }
    public System.DateTime StartDate { get; set; }
    public int? Administrator { get; set; }

    // Navigation property
    public virtual ICollection<Course> Courses { get; private set; }
}

public class Course
{
    public Course()
    {
        this.Instructors = new HashSet<Instructor>();
    }
    // Primary key
    public int CourseID { get; set; }

    public string Title { get; set; }
    public int Credits { get; set; }

    // Foreign key
    public int DepartmentID { get; set; }

    // Navigation properties
    public virtual Department Department { get; set; }
    public virtual ICollection<Instructor> Instructors { get; private set; }
```

```
        public virtual ICollection<Course> Courses { get; private set; }

    }

    public partial class OnlineCourse : Course
    {
        public string URL { get; set; }
    }

    public partial class OnsiteCourse : Course
    {
        public OnsiteCourse()
        {
            Details = new Details();
        }

        public Details Details { get; set; }
    }

    public class Details
    {
        public System.DateTime Time { get; set; }
        public string Location { get; set; }
        public string Days { get; set; }
    }

    public class Instructor
    {
        public Instructor()
        {
            this.Courses = new List<Course>();
        }

        // Primary key
        public int InstructorID { get; set; }
        public string LastName { get; set; }
        public string FirstName { get; set; }
        public System.DateTime HireDate { get; set; }

        // Navigation properties
        public virtual ICollection<Course> Courses { get; private set; }
    }

    public class OfficeAssignment
    {
        // Specifying InstructorID as a primary
        [Key()]
        public Int32 InstructorID { get; set; }

        public string Location { get; set; }

        // When Entity Framework sees Timestamp attribute
        // it configures ConcurrencyCheck and DatabaseGeneratedPattern=Computed.
        [Timestamp]
        public Byte[] Timestamp { get; set; }

        // Navigation property
        public virtual Instructor Instructor { get; set; }
    }
}
```

Interfejs Fluent API przy użyciu VB.NET

27.09.2018 • 11 minutes to read • [Edit Online](#)

Kod umożliwia najpierw zdefiniować model za pomocą języka C# lub klasy VB.NET. Opcjonalnie można wykonać dodatkowe czynności konfiguracyjne przy użyciu atrybutów klas i właściwości lub za pomocą interfejsu API fluent. W tym instruktażu pokazano, jak przeprowadzić fluent Konfiguracja interfejsu API przy użyciu VB.NET.

Ta strona przyjęto założenie, że masz podstawową wiedzę na temat Code First. Zapoznaj się z przewodnikach, aby uzyskać więcej informacji na temat Code First:

- [Najpierw kod do nowej bazy danych](#)
- [Najpierw kod istniejącej bazy danych](#)

Wymagania wstępne

Musisz mieć co najmniej programu Visual Studio 2010 lub Visual Studio 2012 są zainstalowane w tym przewodniku.

Jeśli używasz programu Visual Studio 2010, należy również mieć [NuGet](#) zainstalowane

Tworzenie aplikacji

Aby zachować ich prostotę zamierzamy utworzyć aplikację podstawowa konsola, która używa Code First do dostępu do danych.

- Otwórz program Visual Studio
- **Plik —> New -> projektu...**
- Wybierz **Windows** menu po lewej stronie i **aplikacji konsoli**
- Wprowadź **CodeFirstVBSSample** jako nazwę
- Wybierz **OK**

Zdefiniuj Model

W tym kroku zdefiniujesz VB.NET obiektów POCO typów jednostek, które reprezentują model koncepcyjny. Klasy nie trzeba implementować żadnych interfejsów lub pochodzić od żadnych klas bazowych.

- Dodaj nową klasę do projektu, wprowadź **SchoolModel** nazwy klasy
- Zastęp zawartość nowej klasy, używając następującego kodu

```
Public Class Department
    Public Sub New()
        Me.Courses = New List(Of Course)()
    End Sub

    ' Primary key
    Public Property DepartmentID() As Integer
    Public Property Name() As String
    Public Property Budget() As Decimal
    Public Property StartDate() As Date
    Public Property Administrator() As Integer?
    Public Overridable Property Courses() As ICollection(Of Course)
End Class

Public Class Course
```

```

    Public Sub New()
        Me.Instructors = New HashSet(Of Instructor)()
    End Sub

    ' Primary key
    Public Property CourseID() As Integer
    Public Property Title() As String
    Public Property Credits() As Integer

    ' Foreign key that does not follow the Code First convention.
    ' The fluent API will be used to configure DepartmentID_FK to be the foreign key for this entity.
    Public Property DepartmentID_FK() As Integer

    ' Navigation properties
    Public Overridable Property Department() As Department
    Public Overridable Property Instructors() As ICollection(Of Instructor)
End Class

Public Class OnlineCourse
Inherits Course

    Public Property URL() As String
End Class

Partial Public Class OnsiteCourse
Inherits Course

    Public Sub New()
        Details = New OnsiteCourseDetails()
    End Sub

    Public Property Details() As OnsiteCourseDetails
End Class

' Complex type
Public Class OnsiteCourseDetails
    Public Property Time() As Date
    Public Property Location() As String
    Public Property Days() As String
End Class

Public Class Person
    ' Primary key
    Public Property PersonID() As Integer
    Public Property LastName() As String
    Public Property FirstName() As String
End Class

Public Class Instructor
Inherits Person

    Public Sub New()
        Me.Courses = New List(Of Course)()
    End Sub

    Public Property HireDate() As Date

    ' Navigation properties
    Private privateCourses As ICollection(Of Course)
    Public Overridable Property Courses() As ICollection(Of Course)
    Public Overridable Property OfficeAssignment() As OfficeAssignment
End Class

Public Class OfficeAssignment
    ' Primary key that does not follow the Code First convention.
    ' The HasKey method is used later to configure the primary key for the entity.
    Public Property InstructorID() As Integer

    Public Property Location() As String

```

```
Public Property Timestamp() As Byte()

    ' Navigation property
    Public Overridable Property Instructor() As Instructor
End Class
```

Definiowanie kontekstu pochodne

Jesteśmy o, aby rozpocząć używanie typów z programu Entity Framework, dlatego musimy Dodaj pakiet NuGet platformy EntityFramework.

- ** Projektu —> **Zarządzaj pakietami NuGet...**

NOTE

Jeśli nie masz **Zarządzaj pakietami NuGet...** opcja, należy zainstalować [najnowszej wersji pakietu NuGet](#)

- Wybierz **Online** kartę
- Wybierz **EntityFramework** pakietu
- Kliknij przycisk **instalacji**

Teraz nadszedł czas, aby zdefiniować pochodnej kontekst, który reprezentuje sesję z bazą danych, dzięki czemu nam zapytania i Zapisz dane. Definiujemy kontekstu, który pochodzi z System.Data.Entity.DbContext i udostępnia wpisane DbSet< TEntity > dla każdej klasy w naszym modelu.

- Dodaj nową klasę do projektu, wprowadź **SchoolContext** nazwy klasy
- Zastęp zawartość nowej klasy, używając następującego kodu

```
Imports System.Data.Entity
Imports System.Data.Entity.Infrastructure
Imports System.Data.Entity.ModelConfiguration.Conventions
Imports System.ComponentModel.DataAnnotations
Imports System.ComponentModel.DataAnnotations.Schema

Public Class SchoolContext
    Inherits DbContext

    Public Property OfficeAssignments() As DbSet(Of OfficeAssignment)
    Public Property Instructors() As DbSet(Of Instructor)
    Public Property Courses() As DbSet(Of Course)
    Public Property Departments() As DbSet(Of Department)

    Protected Overrides Sub OnModelCreating(ByVal modelBuilder As DbModelBuilder)
        End Sub
    End Class
```

Konfigurowanie przy użyciu interfejsu API Fluent

W tej sekcji pokazano, jak skonfigurować typów tabel mapowania właściwości w celu mapowania kolumny i relacje między tabelami przy użyciu interfejsów API fluent\typu w modelu. Interfejs fluent API jest dostępna za pośrednictwem **DbModelBuilder** wpisz i najczęściej odbywa się przez zastąpienie **OnModelCreating** metody **DbContext**.

- Skopiuj poniższy kod i dodać go do **OnModelCreating** metody zdefiniowanej w **SchoolContext** klasy zostało wyjaśnione w komentarzach działanie każdego mapowania

```
' Configure Code First to ignore PluralizingTableName convention
```

```

' If you keep this convention then the generated tables
' will have pluralized names.
modelBuilder.Conventions.Remove(Of PluralizingTableNameConvention)()

' Specifying that a Class is a Complex Type

' The model defined in this topic defines a type OnsiteCourseDetails.
' By convention, a type that has no primary key specified
' is treated as a complex type.
' There are some scenarios where Code First will not
' detect a complex type (for example, if you do have a property
' called ID, but you do not mean for it to be a primary key).
' In such cases, you would use the fluent API to
' explicitly specify that a type is a complex type.
modelBuilder.ComplexType(Of OnsiteCourseDetails)()

' Mapping a CLR Entity Type to a Specific Table in the Database.

' All properties of OfficeAssignment will be mapped
' to columns in a table called t_OfficeAssignment.
modelBuilder.Entity(Of OfficeAssignment)().ToTable("t_OfficeAssignment")

' Mapping the Table-Per-Hierarchy (TPH) Inheritance

' In the TPH mapping scenario, all types in an inheritance hierarchy
' are mapped to a single table.
' A discriminator column is used to identify the type of each row.
' When creating your model with Code First,
' TPH is the default strategy for the types that
' participate in the inheritance hierarchy.
' By default, the discriminator column is added
' to the table with the name "Discriminator"
' and the CLR type name of each type in the hierarchy
' is used for the discriminator values.
' You can modify the default behavior by using the fluent API.
modelBuilder.Entity(Of Person)().
    Map(Of Person)(Function(t) t.Requires("Type").
        HasValue("Person")).
    Map(Of Instructor)(Function(t) t.Requires("Type").
        HasValue("Instructor"))

' Mapping the Table-Per-Type (TPT) Inheritance

' In the TPT mapping scenario, all types are mapped to individual tables.
' Properties that belong solely to a base type or derived type are stored
' in a table that maps to that type. Tables that map to derived types
' also store a foreign key that joins the derived table with the base table.
modelBuilder.Entity(Of Course)().ToTable("Course")
modelBuilder.Entity(Of OnsiteCourse)().ToTable("OnsiteCourse")
modelBuilder.Entity(Of OnlineCourse)().ToTable("OnlineCourse")

' Configuring a Primary Key

' If your class defines a property whose name is "ID" or "Id",
' or a class name followed by "ID" or "Id",
' the Entity Framework treats this property as a primary key by convention.
' If your property name does not follow this pattern, use the HasKey method
' to configure the primary key for the entity.
modelBuilder.Entity(Of OfficeAssignment)().
    HasKey(Function(t) t.InstructorID)

' Specifying the Maximum Length on a Property

```

```

' In the following example, the Name property
' should be no longer than 50 characters.
' If you make the value longer than 50 characters,
' you will get a DbEntityValidationException exception.
modelBuilder.Entity(Of Department)().Property(Function(t) t.Name).
    HasMaxLength(60)

' Configuring the Property to be Required

' In the following example, the Name property is required.
' If you do not specify the Name,
' you will get a DbEntityValidationException exception.
' The database column used to store this property will be non-nullable.
modelBuilder.Entity(Of Department)().Property(Function(t) t.Name).
    IsRequired()

' Switching off Identity for Numeric Primary Keys

' The following example sets the DepartmentID property to
' System.ComponentModel.DataAnnotations.DatabaseGeneratedOption.None to indicate that
' the value will not be generated by the database.
modelBuilder.Entity(Of Course)().Property(Function(t) t.CourseID).
    HasDatabaseGeneratedOption(DatabaseGeneratedOption.None)

'Specifying NOT to Map a CLR Property to a Column in the Database
modelBuilder.Entity(Of Department)().
    Ignore(Function(t) t.Administrator)

'Mapping a CLR Property to a Specific Column in the Database
modelBuilder.Entity(Of Department)().Property(Function(t) t.Budget).
    HasColumnName("DepartmentBudget")

'Configuring the Data Type of a Database Column
modelBuilder.Entity(Of Department)().Property(Function(t) t.Name).
    HasColumnType("varchar")

'Configuring Properties on a Complex Type
modelBuilder.Entity(Of OnsiteCourse)().Property(Function(t) t.Details.Days).
    HasColumnName("Days")
modelBuilder.Entity(Of OnsiteCourse)().Property(Function(t) t.Details.Location).
    HasColumnName("Location")
modelBuilder.Entity(Of OnsiteCourse)().Property(Function(t) t.Details.Time).
    HasColumnName("Time")

' Map one-to-zero or one relationship

' The OfficeAssignment has the InstructorID
' property that is a primary key and a foreign key.
modelBuilder.Entity(Of OfficeAssignment)().
    HasRequired(Function(t) t.Instructor).
    WithOptional(Function(t) t.OfficeAssignment)

' Configuring a Many-to-Many Relationship

' The following code configures a many-to-many relationship
' between the Course and Instructor types.
' In the following example, the default Code First conventions
' are used to create a join table.
' As a result the CourseInstructor table is created with
' Course_CourseID and Instructor_InstructorID columns.
modelBuilder.Entity(Of Course)().
    HasMany(Function(t) t.Instructors).
    WithMany(Function(t) t.Courses)

```

```

' Configuring a Many-to-Many Relationship and specifying the names
' of the columns in the join table

' If you want to specify the join table name
' and the names of the columns in the table
' you need to do additional configuration by using the Map method.
' The following code generates the CourseInstructor
' table with CourseID and InstructorID columns.

modelBuilder.Entity(Of Course)().
    HasMany(Function(t) t.Instructors).
    WithMany(Function(t) t.Courses).
    Map(Sub(m)
        m.ToTable("CourseInstructor")
        m.MapLeftKey("CourseID")
        m.MapRightKey("InstructorID")
    End Sub)

' Configuring a foreign key name that does not follow the Code First convention

' The foreign key property on the Course class is called DepartmentID_FK
' since that does not follow Code First conventions you need to explicitly specify
' that you want DepartmentID_FK to be the foreign key.

modelBuilder.Entity(Of Course)().
    HasRequired(Function(t) t.Department).
    WithMany(Function(t) t.Courses).
    HasForeignKey(Function(t) t.DepartmentID_FK)

' Enabling Cascade Delete

' By default, if a foreign key on the dependent entity is not nullable,
' then Code First sets cascade delete on the relationship.
' If a foreign key on the dependent entity is nullable,
' Code First does not set cascade delete on the relationship,
' and when the principal is deleted the foreign key will be set to null.
' The following code configures cascade delete on the relationship.

' You can also remove the cascade delete conventions by using:
' modelBuilder.Conventions.Remove<OneToManyCascadeDeleteConvention>()
' and modelBuilder.Conventions.Remove<ManyToManyCascadeDeleteConvention>()

modelBuilder.Entity(Of Course)().
    HasRequired(Function(t) t.Department).
    WithMany(Function(t) t.Courses).
    HasForeignKey(Function(d) d.DepartmentID_FK).
    WillCascadeOnDelete(False)

```

Przy użyciu modelu

Przeprowadźmy niektóre dane dostępu za pomocą **SchoolContext** wyświetlić się modelu w działaniu.

- Otwórz plik Module1.vb, w którym jest zdefiniowana funkcja Main
- Skopiuj i wklej następującą definicję Module1

```

Imports System.Data.Entity

Module Module1

Sub Main()

Using context As New SchoolContext()

    ' Create and save a new Department.
    Console.Write("Enter a name for a new Department: ")
    Dim name = Console.ReadLine()

    Dim department = New Department With { .Name = name, .StartDate = DateTime.Now }
    context.Departments.Add(department)
    context.SaveChanges()

    ' Display all Departments from the database ordered by name
    Dim departments =
        From d In context.Departments
        Order By d.Name
        Select d

    Console.WriteLine("All Departments in the database:")
    For Each department In departments
        Console.WriteLine(department.Name)
    Next

End Using

Console.WriteLine("Press any key to exit...")
Console.ReadKey()

End Sub

End Module

```

Możesz teraz uruchomić aplikację i ją przetestować.

```

Enter a name for a new Department: Computing
All Departments in the database:
Computing
Press any key to exit...

```

Pierwsze wstawienie kodu, aktualizowanie i usuwanie procedur składowanych

13.09.2018 • 12 minutes to read • [Edit Online](#)

NOTE

EF6 począwszy tylko — funkcje, interfejsów API itp. z opisem na tej stronie zostały wprowadzone w programie Entity Framework 6. Jeśli używasz starszej wersji, niektóre lub wszystkie informacje, nie ma zastosowania.

Domyślnie program Code First służy do konfigurowania wszystkich jednostek, aby wykonać Wstawianie, aktualizowanie i usuwanie poleceń przy użyciu bezpośredniego dostępu do tabel. Uruchamianie platformy EF6 można skonfigurować przez model Code First i używanie procedur składowanych w przypadku niektórych lub wszystkich jednostek w modelu.

Mapowania jednostki podstawowe

Możesz zdecydować się na korzystanie z procedur składowanych do wstawiania, aktualizacji i usunąć za pomocą interfejsu API Fluent.

```
modelBuilder  
    .Entity<Blog>()  
    .MapToStoredProcedures();
```

W ten sposób spowoduje, że Code First na potrzeby niektóre konwencje komplikacji oczekiwanej kształtuje procedury składowanych w bazie danych.

- Trzy procedury składowane o nazwie **<type_name>_Wstaw**, **<type_name>_aktualizuj** i **<type_name>_Usuń** (na przykład Blog_Insert i Blog_Update Blog_Delete).
- Nazwy parametrów odpowiadają nazwy właściwości.

NOTE

Jeśli używasz HasColumnName() lub atrybut kolumny można zmienić nazwy kolumny dla danej właściwości ta nazwa jest używana dla parametrów zamiast nazwy właściwości.

- Procedura składowana insert** będzie mieć parametr dla każdej właściwości, z wyjątkiem tych oznaczonych jako wygenerowane (tożsamość lub obliczona). Procedura składowana powinien zwrócić zestaw wyników z kolumny dla każdej właściwości wygenerowanej.
- Procedura składowana aktualizacji** będzie mieć parametr dla każdej właściwości, z wyjątkiem tych oznaczonych wzorcem wygenerowane "Obliczane". Niektóre tokeny współbieżności wymaga parametru dla oryginalnej wartości, zobacz *tokeny współbieżności* sekcji poniżej, aby uzyskać szczegółowe informacje. Procedura składowana powinien zwrócić zestaw wyników z kolumny dla każdej właściwości obliczanej.
- Usuń procedurę składowaną** powinien mieć parametr wartość klucza jednostki (lub wiele parametrów, jeśli jednostka ma klucz złożony). Ponadto procedury usuwania powinny mieć również parametry żadnych kluczy obcych skojarzenia niezależnie od tabeli docelowej (relacje, które nie mają odpowiednich właściwości klucza obcego zadeklarowane w jednostce). Niektóre tokeny współbieżności wymaga parametru dla oryginalnej wartości, zobacz *tokeny współbieżności* sekcji poniżej, aby uzyskać szczegółowe informacje.

Na przykład, przy użyciu następującej klasy:

```
public class Blog
{
    public int BlogId { get; set; }
    public string Name { get; set; }
    public string Url { get; set; }
}
```

Wartość domyślna, który będzie procedur składowanych:

```
CREATE PROCEDURE [dbo].[Blog_Insert]
    @Name nvarchar(max),
    @Url nvarchar(max)
AS
BEGIN
    INSERT INTO [dbo].[Blogs] ([Name], [Url])
    VALUES (@Name, @Url)

    SELECT SCOPE_IDENTITY() AS BlogId
END
CREATE PROCEDURE [dbo].[Blog_Update]
    @BlogId int,
    @Name nvarchar(max),
    @Url nvarchar(max)
AS
    UPDATE [dbo].[Blogs]
    SET [Name] = @Name, [Url] = @Url
    WHERE BlogId = @BlogId;
CREATE PROCEDURE [dbo].[Blog_Delete]
    @BlogId int
AS
    DELETE FROM [dbo].[Blogs]
    WHERE BlogId = @BlogId
```

Zastępowanie ustawień domyślnych

Można zastąpić część lub całość co zostało skonfigurowane domyślnie.

Można zmienić nazwę jednej lub kilku procedur składowanych. Ten przykład zmienia nazwę procedury składowanej aktualizacji tylko.

```
modelBuilder
    .Entity<Blog>()
    .MapToStoredProcedures(s =>
        s.Update(u => u.HasName("modify_blog")));

```

Ten przykład zmienia nazwę wszystkich trzech procedur składowanych.

```
modelBuilder
    .Entity<Blog>()
    .MapToStoredProcedures(s =>
        s.Update(u => u.HasName("modify_blog"))
        .Delete(d => d.HasName("delete_blog"))
        .Insert(i => i.HasName("insert_blog")));

```

W tych przykładach wywołania są ze sobą w sposób, ale możesz również użyć składni bloku lambda.

```

modelBuilder
    .Entity<Blog>()
    .MapToStoredProcedures(s =>
    {
        s.Update(u => u.HasName("modify_blog"));
        s.Delete(d => d.HasName("delete_blog"));
        s.Insert(i => i.HasName("insert_blog"));
    });

```

Ten przykład zmienia nazwę parametru dla właściwości BlogId na procedury składowanej aktualizacji.

```

modelBuilder
    .Entity<Blog>()
    .MapToStoredProcedures(s =>
        s.Update(u => u.Parameter(b => b.BlogId, "blog_id")));

```

Te wywołania są wszystkie chainable i konfigurowalna. Oto przykład, który zmienia nazwę wszystkich trzech procedur przechowywanych i ich parametrów.

```

modelBuilder
    .Entity<Blog>()
    .MapToStoredProcedures(s =>
        s.Update(u => u.HasName("modify_blog")
            .Parameter(b => b.BlogId, "blog_id")
            .Parameter(b => b.Name, "blog_name")
            .Parameter(b => b.Url, "blog_url"))
        .Delete(d => d.HasName("delete_blog")
            .Parameter(b => b.BlogId, "blog_id"))
        .Insert(i => i.HasName("insert_blog")
            .Parameter(b => b.Name, "blog_name")
            .Parameter(b => b.Url, "blog_url")));

```

Można również zmienić nazwy kolumn w zestawie wyników, zawierającą wartości z bazy danych, wygenerowane.

```

modelBuilder
    .Entity<Blog>()
    .MapToStoredProcedures(s =>
        s.Insert(i => i.Result(b => b.BlogId, "generated_blog_identity")));

```

```

CREATE PROCEDURE [dbo].[Blog_Insert]
    @Name nvarchar(max),
    @Url nvarchar(max)
AS
BEGIN
    INSERT INTO [dbo].[Blogs] ([Name], [Url])
    VALUES (@Name, @Url)

    SELECT SCOPE_IDENTITY() AS generated_blog_id
END

```

Relacje bez klucza obcego w klasie (niezależnie od skojarzeń)

Gdy właściwość klucza obcego jest uwzględniony w definicji klasy, odpowiedniego parametru można zmieniać nazwy w taki sam sposób jak inne właściwości. Gdy istnieje relacja bez właściwości klucza obcego w klasie, domyślna nazwa parametru jest **<navigation_property_name>_<primary_key_name>**.

Na przykład następujące definicje klas spowodowałyby parametr Blog_BlogId jest oczekiwany w procedurach

przechowywanych do wstawiania i aktualizowania wpisów.

```
public class Blog
{
    public int BlogId { get; set; }
    public string Name { get; set; }
    public string Url { get; set; }

    public List<Post> Posts { get; set; }
}

public class Post
{
    public int PostId { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }

    public Blog Blog { get; set; }
}
```

Zastępowanie ustawień domyślnych

Możesz zmienić parametry dla kluczy obcych, które nie są uwzględnione w klasie, podając ścieżkę do właściwość klucza podstawowego do parametru metody.

```
modelBuilder
    .Entity<Post>()
    .MapToStoredProcedures(s =>
        s.Insert(i => i.Parameter(p => p.Blog.BlogId, "blog_id"));
```

Jeśli nie masz właściwości nawigacji jednostki zależne (tj.) nie właściwości Post.Blog) możesz użyć metody skojarzenia, aby zidentyfikować drugiej stronie relacji, a następnie skonfigurować parametry, które odpowiadają każdej z właściwości kluczy.

```
modelBuilder
    .Entity<Post>()
    .MapToStoredProcedures(s =>
        s.Insert(i => i.Navigation<Blog>(
            b => b.Posts,
            c => c.Parameter(b => b.BlogId, "blog_id")));
```

Tokeny współbieżności

Aktualizowanie i usuwanie przechowywane procedury może być również konieczne przeciwdziałania współbieżności:

- Jeśli jednostka zawiera tokeny współbieżności, procedury składowanej mogą opcjonalnie mieć parametr wyjściowy, która zwraca liczbę wierszy, zaktualizowane lub usunięte (wierszy). Taki parametr musi być skonfigurowany przy użyciu metody RowsAffectedParameter.
Domyślnie EF używa wartość zwrotną z elementu ExecuteNonQuery, aby określić, ile wierszy została zmieniona. Określanie parametru wyjściowego odnośnych wierszy jest przydatne, jeśli po wykonaniu dowolnej logiki w swojej procedury sproc, które mogłyby spowodować wartość zwracaną ExecuteNonQuery jest niepoprawny (z perspektywy firmy EF) na końcu wykonywania.
- Dla każdego współbieżności będzie token ma parametr o nazwie **<property_name>_Original** (na przykład Timestamp_Original). To zostaną przekazane oryginalnej wartości tej właściwości – wartości po otrzymaniu kwerendy od bazy danych.
 - Tokeny współbieżności, które są obliczane przez bazy danych — takich jak sygnatury czasowe — będzie

miał tylko oryginalny parametru wartości.

- Obliczane inne niż właściwości, które są ustawione jako tokeny współbieżności Ponadto będziesz mieć parametr nową wartość w ramach procedury aktualizacji. Używa konwencji nazewnictwa już omówiono nowe wartości. Przykładem takiego tokenu będzie przy użyciu adresu URL blogu jako tokenem współbieżności, nowa wartość jest wymagana, ponieważ to mogą być aktualizowane na nową wartość w kodzie (w przeciwnieństwie do token sygnatury czasowej, które są aktualizowane tylko przez bazę danych).

Jest to przykład klasy i aktualizować procedury składowanej z tokenem współbieżności sygnatury czasowej.

```
public class Blog
{
    public int BlogId { get; set; }
    public string Name { get; set; }
    public string Url { get; set; }
    [Timestamp]
    public byte[] Timestamp { get; set; }
}
```

```
CREATE PROCEDURE [dbo].[Blog_Update]
    @BlogId int,
    @Name nvarchar(max),
    @Url nvarchar(max),
    @Timestamp_Original rowversion
AS
    UPDATE [dbo].[Blogs]
    SET [Name] = @Name, [Url] = @Url
    WHERE BlogId = @BlogId AND [Timestamp] = @Timestamp_Original
```

Oto przykład klasy i aktualizować procedury składowanej przy użyciu tokenu nieobliczaną współbieżności.

```
public class Blog
{
    public int BlogId { get; set; }
    public string Name { get; set; }
    [ConcurrencyCheck]
    public string Url { get; set; }
}
```

```
CREATE PROCEDURE [dbo].[Blog_Update]
    @BlogId int,
    @Name nvarchar(max),
    @Url nvarchar(max),
    @Url_Original nvarchar(max),
AS
    UPDATE [dbo].[Blogs]
    SET [Name] = @Name, [Url] = @Url
    WHERE BlogId = @BlogId AND [Url] = @Url_Original
```

Zastępowanie ustawień domyślnych

Opcjonalnie możesz wprowadzić parametr odnośnych wierszy.

```
modelBuilder
    .Entity<Blog>()
    .MapToStoredProcedures(s =>
        s.Update(u => u.RowsAffectedParameter("rows_affected")));
```

Tokeny współbieżności bazy danych jest obliczana — gdzie oryginalną wartość jest przekazywana — po prostu umożliwia parametr standardowe, zmiana nazwy mechanizm Zmień nazwę parametru, aby uzyskać oryginalną wartość.

```
modelBuilder
    .Entity<Blog>()
    .MapToStoredProcedures(s =>
        s.Update(u => u.Parameter(b => b.Timestamp, "blog_timestamp")));
```

Tokeny współbieżności nieobliczaną — gdzie zarówno oryginalne i nowe wartości są przekazywane — możesz użyć przeciążenia parametr, który pozwala podać nazwę dla każdego parametru.

```
modelBuilder
    .Entity<Blog>()
    .MapToStoredProcedures(s => s.Update(u => u.Parameter(b => b.Url, "blog_url", "blog_original_url")));
```

Wiele do wielu relacji

Użyjemy następujących klas jako przykład w tej sekcji.

```
public class Post
{
    public int PostId { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }

    public List<Tag> Tags { get; set; }
}

public class Tag
{
    public int TagId { get; set; }
    public string TagName { get; set; }

    public List<Post> Posts { get; set; }
}
```

Wiele do wielu relacji można mapować do procedur składowanych z następującą składnią.

```
modelBuilder
    .Entity<Post>()
    .HasMany(p => p.Tags)
    .WithMany(t => t.Posts)
    .MapToStoredProcedures();
```

Jeśli zostanie podana żadna inna konfiguracja domyślnie jest używany następujący kształt procedury składowanej.

- Dwie procedury składowane o nazwie **<type_one><type_two>_Wstaw** i **<type_one><type_two>_Usuń** (na przykład PostTag_Insert i PostTag_Delete).
- Parametry będą kluczowe wartości dla każdego typu. Nazwa każdego parametru jest **<type_name>_<property_name>** (na przykład Post_PostId i Tag_TagId).

Poniżej przedstawiono przykład wstawiania i aktualizowania procedur składowanych.

```
CREATE PROCEDURE [dbo].[PostTag_Insert]
    @Post_PostId int,
    @Tag_TagId int
AS
    INSERT INTO [dbo].[Post_Tags] (Post_PostId, Tag_TagId)
    VALUES (@Post_PostId, @Tag_TagId)
CREATE PROCEDURE [dbo].[PostTag_Delete]
    @Post_PostId int,
    @Tag_TagId int
AS
    DELETE FROM [dbo].[Post_Tags]
    WHERE Post_PostId = @Post_PostId AND Tag_TagId = @Tag_TagId
```

Zastępowanie ustawień domyślnych

Nazwy procedury i parametr można skonfigurować w sposób podobny do jednostki przechowywanej procedur.

```
modelBuilder
    .Entity<Post>()
    .HasMany(p => p.Tags)
    .WithMany(t => t.Posts)
    .MapToStoredProcedures(s =>
        s.Insert(i => i.HasName("add_post_tag")
            .LeftKeyParameter(p => p.PostId, "post_id")
            .RightKeyParameter(t => t.TagId, "tag_id"))
        .Delete(d => d.HasName("remove_post_tag")
            .LeftKeyParameter(p => p.PostId, "post_id")
            .RightKeyParameter(t => t.TagId, "tag_id")));
    );
```

Migracje Code First

29.09.2018 • 18 minutes to read • [Edit Online](#)

Migracje Code First jest zalecany sposób rozwijać schemat bazy danych aplikacji, jeśli używasz kodu pierwszego przepływu pracy. Migracje udostępniają zestaw narzędzi, które umożliwiają:

1. Tworzenie początkowej bazy danych, która współdziała z modelem platformy EF
2. Generowanie migracji, aby śledzić zmiany wprowadzone do modelu platformy EF
3. Bazy danych na bieżąco z tymi zmianami

Następujące Instruktaż zapewnia przegląd migracji Code First platformy Entity Framework. Możesz ukończenia całego instruktażu lub od razu przejść do tematu, który Cię interesuje. Omówiono następujące tematy:

Tworzenie początkowej modelu i bazy danych

Zanim zaczniemy, za pomocą migracji potrzeujemy projektu i modelu Code First chcesz pracować. W tym przewodniku są użyjemy canonical **Blog** i **wpis** modelu.

- Utwórz nową **MigrationsDemo** aplikacji konsoli
- Dodaj najnowszą wersję **EntityFramework** pakiet NuGet do projektu
 - **Narzędzia —> Menedżer pakietów biblioteki —> Konsola Menedżera pakietów**
 - Uruchom **EntityFramework instalacji pakietu** polecenia
- Dodaj **Model.cs** pliku z kodem, pokazano poniżej. Ten kod definiuje pojedynczy **Blog** klasę, która sprawia, że nasz model domeny i **BlogContext** klasę, która jest nasz kontekst EF Code First

```
using System.Data.Entity;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Data.Entity.Infrastructure;

namespace MigrationsDemo
{
    public class BlogContext : DbContext
    {
        public DbSet<Blog> Blogs { get; set; }
    }

    public class Blog
    {
        public int BlogId { get; set; }
        public string Name { get; set; }
    }
}
```

- Teraz, gdy model, nadszedł czas na jej używać do wykonywania dostępu do danych. Aktualizacja **Program.cs** pliku z kodem, pokazano poniżej.

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

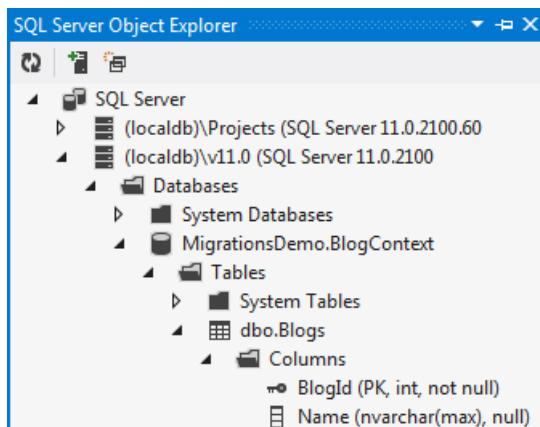
namespace MigrationsDemo
{
    class Program
    {
        static void Main(string[] args)
        {
            using (var db = new BlogContext())
            {
                db.Blogs.Add(new Blog { Name = "Another Blog" });
                db.SaveChanges();

                foreach (var blog in db.Blogs)
                {
                    Console.WriteLine(blog.Name);
                }
            }

            Console.WriteLine("Press any key to exit...");
            Console.ReadKey();
        }
    }
}

```

- Uruchom aplikację i zostanie wyświetlony **MigrationsCodeDemo.BlogContext** baza danych została utworzona dla Ciebie.



Włączanie migracji

Nadszedł czas na pewnych zmian więcej w naszym modelu.

- Umożliwia wprowadzenie właściwość adres Url do klasy blogu.

```
public string Url { get; set; }
```

Jeśli zamierzasz uruchomić aplikację ponownie otrzymamy InvalidOperationException, podając *modelu kopii kontekstu "BlogContext"* została zmieniona od czasu utworzenia bazy danych. Należy rozważyć użycie migracje Code First w aktualizacji bazy danych (<http://go.microsoft.com/fwlink/?LinkId=238269>).

Jak sugeruje wyjątek, jest czas, aby rozpocząć korzystanie z migracje Code First. Pierwszym krokiem jest włączanie funkcji migracje dla nasz kontekst.

- Uruchom **Enable-Migrations** polecenia w konsoli Menedżera pakietów

To polecenie został dodany **migracje** folderu do naszego projektu. Ten nowy folder zawiera dwa pliki:

- **Klasa konfiguracji.** Ta klasa pozwala skonfigurować sposób działania migracji w Twoim kontekście. W tym przewodniku po prostu używamy domyślnej konfiguracji. *Ponieważ istnieje tylko pojedynczy kontekst Code First w projekcie, Enable-Migrations jest wypełniane automatycznie typ kontekstu, którego dotyczy ta konfiguracja.*
- **Migracja InitialCreate.** Ta migracja została wygenerowany, ponieważ już mieliśmy Code First utworzyć bazę danych, zanim umożliwiliśmy migracji. Kod w tej migracji szkieletu reprezentuje obiektów, które zostały już utworzone w bazie danych. W tym przypadku, który jest **Blog** tabeli za pomocą **BlogId** i **nazwa** kolumn. Nazwa pliku zawiera sygnaturę czasową, aby pomóc w kolejności. *Jeśli baza danych nie została już utworzony tej migracji InitialCreate czy nie zostały dodane do projektu. Zamiast tego po raz pierwszy nazywamy migracji Dodaj kod, aby utworzyć te tabele będą szkieletu do nowej migracji.*

Wiele modeli przeznaczonych dla tej samej bazy danych

W przypadku używania wersji przed EF6, tylko jeden model Code First może służyć do generowania/Zarządzanie schematami bazy danych. Jest to wynikiem pojedynczej **_MigrationsHistory** tabeli na bazę danych w sposób identyfikowania zapisy, które należą do modelu.

Począwszy od platformy EF6, **konfiguracji** klasa zawiera **elementu ContextKey** właściwości. Jest to zabezpieczenie jako unikatowy identyfikator dla każdego modelu Code First. W kolumnie **_MigrationsHistory** tabeli umożliwia wpisy z wielu modeli, aby udostępnić w tabeli. Ta właściwość jest domyślnie do w pełni kwalifikowanej nazwy kontekstu.

Generowanie i uruchamianie migracji

Migracje Code First ma dwa podstawowe polecenia, które ma być zapoznanie się z.

- **Dodaj migracji** będzie tworzenia szkieletu dalej migracji, w oparciu o zmiany wprowadzone do modelu od chwili utworzenia ostatniej migracji
- **Update-Database** Zastosuj wszelkie oczekujące migracji do bazy danych

Potrzebujemy do tworzenia szkieletu migracji, aby zadbać o nową właściwość adres Url, które dodaliśmy.

Migracji Dodaj polecenie umożliwia nam nazwij te migracji, po prostu nazwiemy naszych **AddBlogUrl**.

- Uruchom **AddBlogUrl migracji Dodaj** polecenia w konsoli Menedżera pakietów
- W **migracje** folderu w efekcie powstał nowy **AddBlogUrl** migracji. Filename migracji wstępnie ustalony z sygnaturą czasową pomagające w kolejności

```
namespace MigrationsDemo.Migrations
{
    using System;
    using System.Data.Entity.Migrations;

    public partial class AddBlogUrl : DbMigration
    {
        public override void Up()
        {
            AddColumn("dbo.Blogs", "Url", c => c.String());
        }

        public override void Down()
        {
            DropColumn("dbo.Blogs", "Url");
        }
    }
}
```

Firma Microsoft, można teraz edytować lub dodać do tej migracji, ale wszystko wygląda dość dobrze. Użyjmy **Update-Database** dotyczącej tej migracji bazy danych.

- Uruchom **Update-Database** polecenia w konsoli Menedżera pakietów
- Migracje Code First zostanie porównane migracji w naszym **migracje** folder z tymi, które zostały zastosowane do bazy danych. Zostanie on wyświetlony **AddBlogUrl** migracji musi być zastosowane i uruchom go.

MigrationsDemo.BlogContext bazy danych został zaktualizowany do uwzględnienia **adresu Url** kolumny w **blogi** tabeli.

Dostosowywanie migracji

Do tej pory mamy już i uruchamianie migracji bez wprowadzania żadnych zmian. Teraz Przyjrzyjmy się edycją kodu, który pobiera wygenerowany domyślnie.

- Nadszedł czas, aby wprowadzić pewne zmiany więcej w naszym modelu, możemy dodać nową **ocena** właściwości **blogu** klasy

```
public int Rating { get; set; }
```

- Możemy również dodać nowy **wpis** klasy

```
public class Post
{
    public int PostId { get; set; }
    [MaxLength(200)]
    public string Title { get; set; }
    public string Content { get; set; }

    public int BlogId { get; set; }
    public Blog Blog { get; set; }
}
```

- Dodamy również **wpisy** kolekcji **Blog** klasy w celu utworzenia drugiej stronnej relacji między **Blog** i **wpis**

```
public virtual List<Post> Posts { get; set; }
```

Użyjemy **migracji Dodaj** polecenie, aby umożliwić migracje Code First tworzenia szkieletu jego najbardziej prawdopodobny wybór na migrację dla nas. Zamierzamy wywołania tej migracji **AddPostClass**.

- Uruchom **AddPostClass migracji Dodaj** polecenia w konsoli Menedżera pakietów.

Migracje Code First jak całkiem Niezłe rozwiązań zadanie tworzenia szkieletów te zmiany, ale istnieje kilka kwestii, które firma Microsoft może wystąpić potrzeba zmiany:

1. Najpierw w góre, możemy dodać unikatowego indeksu **Posts.Title** kolumny (Dodawanie w wierszu 22 & 29 w poniższym kodzie).
2. Dodajemy również dopuszcza **Blogs.Rating** kolumny. W przypadku istniejących danych w tabeli zostaną przypisane domyślne CLR typu danych dla nowej kolumny (ocena jest liczba całkowita, tak byłoby **0**). Ale chcemy określić wartość domyślną **3** więc w istniejącym wiersze **blogi** tabeli rozpoczyna się od klasyfikacji znośnego. (Wartość domyślna określona w wierszu 24 kod poniżej można znaleźć)

```

namespace MigrationsDemo.Migrations
{
    using System;
    using System.Data.Entity.Migrations;

    public partial class AddPostClass : DbMigration
    {
        public override void Up()
        {
            CreateTable(
                "dbo.Posts",
                c => new
                {
                    PostId = c.Int(nullable: false, identity: true),
                    Title = c.String(maxLength: 200),
                    Content = c.String(),
                    BlogId = c.Int(nullable: false),
                })
                .PrimaryKey(t => t.PostId)
                .ForeignKey("dbo.Blogs", t => t.BlogId, cascadeDelete: true)
                .Index(t => t.BlogId)
                .Index(p => p.Title, unique: true);

            AddColumn("dbo.Blogs", "Rating", c => c.Int(nullable: false, defaultValue: 3));
        }

        public override void Down()
        {
            DropIndex("dbo.Posts", new[] { "Title" });
            DropIndex("dbo.Posts", new[] { "BlogId" });
            DropForeignKey("dbo.Posts", "BlogId", "dbo.Blogs");
            DropColumn("dbo.Blogs", "Rating");
            DropTable("dbo.Posts");
        }
    }
}

```

Nasze edytowanych migracji jest gotowy do Przejdz, dlatego użyjemy **Update-Database** Aby przełączyć bazę danych aktualne. Teraz możemy określić — **pełne** Flaga, dzięki czemu możesz zobaczyć SQL Server, który działa migracje Code First.

- Uruchom **Update-Database — pełne** polecenia w konsoli Menedżera pakietów.

Ruchu danych niestandardowymi SQL

Do tej pory mają przyjrzały się migracji, które operacje, które nie zmieniać i przenieść wszystkie dane teraz możemy przyjrzeć się coś wymagających przenoszenia niektórych danych. Istnieje jeszcze nie natywną obsługę ruchu danych, ale niektóre dowolnych poleceń SQL w dowolnym momencie można uruchomić w naszego skryptu.

- Dodajmy **Post.Abstract** właściwość nasz model. Później, będziemy do wstępnego wypełniania **abstrakcyjne** dla istniejących wpisów od początku przy użyciu jakiś tekst **zawartości** kolumny.

```
public string Abstract { get; set; }
```

Użyjemy **migracji Dodaj** polecenie, aby umożliwić migracje Code First tworzenia szkieletu jego najbardziej prawdopodobny wybór naację dla nas.

- Uruchom **AddPostAbstract migracji Dodaj** polecenia w konsoli Menedżera pakietów.
- Wygenerowany migracji zajmie się zmiany schematu, ale chcemy również do wstępniego wypełniania

abstrakcyjne kolumny przy użyciu 100 pierwszych znaków zawartości dla każdego wpisu. Możemy to zrobić przez rozwijanej opuszczą SQL i uruchamianie **aktualizacji** instrukcji po dodaniu kolumny. (Dodanie w wierszu 12 w poniższym kodzie)

```
namespace MigrationsDemo.Migrations
{
    using System;
    using System.Data.Entity.Migrations;

    public partial class AddPostAbstract : DbMigration
    {
        public override void Up()
        {
            AddColumn("dbo.Posts", "Abstract", c => c.String());

            Sql("UPDATE dbo.Posts SET Abstract = LEFT(Content, 100) WHERE Abstract IS NULL");
        }

        public override void Down()
        {
            DropColumn("dbo.Posts", "Abstract");
        }
    }
}
```

Nasze edytowanych migracji jest dobra, dlatego użyjemy **Update-Database** Aby przełączyć bazę danych aktualne. Firma Microsoft będzie określić — **pełne** Flaga, dzięki czemu możemy zobaczyć SQL są uruchamiane w bazie danych.

- Uruchom **Update-Database — pełne** polecenia w konsoli Menedżera pakietów.

Migrowanie do określonej wersji (w tym obniżenia poziomu)

Do tej pory zawsze aktualniliśmy do najnowszych migracji, ale mogą zaistnieć sytuacje, kiedy zechcesz, aktualnianie i obniżanie wersji do migracji.

Załóżmy, że chcemy przeprowadzić migrację naszych bazy danych do stanu sprzed po uruchomieniu naszych **AddBlogUrl** migracji. Możemy użyć — **TargetMigration** przełącznika na starszą wersję tej migracji.

- Uruchom **Update-Database-TargetMigration: AddBlogUrl** polecenia w konsoli Menedżera pakietów.

To polecenie uruchomi skrypt w dół dla naszych **AddBlogAbstract** i **AddPostClass** migracji.

Jeśli chcesz przeprowadzić wdrożenie aż z powrotem do pustej bazy danych, wówczas można użyć **Update-Database-TargetMigration: \$InitialDatabase** polecenia.

Pobieranie skryptu SQL

Jeśli inny Deweloper chce tych zmian na swoich maszynach one po prostu synchronizacji po sprawdzenie zmian w systemie kontroli źródła. Po naszej nowej migracji one po prostu uruchomić polecenie Update-Database, aby zmiany zastosowana lokalnie. Jednak jeśli chcemy wdrożyć tych zmian na serwerze testowym i po pewnym czasie produkcji, prawdopodobnie chcemy skrypt SQL, które firma Microsoft może przekazują do naszych DBA.

- Uruchom **Update-Database** polecenia, ale tym razem określ — **skrypt** Flaga tak, aby zmiany są zapisywane do skryptu zamiast stosowane. Firma Microsoft będzie również określić źródło i cel migracji można wygenerować skryptu dla. Chcemy, aby skrypt, aby przejść od pustej bazy danych (**\$InitialDatabase**) do najnowszej wersji (migracja **AddPostAbstract**). Jeśli nie określiłeś migracji docelowego migracji użyje najnowszych migracji jako element docelowy. Jeśli nie określiłeś migracje źródła migracje użyje bieżący stan bazy danych.

- Uruchom **Update-Database-Script - SourceMigration: \$InitialDatabase - TargetMigration:**
AddPostAbstract polecenia w konsoli Menedżera pakietów

Migracje Code First uruchomi proces migracji, ale zamiast faktycznie stosowania zmian go będzie zapisać je do pliku SQL dla Ciebie. Wygenerowany skrypt jest otwierany automatycznie w programie Visual Studio gotowe wyświetlić lub zapisać.

Generowanie skryptów Idempotentne

Począwszy od platformy EF6, jeśli określisz — **SourceMigration \$InitialDatabase** wygenerowany skrypt będzie "idempotentne". Skrypty Idempotentne można uaktualnić bazy danych obecnie w dowolnej wersji do najnowszej wersji (lub określonej wersji, jeśli używasz — **TargetMigration**). Wygenerowany skrypt zawiera logikę do sprawdzenia **_MigrationsHistory** tabeli i tylko zastosować zmiany, które nie zostały zastosowane wcześniej.

Automatycznie uaktualnianie przy uruchamianiu aplikacji (inicjatorze **MigrateDatabaseToLatestVersion**)

Jeśli aplikacja jest wdrażana można ją automatycznie uaktualnić bazy danych (stosując wszystkie oczekujące migracje) po uruchomieniu aplikacji. Można to zrobić, rejestrując **MigrateDatabaseToLatestVersion** inicjatora bazy danych. Inicjator bazy danych zawiera po prostu logikę, który służy do upewnić się, że baza danych jest prawidłowo skonfigurowana. Logika jest uruchamiany po raz pierwszy kontekstu jest używana w ramach procesu aplikacji (**AppDomain**).

Firma Microsoft może aktualizować **Program.cs** pliku, jak pokazano poniżej, aby ustawić **MigrateDatabaseToLatestVersion** inicjator dla BlogContext, zanim użyjemy kontekstu (wiersz 14). Należy pamiętać, że trzeba będzie również dodać za pomocą instrukcji `for System.Data.Entity` przestrzeni nazw (wiersz 5).

*Kiedy tworzymy wystąpienie tego inicjatora, należy określić typ kontekstu (**BlogContext**) i konfiguracji migracji (**konfiguracji**) — Konfiguracja migracji jest klasa, która otrzymała dodany do naszych **migracje** folderu podczas umożliwiliśmy migracji.*

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data.Entity;
using MigrationsDemo.Migrations;

namespace MigrationsDemo
{
    class Program
    {
        static void Main(string[] args)
        {
            Database.SetInitializer(new MigrateDatabaseToLatestVersion<BlogContext, Configuration>());

            using (var db = new BlogContext())
            {
                db.Blogs.Add(new Blog { Name = "Another Blog" });
                db.SaveChanges();

                foreach (var blog in db.Blogs)
                {
                    Console.WriteLine(blog.Name);
                }
            }

            Console.WriteLine("Press any key to exit...");
            Console.ReadKey();
        }
    }
}
```

Teraz zawsze, gdy nasze działania aplikacji, najpierw będzie sprawdzał, jeśli baza danych jego celem jest element jest aktualny i Zastosuj wszelkie oczekujące migracji, jeśli nie jest.

Migracje automatyczne Code First

27.09.2018 • 10 minutes to read • [Edit Online](#)

Automatycznej migracji pozwala użyć migracje Code First bez potrzeby plik kodu dla każdej zmiany wprowadzone w projekcie. Nie wszystkie zmiany można zastosować automatycznie — na przykład zmienia nazwę kolumny korzystającą z migracji za pomocą kodu.

NOTE

W tym artykule przyjęto założenie, że wiesz, jak użyć migracje Code First w podstawowych scenariuszy. Jeśli tego nie zrobisz, a następnie należy odczytać [migracje Code First](#) przed kontynuowaniem.

Zalecenia dotyczące środowiska zespołowe

Możesz rozdzielać migracje w architekturze kodu i automatyczne, ale nie jest to zalecane w scenariuszach tworzenia zespołu. Jeśli jesteś częścią zespołu deweloperów korzystających z kontroli źródła albo należy używać wyłącznie automatycznej migracji lub migracje oparte wyłącznie na kodzie. Biorąc pod uwagę ograniczenia na potrzeby automatycznej migracji firma Microsoft zaleca, za pomocą migracje oparte na kodzie w środowiskach zespołu.

Tworzenie początkowej modelu i bazy danych

Zanim zaczniemy, za pomocą migracji potrzebujemy projektu i model Code First chcesz pracować. W tym przewodniku są użyjemy canonical **Blog** i **wpis** modelu.

- Utwórz nową **MigrationsAutomaticDemo** aplikacji konsoli
- Dodaj najnowszą wersję **EntityFramework** pakiet NuGet do projektu
 - **Narzędzia** —> **Menedżer pakietów biblioteki** —> **Konsola Menedżera pakietów**
 - Uruchom **EntityFramework instalacji pakietu** polecenia
- Dodaj **Model.cs** pliku z kodem, pokazano poniżej. Ten kod definiuje pojedynczy **Blog** klasę, która sprawia, że nasz model domeny i **BlogContext** klasę, która jest nasz kontekst EF Code First

```
using System.Data.Entity;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Data.Entity.Infrastructure;

namespace MigrationsAutomaticDemo
{
    public class BlogContext : DbContext
    {
        public DbSet<Blog> Blogs { get; set; }
    }

    public class Blog
    {
        public int BlogId { get; set; }
        public string Name { get; set; }
    }
}
```

- Teraz, gdy model, nadszedł czas na jej używać do wykonywania dostępu do danych. Aktualizacja **Program.cs**

pliku z kodem, pokazano poniżej.

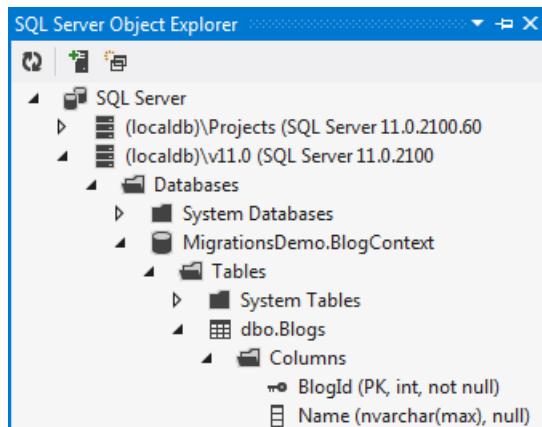
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace MigrationsAutomaticDemo
{
    class Program
    {
        static void Main(string[] args)
        {
            using (var db = new BlogContext())
            {
                db.Blogs.Add(new Blog { Name = "Another Blog" });
                db.SaveChanges();

                foreach (var blog in db.Blogs)
                {
                    Console.WriteLine(blog.Name);
                }
            }

            Console.WriteLine("Press any key to exit...");
            Console.ReadKey();
        }
    }
}
```

- Uruchom aplikację i zostanie wyświetlony **.MigrationsAutomaticCodeDemo.BlogContext** baza danych została utworzona dla Ciebie.



Włączanie migracji

Nadszedł czas na pewnych zmian więcej w naszym modelu.

- Umożliwia wprowadzenie właściwość adres Url do klasy blogu.

```
public string Url { get; set; }
```

Jeśli zamierzasz uruchomić aplikację ponownie otrzymamy InvalidOperationException, podając *modelu kopii kontekstu "BlogContext" została zmieniona od czasu utworzenia bazy danych. Należy rozważyć użycie migracje Code First w aktualizacji bazy danych* (<http://go.microsoft.com/fwlink/?LinkId=238269>).

Jak sugeruje wyjątek, jest czas, aby rozpocząć korzystanie z migracji Code First. Ponieważ chcemy użyć migracje automatyczne zamierzamy określi — **EnableAutomaticMigrations** przełącznika.

- Uruchom **Enable-Migrations** — **EnableAutomaticMigrations** polecenia w poleceniu ten konsoli Menedżera pakietów została dodana **migracje** folderu do naszego projektu. Ten nowy folder zawiera jeden plik:
- Klasa konfiguracji.** Ta klasa pozwala skonfigurować sposób działania migracji w Twoim kontekście. W tym przewodniku po prostu używamy domyślnej konfiguracji. *Ponieważ istnieje tylko pojedynczy kontekst Code First w projekcie, Enable-Migrations jest wypełniane automatycznie typ kontekstu, którego dotyczy ta konfiguracja.*

Pierwszy automatycznej migracji

Migracje Code First ma dwa podstawowe polecenia, które ma być zapoznanie się z.

- Dodaj migracji** będzie tworzenia szkieletu dalej migracji, w oparciu o zmiany wprowadzone do modelu od chwili utworzenia ostatniej migracji
- Update-Database** Zastosuj wszelkie oczekujące migracji do bazy danych

Firma Microsoft zamierza uniknąć za pomocą Dodaj migracji (chyba że to naprawdę musimy) i skoncentrować się na umożliwienie migracje Code First automatycznie obliczyć i zastosować zmiany. Użyjmy **Update-Database** można pobrać migracje Code First, aby wypchnąć zmiany do nasz model (nowy **Blog.Urwashiowość I**) w bazie danych.

- Uruchom **Update-Database** polecenia w konsoli Menedżera pakietów.

MigrationsAutomaticDemo.BlogContext bazy danych został zaktualizowany do uwzględnienia **adresu Url** kolumny w **blogi** tabeli.

Drugi automatycznej migracji

Upewnijmy się, innej zmiany i pozwól migracje Code First automatycznie wypychać zmiany do bazy danych dla nas.

- Możemy również dodać nowy **wpis** klasy

```
public class Post
{
    public int PostId { get; set; }
    [MaxLength(200)]
    public string Title { get; set; }
    public string Content { get; set; }

    public int BlogId { get; set; }
    public Blog Blog { get; set; }
}
```

- Dodamy również **wpisy** kolekcji **Blog** klasy w celu utworzenia drugiej stronie relacji między **Blog** i **wpis**

```
public virtual List<Post> Posts { get; set; }
```

Teraz za pomocą **Update-Database** Aby przełączyć bazę danych aktualne. Teraz możemy określić — **pełne** Flaga, dzięki czemu możesz zobaczyć SQL Server, który działa migracje Code First.

- Uruchom **Update-Database** — **pełne** polecenia w konsoli Menedżera pakietów.

Dodawanie kodu na podstawie migracji

Teraz Przyjrzymy się coś, co firma Microsoft może być konieczne użycie oparty na kodzie migracja.

- Dodajmy **ocena** właściwości **blogu** klasy

```
public int Rating { get; set; }
```

Firma Microsoft może po prostu uruchomić **Update-Database** do wypychania tych zmian w bazie danych. Jednak dodajemy dopuszcza **Blogs.Rating** kolumny, w przypadku istniejących danych w tabeli zostaną zostaną przypisane domyślne CLR typu danych dla nowej kolumny (ocena jest liczba całkowita, tak byłoby **0**). Ale chcemy określić wartość domyślną **3** więc w istniejącym wiersze **blogi** tabeli rozpoczyna się od klasyfikacji znośnego. Użyjemy polecenia Add-migracji można zapisać tej zmiany, które się do migracji za pomocą kodu, dzięki czemu możemy poddać edycji. **Migracji Dodaj** polecenie umożliwia nam nazwij te migracji, po prostu nazwiemy naszych **AddBlogRating**.

- Uruchom **AddBlogRating migracji Dodaj** polecenia w konsoli Menedżera pakietów.
- W **migracje** folderu w efekcie powstał nowy **AddBlogRating** migracji. Nazwa pliku migracji jest wstępnie stała z sygnaturą czasową pomagającą w kolejności. Umożliwia edytowanie wygenerowany kod, aby określić domyślną wartość 3 dla Blog.Rating (wiersz 10 w poniższym kodzie)

Migracji ma również plik związany z kodem, który przechwytuje niektórych metadanych. Te metadane umożliwi migracje Code First replikowanie automatycznej migracji, możemy wykonać przed migracją to oparty na kodzie. Jest to ważne, jeśli inny Deweloper chce Uruchom migracje naszych lub gdy nadziej się czas wdrażania naszej aplikacji.

```
namespace MigrationsAutomaticDemo.Migrations
{
    using System;
    using System.Data.Entity.Migrations;

    public partial class AddBlogRating : DbMigration
    {
        public override void Up()
        {
            AddColumn("Blogs", "Rating", c => c.Int(nullable: false, defaultValue: 3));
        }

        public override void Down()
        {
            DropColumn("Blogs", "Rating");
        }
    }
}
```

Nasze edytowanych migracji jest dobra, dlatego użyjemy **Update-Database** Aby przełączyć bazę danych aktualne.

- Uruchom **Update-Database** polecenia w konsoli Menedżera pakietów.

Powrót do automatycznej migracji

Jesteśmy teraz bezpłatnych wrócić do automatycznej migracji dla prostsze zmian. Migracje Code First zajmie się przeprowadzania migracji automatycznej i oparte na kodzie w odpowiedniej kolejności, na podstawie metadanych, która jest przechowywana w pliku związanym z kodem dla każdego migracji opartego na kodzie.

- Dodajmy właściwość Post.Abstract naszym modelu

```
public string Abstract { get; set; }
```

Teraz możemy użyć **Update-Database** można pobrać migracje Code First wypychania tej zmiany do bazy danych za pomocą automatycznej migracji.

- Uruchom **Update-Database** polecenia w konsoli Menedżera pakietów.

Podsumowanie

W tym przewodniku pokazano, jak użyć migracje automatycznego wypychania modelu zmienia się z bazą danych. Przedstawiono także sposób tworzenia szkieletu i uruchamiania oparte na kodzie migracje między automatycznej migracji, gdy potrzebujesz większej kontroli.

Migracje Code First przy użyciu istniejącej bazy danych

13.09.2018 • 14 minutes to read • [Edit Online](#)

NOTE

EF4.3 począwszy tylko — funkcje, interfejsów API itp. z opisem na tej stronie zostały wprowadzone w programie Entity Framework 4.1. Jeśli używasz starszej wersji, niektóre lub wszystkie informacje, nie ma zastosowania.

W tym artykule opisano, migracje Code First przy użyciu istniejącej bazy danych, który nie został utworzony przez program Entity Framework.

NOTE

W tym artykule przyjęto założenie, że wiesz, jak użyć migracje Code First w podstawowych scenariuszy. Jeśli tego nie zrobisz, a następnie należy odczytać [migracje Code First](#) przed kontynuowaniem.

Zrzuty ekranu

Jeśli zrzut ekranu niż przeczytaj ten artykuł będzie wolisz obejrzeć, następujące dwa filmy wideo obejmuje tę samą zawartość, jak w tym artykule.

Wideo 1: "Migracje - Under the Hood"

[Ten zrzut ekranu](#) opisano sposób migracji śledzi i używa informacji o modelu, aby wykrywać zmiany modelu.

Dwa wideo: "Migracje - istniejących baz danych"

Opierając się na koncepcji z poprzednim filmu wideo, [ten zrzut ekranu](#) opisano, jak włączyć i użyć migracje z istniejącej bazy danych.

Krok 1: Tworzenie modelu

Pierwszym krokiem będzie utworzenie model Code First, który jest przeznaczony dla istniejącej bazy danych. [Code First istniejącą bazę danych](#) temat zawiera szczegółowe wskazówki, jak to zrobić.

NOTE

Należy wykonać pozostałe kroki w tym temacie przed wprowadzeniem jakichkolwiek zmian, które wymagałyby zmiany schematu bazy danych modelu. Poniższe kroki wymagają modelu, który ma zostać zsynchronizowana ze schematem bazy danych.

Krok 2: Włączanie funkcji migracje

Następnym krokiem jest włączanie funkcji migracje. Można to zrobić, uruchamiając **Enable-Migrations** polecenia w konsoli Menedżera pakietów.

To polecenie Utwórz folder w rozwiążaniu o nazwie migracje i umieścić jedną klasę wewnątrz niego nazywaną konfiguracją. Klasa konfiguracji jest konfigurowania migracji dla aplikacji, możesz dowiedzieć się więcej na temat [migracje Code First](#) tematu.

Krok 3. Dodawanie początkowej migracji

Po migracji zostały utworzone i zastosować do lokalnej bazy danych, można także zastosować je zmieni się z innymi bazami danych. Na przykład lokalna baza danych może być test bazy danych i ostatecznie może chcesz również zastosować zmiany do produkcyjnej bazy danych i/lub innym deweloperom testowanie bazy danych. Dostępne są dwie opcje dla tego kroku i zależy od tego, który należy wybrać, czy schematów innych baz danych jest pusta lub obecnie jest zgodny ze schematem lokalnej bazy danych.

- **Opcja jeden: Użyj istniejącego schematu jako punkt początkowy.** To podejście należy używać, gdy innych baz danych, które migracje będą dotyczyć w przyszłości będzie miał ten sam schemat, jak lokalna baza danych ma obecnie. Na przykład można użyć, to gdy bazy danych lokalnego testu jest obecnie zgodna v1 produkcyjnej bazy danych, a później będą stosowane te migracji, aby zaktualizować produkcyjnej bazy danych do wersji 2.
- **Opcja 2: Użyj pustej bazy danych jako punkt początkowy.** To podejście należy używać, gdy innych baz danych, które migracje będą dotyczyć w przyszłości są puste (lub jeszcze nie istnieją). Na przykład można użyć, to gdy Rozpocznij tworzenie aplikacji swoją aplikację przy użyciu bazy danych testu, ale bez użycia migracji i można będzie później chcesz utworzyć produkcyjnej bazy danych od podstaw.

Opcja jeden: Użyj istniejącego schematu jako punktu wyjścia

Migracje Code First używa migawkę modelu przechowywane w najnowszych migracji do wykrywania zmian do modelu (można znaleźć szczegółowe informacje o tym w [migracje Code First na środowiska zespółowe](#)). Ponieważ firma Microsoft zamierza przyjęto założenie, że bazy danych istnieje już schemat bieżącego modelu, wygenerujemy pusty migracji (pusta), które jest bieżący model jako migawka.

1. Uruchom **InitialCreate migracji Dodaj — IgnoreChanges** polecenia w konsoli Menedżera pakietów. Spowoduje to utworzenie pustego migracji w obecnym modelu jako migawka.
2. Uruchom **Update-Database** polecenia w konsoli Menedżera pakietów. Migracja InitialCreate zostaną zastosowane do bazy danych. Ponieważ rzeczywistą migrację nie zawiera żadnych zmian, po prostu spowoduje dodanie wiersza do `_MigrationsHistory` tabeli wskazujący, że migracja została już zainstalowana.

Opcja 2: Użyj pustej bazy danych jako punktu wyjścia

W tym scenariuszu należy migracji, aby można było utworzyć całą bazę danych od podstaw — łącznie z tabelami, które już znajdują się w naszym lokalnej bazy danych. Zamierzamy Wygeneruj migrację InitialCreate, który zawiera logikę w celu utworzenia istniejącego schematu. Udoskonalimy naszej istniejącej bazy danych, wyglądają tak, ta migracja została już zainstalowana.

1. Uruchom **InitialCreate migracji Dodaj** polecenia w konsoli Menedżera pakietów. Spowoduje to utworzenie migracji, aby utworzyć istniejącego schematu.
2. Komentarz cały kod w metodzie w góre nowo utworzony migracji. Pozwoli to nam na element "apply" migracji w lokalnej bazie danych bez próby odtworzyć wszystkie tabele itp., które już istnieją.
3. Uruchom **Update-Database** polecenia w konsoli Menedżera pakietów. Migracja InitialCreate zostaną zastosowane do bazy danych. Ponieważ nie zawiera rzeczywistą migrację zmiany (ponieważ możemy tymczasowo komentarzem nalepek), po prostu spowoduje dodanie wiersza do `_MigrationsHistory` tabeli wskazujący, że migracja została już zainstalowana.
4. ONZ komentarz kod w metodzie w góre. Oznacza to, że podczas tej migracji jest stosowany do przyszłych baz danych, schemat, który istniał już w lokalnej bazie danych zostanie utworzona przez migracje.

Co należy wiedzieć o

Istnieje kilka rzeczy, które musisz wiedzieć, korzystając z migracji z istniejącej bazy danych.

Domyślne/obliczane nazwy nie może odpowiadać istniejącego schematu

Migracje jawnie określa nazwy kolumn i tabel, gdy jej scaffolds migracji. Istnieją jednak inne migracje oblicza domyślnej nazwy dla podczas stosowania migracje obiekty bazy danych. Obejmuje to indeksy i ograniczenia klucza obcego. Podczas określania wartości schematu, te nazwy obliczeniowe może nie odpowiadać, co faktycznie

istnieje w bazie danych.

Gdy muszą być tego świadomym, poniżej przedstawiono kilka przykładów:

Jeśli użyto "jedną z opcji: Użyj istniejącego schematu jako punktu wyjścia" uzyskaną w kroku 3:

- Przyszłe zmiany w modelu wymagają, zmienianie lub porzucanie jeden z obiektów bazy danych, które są nazwane w różny sposób, należy zmodyfikować szkieletu migracji, aby określić poprawną nazwę. Interfejsy API migracji ma opcjonalny parametr nazwę, która pozwala na to. Na przykład istniejącego schematu może zawierać tabelę wpis z BlogId kolumny klucza obcego, która ma indeks o nazwie IndexFk_BlogId. Jednak domyślnie migracje oczekują tego indeksu, aby mieć nazwę IX_BlogId. Jeśli wprowadzisz zmiany w modelu, które powoduje usunięcie tego indeksu będzie konieczne zmodyfikowanie szkieletu wywołana DropIndex w celu określenia IndexFk_BlogId nazwy.

Jeśli użyto "dwóch opcji: Użyj pustej bazy danych jako punktu wyjścia" uzyskaną w kroku 3:

- Próby uruchomienia metody szczegółów początkowej migracji (które powracanie do pustej bazy danych) względem lokalnej bazy danych może zakończyć się niepowodzeniem, ponieważ migracje podejmie próbę Porzuć indeksy i ograniczenia klucza obcego przy użyciu niepoprawnymi nazwami. Wpłynie to tylko lokalna baza danych od innych baz danych zostanie utworzona od podstaw przy użyciu metody w góre początkowej migracji. Jeśli chcesz użyć istniejącej lokalnej bazy danych do stanu pustego najłatwiej można to zrobić ręcznie, poprzez usunięcie bazy danych lub usunięcie wszystkich tabel. Po tej początkowej obniżenia poziomu których wszystkie obiekty bazy danych zostaną odtworzone z domyślnymi nazwami więc ten problem nie przedstawi się ponownie.
- Jeśli przyszłe zmiany w modelu wymagają, zmienianie lub porzucanie jeden z obiektów bazy danych, które są nazwane w różny sposób, to nie będzie działać względem istniejącej lokalnej bazy danych — od nazwy nie pasują do wartości domyślnych. Jednak będą działać względem bazy danych, które zostały utworzone "od zera", ponieważ te będą używane domyślne nazwy wybranego przez migracje. Można ręcznie wprowadzić te zmiany w istniejącej lokalnej bazy danych lub należy rozważyć utworzenie migracje ponownie utworzyć od podstaw — bazy danych, jak wpłynie to na innych komputerach.
- Bazy danych utworzone przy użyciu metody w góre początkowej migracji mogą się nieznacznie różnić z lokalnej bazy danych od czasu obliczeniowego domyślne nazwy dla indeksów i ograniczeń klucza obcego, który będzie używany. Użytkownik może też pozostać przy użyciu dodatkowych indeksów jak migracji utworzy indeksy na kolumny klucza obcego domyślnie — to nie mogły być w przypadku oryginalnej bazy danych lokalnych.

Nie wszystkie obiekty bazy danych są reprezentowane w modelu

Obiekty bazy danych, które nie są częścią modelu nie będzie obsługiwany przez migracje. Może to obejmować widoki, procedury składowane, uprawnienia, tabel, które nie są częścią Twojego modelu, dodatkowe indeksy itp.

Gdy muszą być tego świadomym, poniżej przedstawiono kilka przykładów:

- Niezależnie od opcji wybranej w kroku 3, jeśli wymagają przyszłe zmiany w modelu, zmienianie lub porzucanie tych obiektów dodatkowych migracji nie będzie wiedzieć, aby wprowadzić te zmiany. Na przykład jeśli usuniesz kolumny, która ma inny indeks, migracje będą nie wiedzieć, aby usunąć indeksu. Należy ręcznie dodać to do szkieletu migracji.
- Jeśli użyto "dwóch opcji: Użyj pustej bazy danych jako punkt początkowy", te dodatkowe obiekty nie zostaną utworzone przez metodę w góre początkowej migracji. Można zmodyfikować góre i w dół do metody, aby zadbać o tych obiektów dodatkowych, jeśli chcesz. Dla obiektów, które nie są obsługiwane natywnie w interfejsie API migracje — takich jak widoki — możesz użyć [Sql](#) metodę, aby uruchomić pierwotne SQL do tworzenia i upuścić je.

Dostosowywanie tabeli historii migracji

07.01.2019 • 5 minutes to read • [Edit Online](#)

NOTE

EF6 począwszy tylko — funkcje, interfejsów API itp. z opisem na tej stronie zostały wprowadzone w programie Entity Framework 6. Jeśli używasz starszej wersji, niektóre lub wszystkie informacje, nie ma zastosowania.

NOTE

W tym artykule przyjęto założenie, że wiesz, jak użyć migracje Code First w podstawowych scenariuszy. Jeśli tego nie zrobisz, a następnie należy odczytać [migracje Code First](#) przed kontynuowaniem.

Co to jest tabela historii migracji?

Migracji historii jest tabela używany przez migracje Code First do przechowywania informacji o migracji do niej zastosować. Domyślnie nazwa tabeli w bazie danych jest `_MigrationHistory` i jest tworzony podczas stosowania pierwszej migracji do bazy danych. W Entity Framework 5 Ta tabela została tabeli systemowej, jeśli aplikacja używa bazy danych programu Microsoft Sql Server. Jednakże, to została zmieniona w Entity Framework 6, a tabela historii migracji nie jest już oznaczone tabeli systemowej.

Dlaczego warto dostosować tabeli historii migracji?

Migracji tabeli historii mają być używane wyłącznie przez migracje Code First i zmieniając go ręcznie, może przerwać migracji. Jednak czasem domyślnej konfiguracji nie nadaje się i tabeli musi być dostosowane, na przykład:

- Należy zmienić nazwy i/lub aspekty kolumn, aby umożliwić 3^{usług pulpitu zdalnego} migracje dostawcy
- Aby zmienić nazwę tabeli
- Należy użyć innego niż domyślny schemat `_MigrationHistory` tabeli
- Potrzeba przechowywania dodatkowych danych dla danej wersji kontekstu i w związku z tym należy dodać dodatkową kolumnę do tabeli

Wyrazy środki ostrożności

Zmiana tabeli migracji historii jest skuteczna, ale należy uważać, aby nie nadużywać. Środowiska uruchomieniowego EF aktualnie nie sprawdza, czy tabela historii dostosowane migracji jest zgodne ze środowiskiem uruchomieniowym. Jeśli nie jest aplikacja może przerwać w czasie wykonywania lub nieprzewidywalne zachowanie. Jest to szczególnie ważne, jeśli używasz wielu kontekstach na bazę danych w przypadku wielu kontekstach można użyć tej samej tabeli historii migracji do przechowywania informacji o migracji.

Jak dostosowywać tabelę historii migracji?

Przed rozpoczęciem musisz wiedzieć, można dostosować w tabeli historii migracji tylko w przypadku, przed zainstalowaniem pierwszej migracji. Teraz w kodzie.

Najpierw należy utworzyć klasę pochodną klasy `System.Data.Entity.Migrations.History.HistoryContext`. Klasa

HistoryContext jest pochodną klasy DbContext, więc konfigurowania migracji tabeli historii jest bardzo podobny do konfigurowania EF modeli za pomocą interfejsu API fluent. Wystarczy zastąpić metodę OnModelCreating i skonfigurować tabelę za pomocą interfejsu API fluent.

NOTE

Zwykle po skonfigurowaniu EF modele nie trzeba wywołać podstawowej OnModelCreating() z przeciążonej OnModelCreating od DbContext.OnModelCreating() ma pustą treść. Nie jest tak, podczas konfigurowania migracji tabeli historii. W tym przypadku pierwsze należy w przesłonięcia OnModelCreating() jest faktycznie wywołać podstawowej OnModelCreating(). Skonfiguruje migracji tabeli historii w sposób domyślny, który następnie dostosować w metodzie nadzorowanych.

Załóżmy, że chcesz zmienić nazwę tabeli historii migracje i umieść je ze schematem niestandardowej o nazwie "admin". Oprócz administratora bazy danych chcesz, możesz zmienić nazwy kolumny MigrationId migracji_identyfikatora. Można to osiągnąć, tworząc następujące klasy pochodne HistoryContext:

```
using System.Data.Common;
using System.Data.Entity;
using System.Data.Entity.Migrations.History;

namespace CustomizableMigrationsHistoryTableSample
{
    public class MyHistoryContext : HistoryContext
    {
        public MyHistoryContext(DbConnection dbConnection, string defaultSchema)
            : base(dbConnection, defaultSchema)
        {
        }

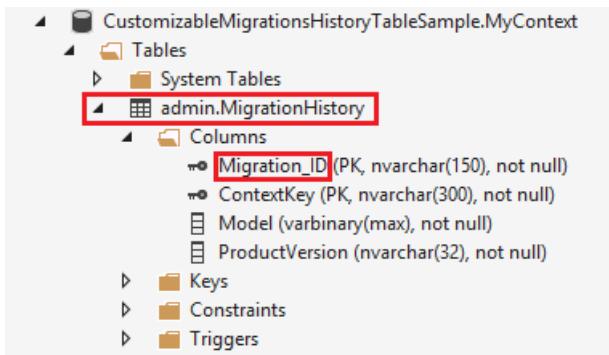
        protected override void OnModelCreating(DbModelBuilder modelBuilder)
        {
            base.OnModelCreating(modelBuilder);
            modelBuilder.Entity<HistoryRow>().ToTable(tableName: "MigrationHistory", schemaName: "admin");
            modelBuilder.Entity<HistoryRow>().Property(p => p.MigrationId).HasColumnName("Migration_ID");
        }
    }
}
```

Gdy Twoje niestandardowe HistoryContext jest gotowy należy wprowadzić EF informację, rejestrując go za pomocą [konfiguracja na podstawie kodu](#):

```
using System.Data.Entity;

namespace CustomizableMigrationsHistoryTableSample
{
    public class ModelConfiguration : DbConfiguration
    {
        public ModelConfiguration()
        {
            this.SetHistoryContext("System.Data.SqlClient",
                (connection, defaultSchema) => new MyHistoryContext(connection, defaultSchema));
        }
    }
}
```

To wszystko właściwie. Teraz możesz przejść do konsoli Menedżera pakietów Enable-Migrations Dodaj migracji, a na koniec Aktualizuj bazy danych. Powinno to spowodować dodanie do tabeli historii migracje skonfigurowana zgodnie ze szczegółami, które określiłeś w klasie pochodnej HistoryContext bazy danych.



Za pomocą migrate.exe

01.10.2018 • 8 minutes to read • [Edit Online](#)

Migracje Code First pozwala zaktualizować bazę danych z wewnątrz programu visual studio, ale mogą być również wykonywane za pośrednictwem migrate.exe narzędzia wiersza polecenia. Ta strona będzie zapewniają szybki przegląd dotyczące sposobu używania migrate.exe do wykonania migracji w bazie danych.

NOTE

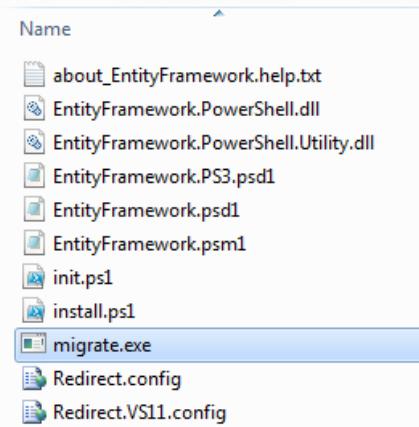
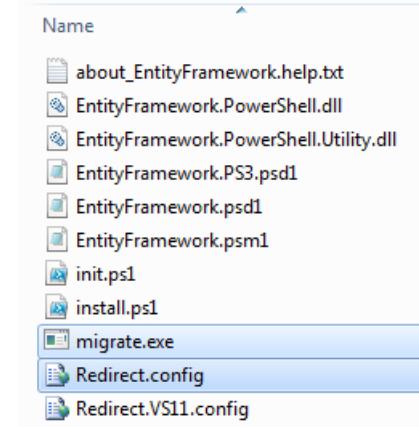
W tym artykule przyjęto założenie, że wiesz, jak użyć migracje Code First w podstawowych scenariuszy. Jeśli tego nie zrobisz, a następnie należy odczytać [migracje Code First](#) przed kontynuowaniem.

Skopiuj migrate.exe

Po zainstalowaniu programu Entity Framework za pomocą narzędzia NuGet migrate.exe będzie znajdować się w folderze narzędzia pobranego pakietu. W <folderu projektu>\pakietów\platformy EntityFramework.<Wersja>\narzędzia

Po utworzeniu migrate.exe musisz skopiować go do lokalizacji zestawu, który zawiera migracji.

Jeśli aplikacja jest przeznaczony dla .NET 4, a nie 4.5, należy skopiować **Redirect.config** w lokalizacji jako dobrze i zmień jego nazwę **migrate.exe.config**. Jest to więc migrate.exe pobiera przekierowania powiązań poprawna, aby można było zlokalizować zestawu platformy Entity Framework.

.NET 4.5	.NET 4.0
	

NOTE

migrate.exe nie obsługuje x64 zestawów.

Po migrate.exe zostały przeniesione do poprawnego folderu powinno być można używać go do wykonania migracji w bazie danych. Jest wszystko, czego narzędzie zaprojektowano w celu wykonania migracji. Nie można wygenerować migracje ani utworzyć skrypt SQL.

[Zobacz Opcje](#)

```
Migrate.exe /?
```

Powyższe spowoduje wyświetlenie strony pomocy skojarzony z tym narzędziu, należy pamiętać, że musisz mieć EntityFramework.dll w tej samej lokalizacji, migrate.exe uruchomionych w kolejności, aby to działało.

Migracja do najnowszych migracji

```
Migrate.exe MyMvcApplication.dll /startupConfigurationFile="..\web.config"
```

Podczas uruchamiania migrate.exe tylko obowiązkowy parametr jest zestawu, który jest zestaw, który zawiera migracji, które próbujesz uruchomić, lecz będzie używać konwencji wszystkie na podstawie ustawienia, jeśli nie określisz pliku konfiguracji.

Migrowanie do dotyczące migracji

```
Migrate.exe MyApp.exe /startupConfigurationFile="MyApp.exe.config" /targetMigration="AddTitle"
```

Jeśli chcesz uruchomić migracje maksymalnie dotyczące migracji, można określić nazwę migracji. Spowoduje to uruchomienie wszystkich poprzednich migracji zgodnie z wymaganiami aż do migracji, określić.

Określ katalog roboczy

```
Migrate.exe MyApp.exe /startupConfigurationFile="MyApp.exe.config" /startupDirectory="c:\\MyApp"
```

Jeśli użytkownik zestawu ma zależności lub odczytuje pliki względem katalogu roboczego należy ustawić startupDirectory.

Określ konfigurację migracji w celu użycia

```
Migrate.exe MyAssembly CustomConfig /startupConfigurationFile="..\web.config"
```

Jeśli masz wiele klas konfiguracji migracji, klas dziedziczących DbMigrationConfiguration, należy określić, który ma być używany dla wykonania. To jest określona, zapewniając opcjonalny drugi parametr bez przełącznika jako powyżej.

Podaj parametry połączenia

```
Migrate.exe BlogDemo.dll /connectionString="Data Source=localhost;Initial Catalog=BlogDemo;Integrated Security=SSPI" /connectionProviderName="System.Data.SqlClient"
```

Jeśli chcesz określić parametry połączenia, w wierszu polecenia, należy również podać nazwę dostawcy. Nazwa dostawcy nie został podany spowoduje, że wyjątek.

Typowe problemy

KOMUNIKAT O BŁĘDZIE	ROZWIĄZANIE
Nieobsługiwany wyjątek: System.IO.FileLoadException: nie można załadować pliku lub zestawu "platformy EntityFramework, wersja = 5.0.0.0, Culture = neutral, PublicKeyToken = b77a5c561934e089" lub jednej z jego zależności. Definicja manifestu zestawu znajduje się niezgodna odwołanie do zestawu. (Wyjątek od HRESULT: 0x80131040)	Zwykle oznacza to, czy używasz aplikacji .NET 4, bez pliku Redirect.config. Należy skopiować Redirect.config do tej samej lokalizacji co migrate.exe i zmień jego nazwę na migrate.exe.config.
Nieobsługiwany wyjątek: System.IO.FileLoadException: nie można załadować pliku lub zestawu "platformy EntityFramework, wersja = 4.4.0.0, Culture = neutral, PublicKeyToken = b77a5c561934e089" lub jednej z jego zależności. Definicja manifestu zestawu znajduje się niezgodna odwołanie do zestawu. (Wyjątek od HRESULT: 0x80131040)	Ten wyjątek oznacza, że działają .NET 4.5 aplikacji przy użyciu Redirect.config skopowane do lokalizacji migrate.exe. Jeśli Twoja aplikacja jest .NET 4.5 nie trzeba mieć plik konfiguracji z przekierowywaniem wewnątrz. Usuń plik migrate.exe.config.
Błąd: Nie można zaktualizować bazy danych do dopasowania w bieżącym modelu, ponieważ istnieją oczekujące zmiany, a następnie automatycznej migracji jest wyłączona. Zapisać zmiany oczekujące modelu migracji za pomocą kodu lub umożliwić automatyczną migrację. Ustaw DbMigrationsConfiguration.AutomaticMigrationsEnabled na wartość PRAWDA, aby włączyć automatyczną migrację.	Ten błąd występuje podczas uruchamiania migracji nie utworzono migracji, aby poradzić sobie ze zmianami wprowadzonymi w modelu, gdy baza danych nie jest zgodny z modelem. Dodawanie właściwości do klasy modelu, ponownie uruchomić migrate.exe bez tworzenia migracji, aby uaktualnić bazę danych jest przykładem.
Błąd: Typ nie zostanie rozwiązany dla elementu członkowskiego "System.Data.Entity.Migrations.Design.ToolingFacade+UpdateRunner,EntityFramework, Version = 5.0.0.0, Culture = neutral, PublicKeyToken = b77a5c561934e089".	Ten błąd może być spowodowany przez określenie katalogu startup niepoprawne. Musi to być lokalizacja migrate.exe
Nieobsługiwany wyjątek: System.NullReferenceException: odwołanie nie ustawione na wystąpienie obiektu do obiektu. w System.Data.Entity.Migrations.Console.Program.Main (String [] args)	Może to być spowodowane nie został podany wymagany parametr dla scenariusza, którego używasz. Na przykład określenie parametrów połączenia bez określenia nazwy dostawcy.
Błąd: znaleziono więcej niż jeden typ konfiguracji migracji w zestawie "ClassLibrary1". Określ nazwę, aby korzystać.	Jak opisu błędu wynika, ma więcej niż jedną klasę konfiguracji w danym zestawie. Należy użyć przełącznika /configurationType, aby określić, której mają zostać użyte.
Błąd: Nie można załadować pliku lub zestawu '<assemblyName>' lub jednej z jego zależności. Dany zestaw nazwy lub codebase był nieprawidłowy. (Wyjątek od HRESULT: 0x80131047)	Może to być spowodowane przez niepoprawnie określający nazwę zestawu, lub nie ma
Błąd: Nie można załadować pliku lub zestawu '<assemblyName>' lub jednej z jego zależności. Próbowano załadować program w niepoprawnym formacie.	Dzieje się tak, jeśli chcesz uruchamiać migrate.exe x64 aplikacji. EF 5.0 i poniżej spowoduje działają tylko na x86.

Migracje Code First na środowiska zespołowe

05.12.2018 • 26 minutes to read • [Edit Online](#)

NOTE

W tym artykule przyjęto założenie, że wiesz, jak użyć migracji Code First w podstawowych scenariuszy. Jeśli tego nie zrobisz, a następnie należy odczytać [migracje Code First](#) przed kontynuowaniem.

Pobierz kawy, musisz przeczytać ten artykuł, całe

Problemów w środowiskach zespołu są głównie wokół scalanie migracji po dwóch programistów wygenerowały migracji w swoich lokalnych kod podstawowy. Kroki, aby rozwiązać te są całkiem proste, wymagają one może być pełny opis sposobu działania migracji. Proszę nie po prostu przejść od razu na końcu — Poświęć chwilę Aby odczytać cały artykuł, aby upewnić się, że możesz pomyślnie.

Ogólne wskazówki

Przed rozpoczęciem omawiania jak zarządzać scalania migracje generowane przez wielu deweloperów, poniżej przedstawiono ogólne wskazówki, które ukierunkowane na powodzenie.

Każdy członek zespołu powinien mieć lokalny rozwój bazy danych

Zastosowań migracje `_MigrationsHistory` tabelę do przechowywania, jakie migracji zostały zastosowane do bazy danych. Jeśli masz wielu deweloperów generowania różne migracje podczas próby pod kątem tej samej bazy danych (i udostępnianie w ten sposób `_MigrationsHistory` tabeli) migracje zamierza uzyskać bardzo mylące.

Oczywiście w przypadku członków zespołu, którzy nie są Generowanie migracje występuje problem, nie pozwól, aby udostępnić bazę danych centralnej rozwoju.

Należy unikać automatycznej migracji

Mierzenie to, że automatycznej migracji początkowo wyglądają dobrze w środowiskach zespołu, ale w rzeczywistości po prostu nie działają. Jeśli chcesz wiedzieć Dlaczego, Zachowaj czytania — Jeśli nie, następnie można przejść do następnej sekcji.

Automatyczne migracji umożliwia zostały zaktualizowane w celu dopasowania bieżącego modelu bez konieczności generowanie kodu plików (migracje oparte na kodzie) schemat bazy danych. Automatyczne migracje będzie bardzo dobrze pracować w środowisku zespołowym, jeśli tylko nigdy nie użył ich i nigdy nie generowane wszystkie migracje oparte na kodzie. Problem polega na czy automatycznej migracji są ograniczone i nie obsługują wielu operacji — zmienia nazwę kolumny właściwości/przenoszenia danych do innej tabeli itp. Do obsługi tych scenariuszy, na końcu generowania migracje oparte na kod (i edytowania utworzony szkielet kodu), które są zmieszczone Between zmiany, które są obsługiwane przez automatyczne migracji. Dzięki temu niemal na niemożliwe do scalania zmian, gdy dwa deweloperzy ewidencjonują migracji.

Zrzuty ekranu

Jeśli zrzut ekranu niż przeczytać ten artykuł będzie wolisz obejrzeć, następujące dwa filmy video obejmuje tę samą zawartość, jak w tym artykule.

Wideo 1: "Migracje - Under the Hood"

Ten zrzut ekranu opisano sposób migracji śledzi i używa informacji o modelu, aby wykrywać zmiany modelu.

Dwa wideo: "Migracje - środowiska zespołowe"

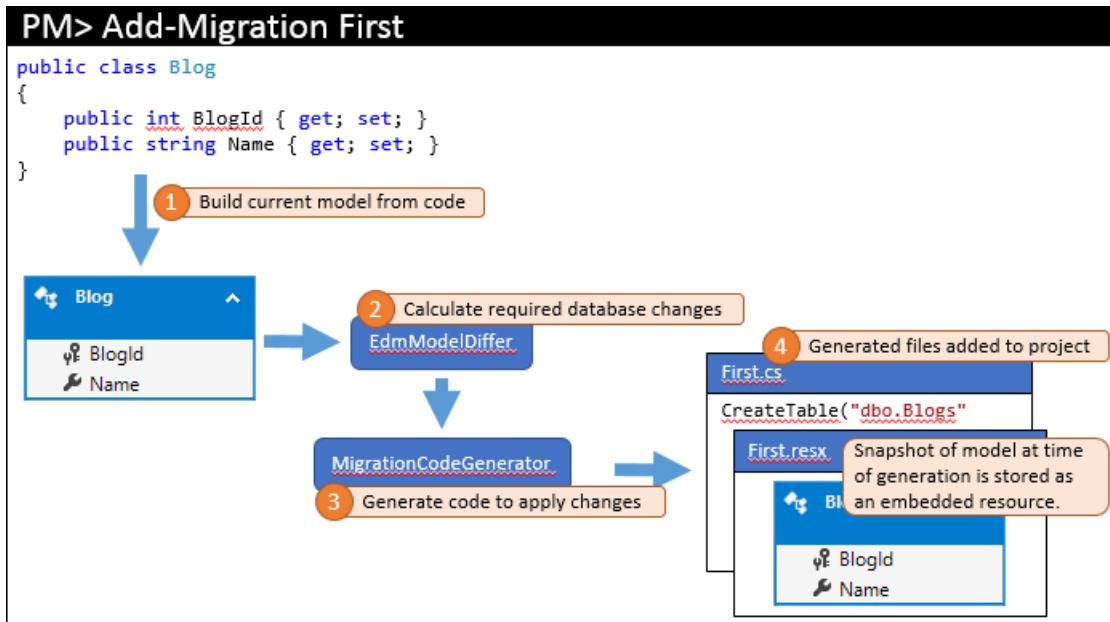
Opierając się na koncepcji z poprzednim filmu wideo, [ten zrzut ekranu](#) omówiono zagadnienia, które pojawiają się w środowisku zespołowym oraz sposoby ich rozwiązywania.

Zrozumienie sposobu działania migracji

Klawisz, aby pomyślnie przy użyciu migracji w środowisku zespołowym, to podstawowy zrozumienie, jak migracje śledzi i używa informacji o modelu do wykrycia zmiany modelu.

Pierwszej migracji

Po pierwszej migracji możesz dodać do projektu, możesz uruchomić podobny **migracji Dodaj pierwszy** w konsoli Menedżera pakietów. Kroki wysokiego poziomu, które wykonuje to polecenie jest na ilustracji poniżej.



Bieżący model jest obliczana w kodzie (1). Obiekty bazy danych wymagane jest następnie obliczana z użyciem różnią się modelu (2) — ponieważ jest to pierwszy migracji modelu różni się jedynie używa pusty model do porównania. Wymagane zmiany są przekazywane do generatora kodu, aby skompilować kod Wymagana migracja (3), która jest następnie dodawana do rozwiązania programu Visual Studio (4).

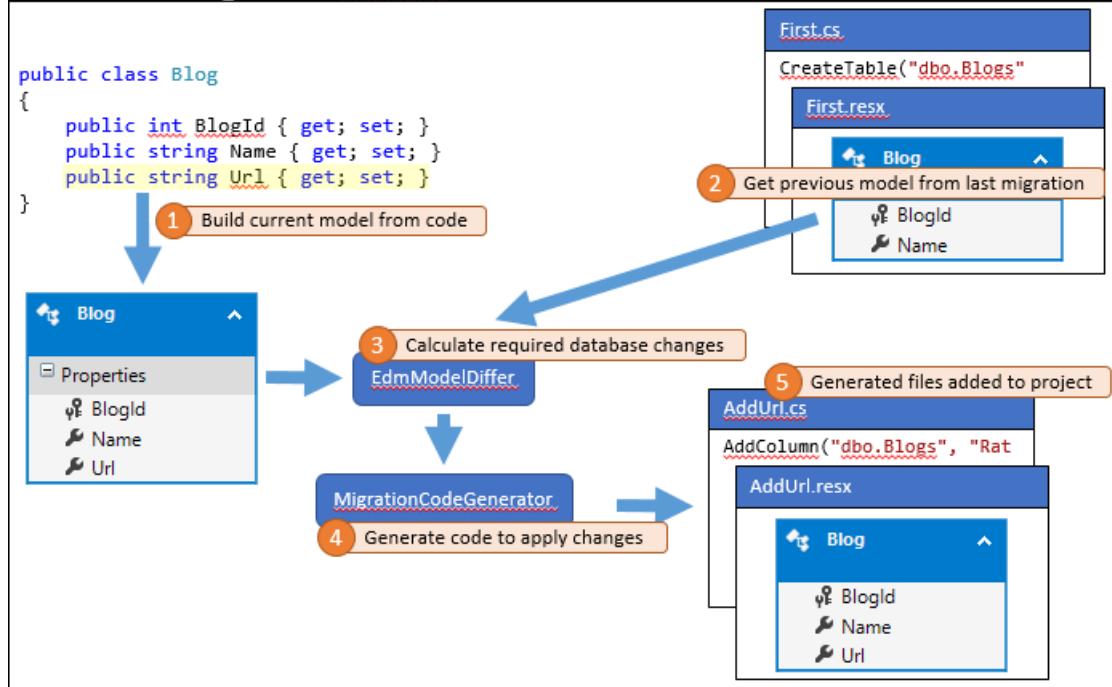
Oprócz kodu rzeczywistą migrację, który jest przechowywany w pliku głównego kodu migracje generuje również niektórych dodatkowych plików z kodem. Te pliki są metadane, który jest używany przez migracje i nie coś, co należy edytować. Jeden z tych plików jest plik zasobów (.resx) zawiera migawkę tego modelu w czasie migracji została wygenerowana. Zobaczysz, jak jest on używany w następnym kroku.

W tym momencie należy prawdopodobnie uruchomić **Update-Database** Aby zastosować zmiany do bazy danych, a następnie przejdź dotyczących implementowania innych obszarów aplikacji.

Kolejne migracje

Później możesz wrócić i wprowadzić pewne zmiany na modelu — w tym przykładzie dodamy **adresu Url** właściwości **blogu**. Następnie może wydać polecenie takich jak **AddUrl migracji Dodaj** zmienia się do tworzenia szkieletu migracji, aby zastosować odpowiednie bazy danych. Kroki wysokiego poziomu, które wykonuje to polecenie jest na ilustracji poniżej.

PM > Add-Migration AddUrl



Podobnie jak wcześniej bieżący model jest obliczana na podstawie kodu (1). Tym razem istnieją migracji istniejących więc w poprzednim modelu są pobierane z najnowszych migracji (2). Te dwa modele są diffed można znaleźć wymaganych zmian w bazie danych (3), a następnie proces kończy się tak jak poprzednio.

Ten sam proces jest używany dla dalszych migracji, które dodajesz do projektu.

Dlaczego odblokowane z migawką modelu?

Możesz się zastanawiać, dlaczego EF bothers z migawką modelu — Dlaczego się nie tylko bazy danych. Jeśli tak, Czytaj dalej. Jeśli użytkownik nie chce można pominąć tę sekcję.

Istnieje wiele możliwych przyczyn, które EF przechowuje migawki modelu wokół:

- Umożliwia ona bazy danych w celu odstają od modelu platformy EF. Tych zmian bezpośrednio w bazie danych lub utworzony szkielet kodu można zmienić w migracji w taki sposób, aby wprowadzić zmiany. Poniżej przedstawiono kilka przykładów tego w praktyce:
 - Aby dodać kolumnę do jednej lub kilku tabel wstawione i zaktualizowane, ale nie chcesz uwzględnić te kolumny w modelu platformy EF. Jeśli migracja przyjrzało się bazy danych stale może spróbować usunąć te kolumny, za każdym razem, gdy działanie migracji. Przy użyciu migawki modelu, EF tylko wykryje uzasadnione zmiany w modelu.
 - Chcesz zmienić treść funkcji procedurę składowaną, która jest używana w przypadku aktualizacji do uwzględnienia niektórych rejestrów. Jeśli migracji przyjrzało się procedurę składowaną z bazy danych będzie stale spróbuj i przywrócić jego definicji, który oczekuje, że EF. Za pomocą migawki modelu, EF będzie tylko tworzenia szkieletu kodu lub zmieniać procedury składowanej w przypadku zmian kształtu procedury opisaną w modelu platformy EF.
 - Te same zasady mają zastosowanie do dodawania dodatkowych indeksów, w tym dodatkowe tabele w bazie danych, mapowanie EF na widok bazy danych, który znajduje się za pośrednictwem tabeli itp.
- Modelu platformy EF zawiera więcej niż tylko kształt bazy danych. Posiadanie cały model umożliwia migracji wyświetlić informacje dotyczące właściwości i klasy w modelu oraz sposobu mapowania ich na kolumn i tabel. Informacje te pozwalają migracji się bardziej intelligentne w kodzie, który go szkielety mechanizmów. Na przykład jeśli zmienisz nazwę kolumny, która właściwość mapuje do migracji może wykrywać zmiany nazwy, zobaczysz, że jest tą samą właściwość — coś, co nie można przeprowadzić, jeśli masz tylko schemat bazy danych.

Co powoduje, że problemy w środowiskach zespołu

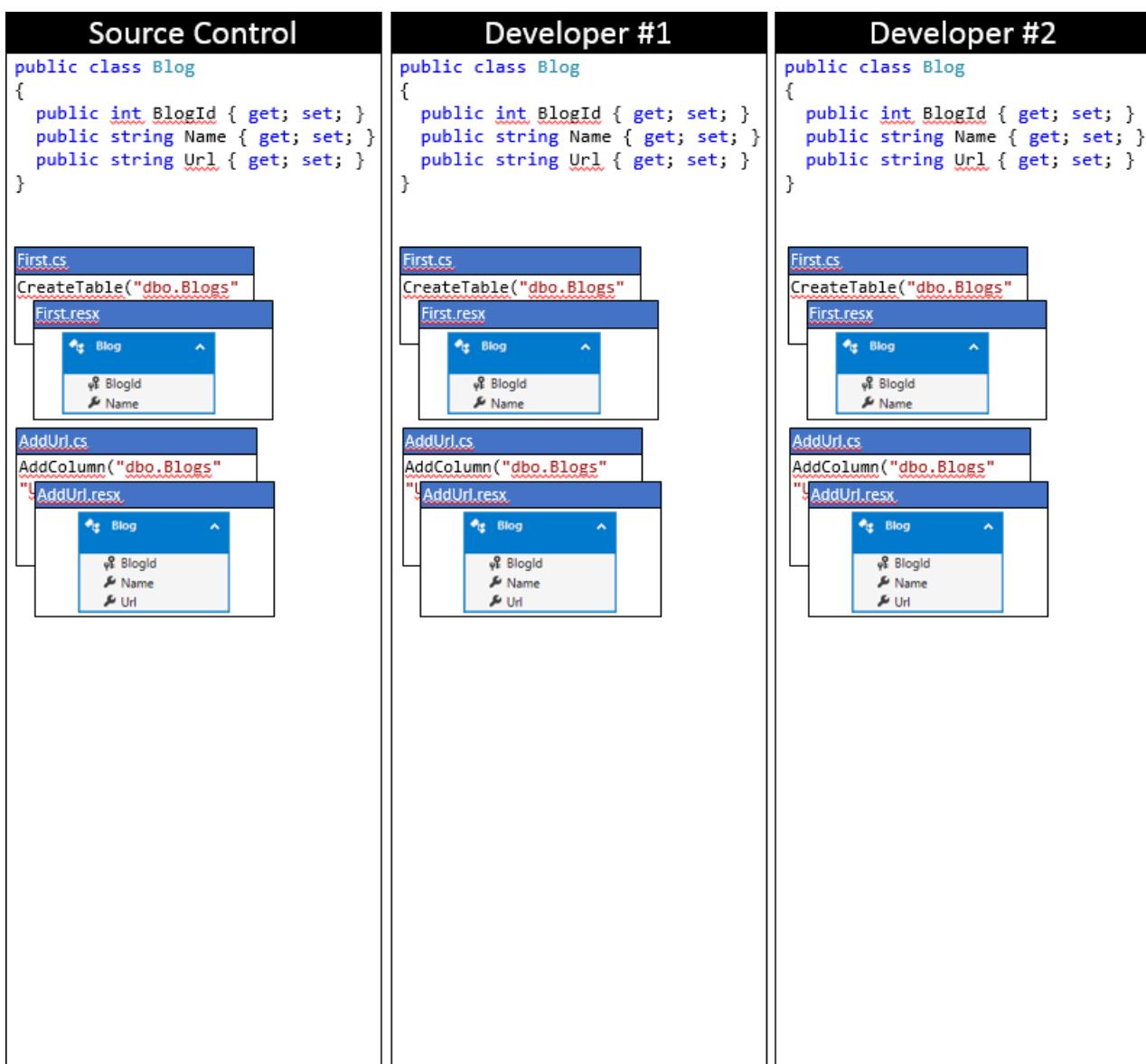
Przepływ pracy omówione w poprzednich działach sekcja doskonała, podczas pracy nad aplikacją jednego dewelopera. Działa dobrze w środowisku zespołowym, jeśli jesteś jedyną osobą, zmiany wprowadzone w modelu. W tym scenariuszu można wprowadzić zmiany w modelu, generowania migracji i przesyłanie ich do kontroli źródła. Inni deweloperzy mogą synchronizowanie zmian i uruchamiać **Update-Database** zastosować zmian schematu.

Problemy z Rozpocznij wystąpić w przypadku wielu deweloperów wprowadzanie zmian modelu platformy EF i przesyłanie do kontroli źródła, w tym samym czasie. Nie posiada EF jest najwyższej klasy można scalić migracji lokalnej za pomocą migracji, które innemu deweloperowi zostało przesłane do kontroli źródła, ponieważ Ostatnia synchronizacja.

Przykładem konfliktu scalania

Pierwszy Przyjrzymy się konkretny przykład konfliktu scalania. Będziemy dalej na przykład, który przyjrzelismy się wcześniej. Jako początkowy punkt możemy założyć zmian z poprzedniej sekcji zostały zaewidencjonowane przez dewelopera, oryginalnym. Będziesz Śledzimy dwa deweloperom, jak oni wprowadzić zmiany do kodu podstawowego

Firma Microsoft będzie śledzić modelu platformy EF migracjach do większej liczby zmian. Punkt początkowy zarówno deweloperów zsynchronizowanych repozytorium kontroli źródła, jak pokazano na poniższym rysunku.

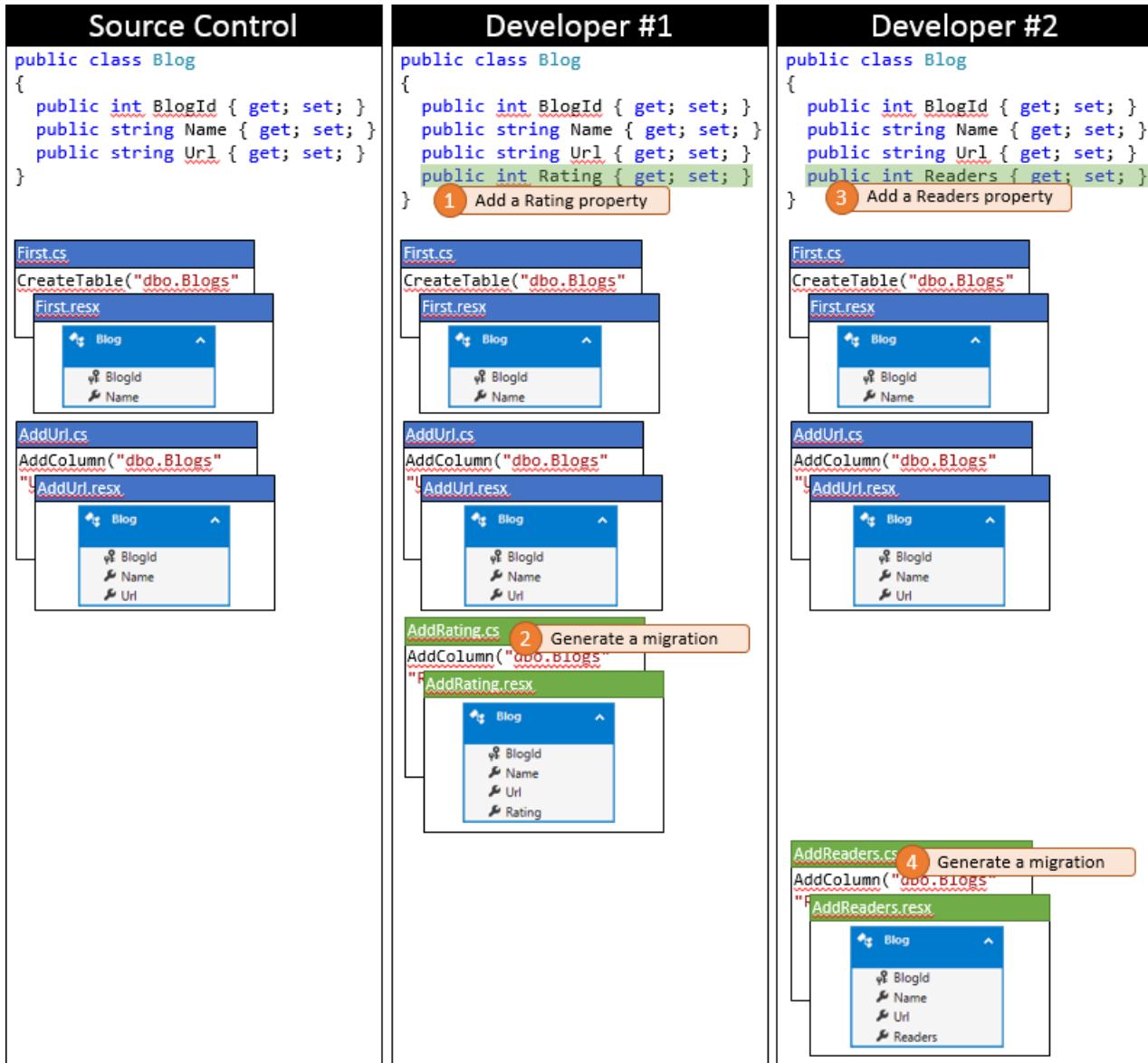


Deweloper #1 i deweloperów #2 teraz sprawia, że pewne zmiany do modelu platformy EF w swoich lokalnych kod

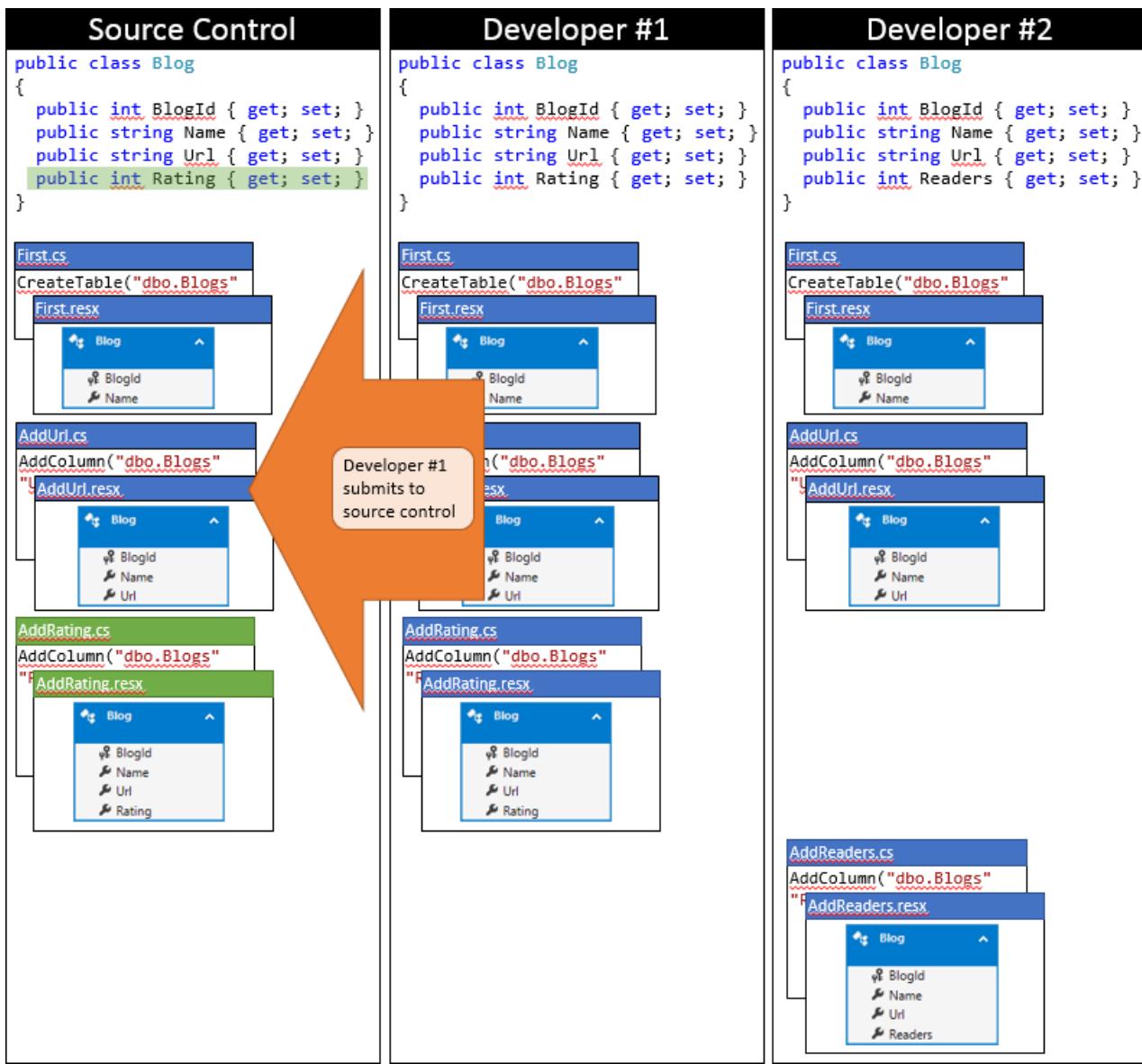
podstawowy. Deweloper #dodaje 1 ocena właściwości **Blog** — i generuje **AddRating** migracji w celu zastosowania zmian w bazie danych. Deweloper #2 dodaje **czytelnicy** właściwości **Blog** — i generuje odpowiedni **AddReaders** migracji. Uruchom zarówno deweloperów **Update-Database**, aby zastosować zmiany w swoich lokalnych baz danych, a następnie kontynuuj, tworzenia aplikacji.

NOTE

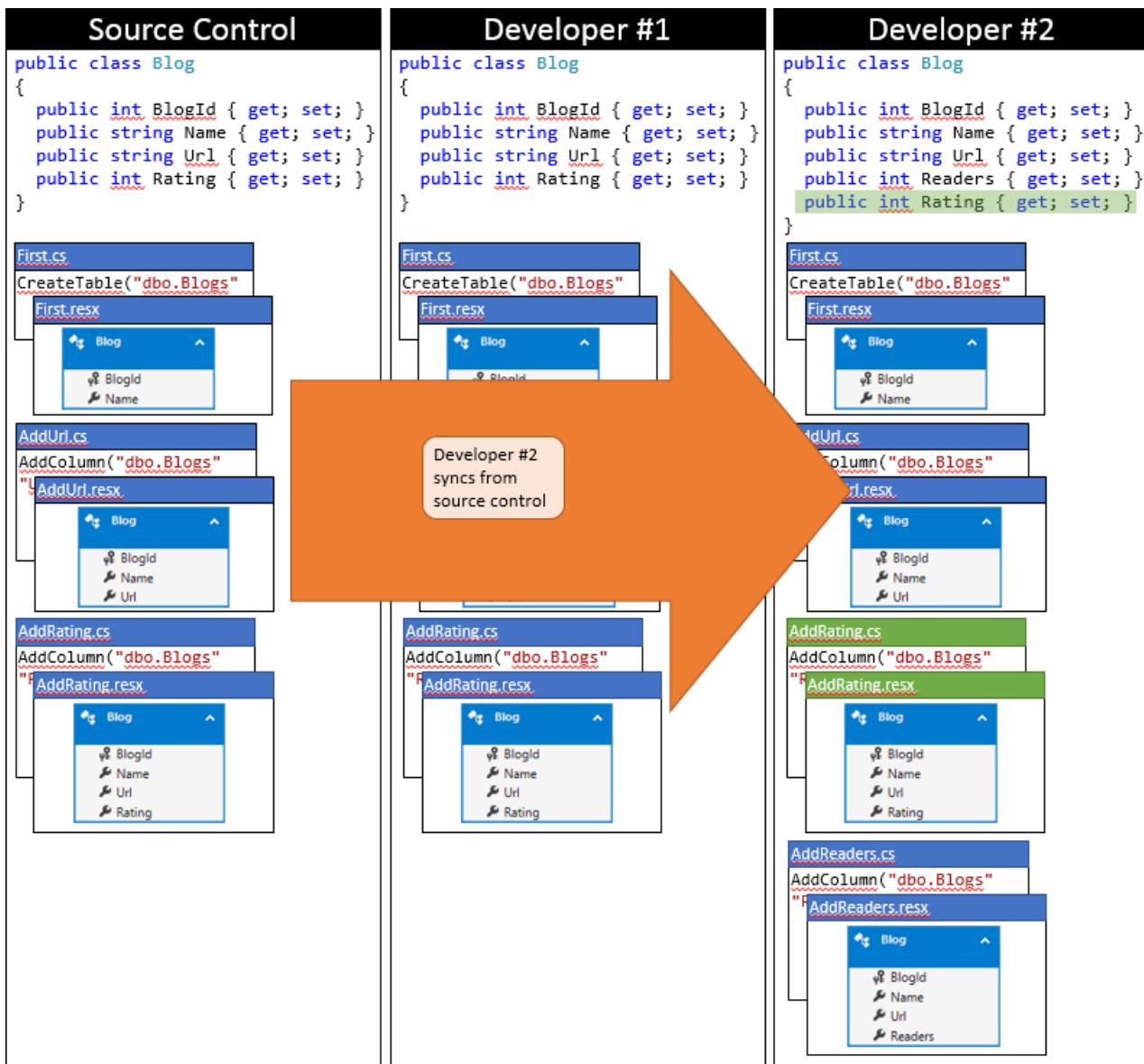
Migracje mają prefiks sygnaturę czasową, dzięki czemu nasze grafiki przedstawia, migracja AddReaders z deweloperów #2 pochodzi po migracji AddRating Developer #1. Czy dla deweloperów #1 lub #2 generowane migracji pierwszej sprawia, że nie ma wpływu problemów pracy w zespole lub procesu scalania ich, które omówimy w następnej sekcji.



To dzień, szczerście Deweloper #1 po ich wprowadzeniu przedstawią ich zmiany. Ponieważ żaden inny użytkownik zaewidencjonował ponieważ one zsynchronizowane z własnym repozytorium, ich zmiany może przesyłać tylko bez przeprowadzania żadnych scalania.



Teraz nadszedł czas na dewelopera #2, aby przesłać. Nie są one takie szczęście. Ponieważ ktoś inny zostało przesłane zmiany, ponieważ są synchronizowane, należy ściągnąć zmiany i scalania. System kontroli źródła prawdopodobnie będzie automatycznie scalić zmiany na poziomie kodu, ponieważ są one bardzo proste. Stan dla deweloperów #2 użytkownika lokalnego repozytorium po synchronizacji jest przedstawiony na poniższym rysunku.



W tym testowaniu Developer #2 można uruchomić **Update-Database** wykryje nowe **AddRating** migracji (nie został zastosowany do deweloperów #2 bazy danych) i zastosować je. Teraz **ocena** kolumna zostanie dodana do **blogi** tabeli i bazy danych jest zsynchronizowany z modelem.

Istnieje jednak kilka problemów:

1. Mimo że **Update-Database** zastosuje **AddRating** migracji również zgłosi ostrzeżenie: *nie można zaktualizować bazy danych, aby dopasować w bieżącym modelu, ponieważ istnieją oczekujące zmiany i Automatyczna migracja jest wyłączona...* Problem polega na to, że migawka modelu przechowywane w ostatniej migracji (**AddReader**) brakuje **ocena** właściwość **blogu** (ponieważ nie było częścią modelu po Migracja została wygenerowana). Kod najpierw wykrywa, że model w ostatniej migracji nie jest zgodny z bieżącym modelem i wywołuje ostrzeżenie.
2. Uruchomiona jest aplikacja mogłoby spowodować `InvalidOperationException` o treści "*modelu kopii kontekstu "BlogginContext" została zmieniona od czasu utworzenia bazy danych. Należy wziąć pod uwagę przy użyciu migracje Code First w aktualizacji bazy danych...*" Ponownie problem polega na migawki modelu, przechowywane w ostatniej migracji nie jest zgodna bieżącego modelu.
3. Na koniec będzie oczekujemy, że działa **migracji Dodaj** teraz wygeneruje pusty migracji (ponieważ nie wprowadzono żadnych zmian do zastosowania do bazy danych). Ale ponieważ migracje porównuje bieżący model na jeden z ostatnich migracji (czyli Brak **ocena** właściwość) on zostanie faktycznie tworzenia szkieletu innego **AddColumn** wywołanie do dodania w **Ocena** kolumny. Oczywiście, ta migracja może zakończyć się niepowodzeniem podczas **Update-Database** ponieważ **ocena** kolumna już istnieje.

Rozwiązywanie konfliktu scalania

Dobra wiadomość jest, że nie jest zbyt trudne do przeciwdziałania scalanie ręczne — pod warunkiem, że zrozumienie sposobu działania migracji. Jeśli zostały pominięte w sekcji do tej sekcji... Niestety musisz przejść wstecz i czytania dalszej części tego artykułu, najpierw!

Dostępne są dwie opcje najprostsza polega na generowaniu pustego migracji, który ma poprawne bieżący model jako migawka. Drugą opcją jest aktualizacja migawki w ostatniej migracji ma poprawny model migawki. Drugą opcją jest nieco bardziej skomplikowane i nie można użyć w każdym scenariuszu, ale jest również bardziej przejrzysty, ponieważ nie wymaga dodanie dodatkowych migracji.

Opcja 1: Dodaj migrację puste scalania

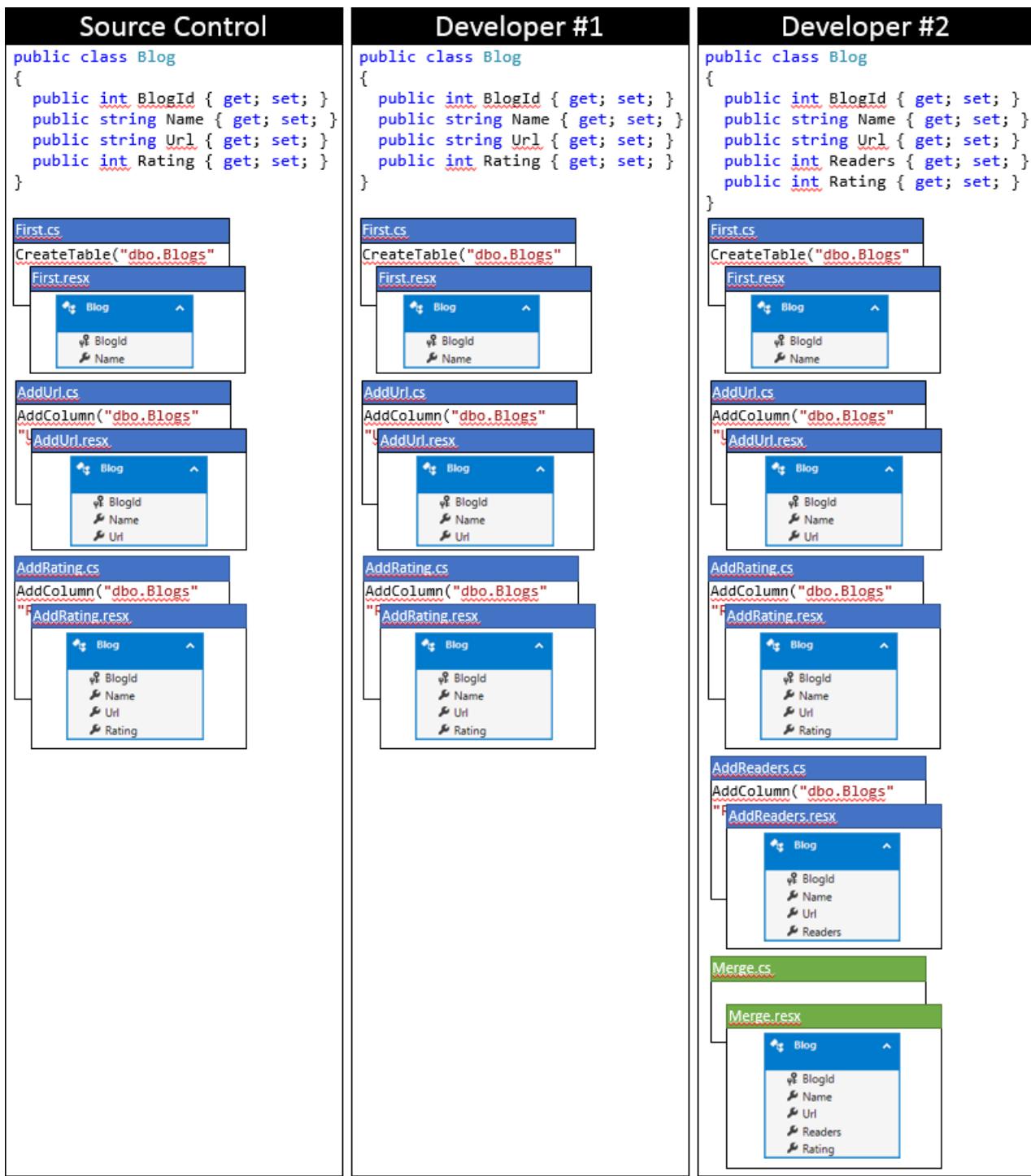
W przypadku tej opcji, wygenerowanie pustego migracji wyłącznie w celu umożliwienia się, że najnowsze migracji ma w nim przechowywane migawki poprawny model.

Ta opcja może służyć niezależnie od wartości która wygenerowała ostatniej migracji. W przykładzie, możemy wykonywano Developer #2 jest zwracając szczególną uwagę scalania i ich wystąpienia, aby wygenerować ostatniej migracji. Ale te same kroki mogą być używane, jeśli deweloper #1 generowane ostatniej migracji. Te kroki mają zastosowanie, jeśli wiele migracji są zaangażowani — firma Microsoft została właśnie zostało spojrzenie na dwa w celu uproszczenia.

Następujący proces może służyć w tym podejściu, rozpoczynając od godziny, o których należy pamiętać, że masz zmiany, które muszą zostać zsynchronizowane z kontrolą źródła.

1. Upewnij się, że wszelkie zmiany oczekujące modelu w kodzie lokalnych, zostały zapisane w migracji. Ten krok zapewnia, że nie przegap wszelkie uzasadnione zmiany, kiedy nastąpi moment, aby wygenerować pusty migracji.
2. Synchronizowanie z kontrolą źródła.
3. Uruchom **Update-Database** do zastosowania nowej migracji, które zostały zaewidencjonowane innym deweloperom. **Uwaga:** *Jeśli nie dostaniesz żadnych ostrzeżeń polecenia Update-Database nie było żadnych nowych migracji od innych deweloperów i trzeba wykonywać dalszych scalania.*
4. Uruchom **migracji Dodaj <wybierz_nazwa> — IgnoreChanges** (na przykład **scalania migracji Dodaj — IgnoreChanges**). To generuje migracji za pomocą wszystkich metadanych (w tym migawki bieżącego modelu), ale będzie ignorować wszelkie zmiany wykrycia podczas porównywania bieżący model z migawką w ostatnim migracji (co oznacza, Pobierz pusty **się i Dół** metody).
5. Kontynuować tworzenie lub Prześlij do kontroli źródła (po wykonywania z testów jednostkowych oczywiście).

W tym miejscu jest stan dla deweloperów #2 użytkownika lokalnego kodu bazowego po zakończeniu korzystania z tej metody.



Opcja 2: Zaktualizuj migawkę modelu w ostatniej migracji

Ta opcja jest bardzo podobna do opcji 1, ale usuwa dodatkowe puste migracji —, ponieważ umożliwia prawdzie, kto chce, aby pliki dodatkowego kodu w swoich rozwiązańach.

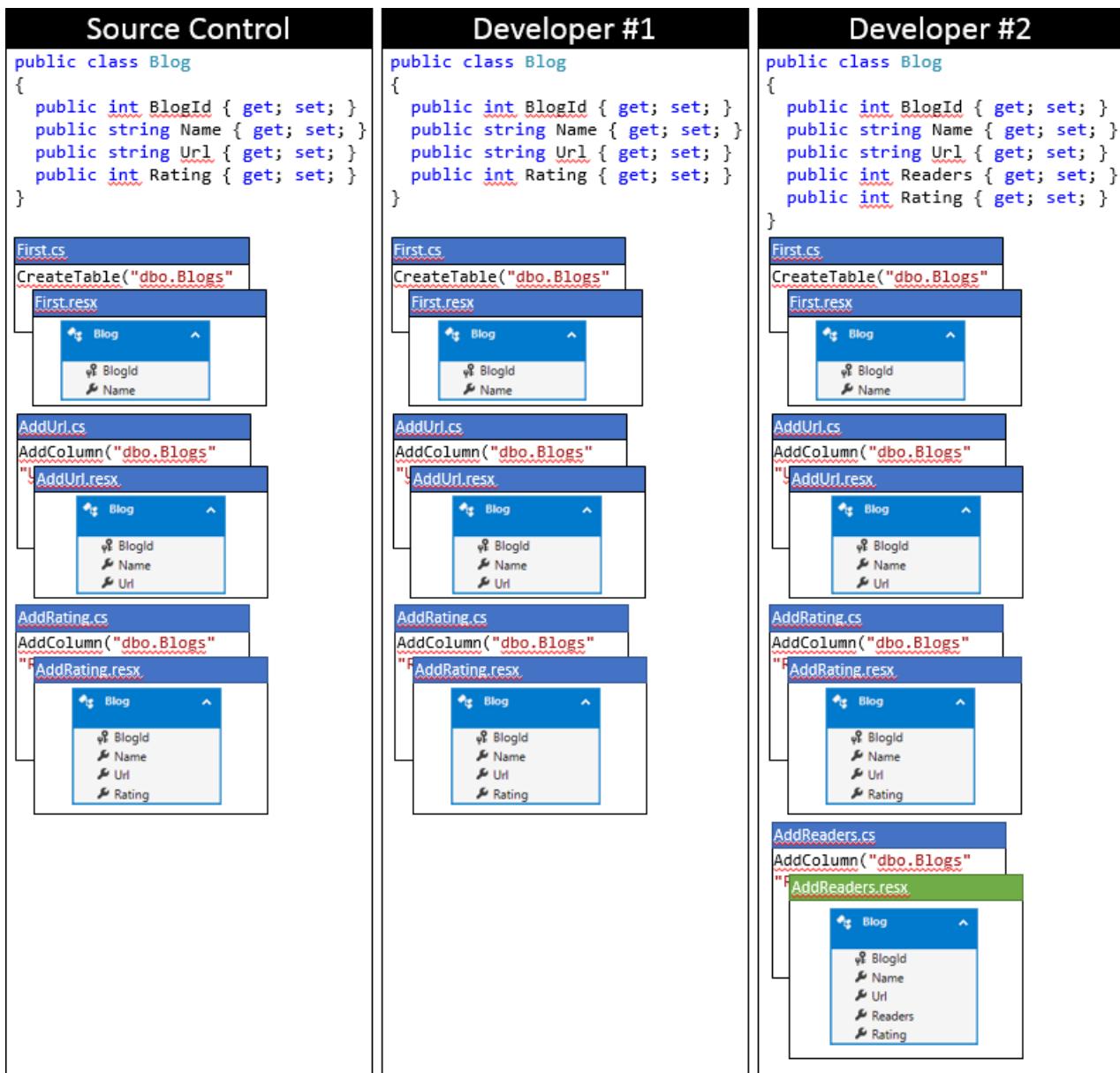
To podejście jest możliwe tylko, jeśli istnieje tylko w podstawowym kod lokalny najnowszych migracji, a nie jeszcze został przesłany do kontroli źródła (na przykład, jeśli ostatni migracji została wygenerowany przez użytkownika, wykonując scalanie). Edytować metadane migracji, które inni deweloperzy już zastosowane do swojej bazy danych rozwoju — lub nawet co gorsza stosowane do produkcyjnej bazy danych — może spowodować nieoczekiwane działania niepożądane. W procesie zamierzamy wycofać migrację ostatni w naszym lokalnej bazy danych, a następnie ponownie zastosuj go ze zaktualizowanymi metadanymi.

Podczas ostatniej migracji konieczne po prostu znajdować się w lokalnym kod podstawowy, nie ma żadnych ograniczeń liczbę lub kolejność migracji, które ją kontynuować. Może istnieć wiele migracji z wielu różnych deweloperów i Zastosuj te same czynności — firma Microsoft została właśnie spojrzenie na dwa w celu uproszczenia.

Następujący proces może służyć w tym podejściu, rozpoczynając od godziny, o których należy pamiętać, że masz zmiany, które muszą zostać zsynchronizowane z kontroli źródła.

1. Upewnij się, że wszelkie zmiany oczekujące modelu w kodzie lokalnych, zostały zapisane w migracji. Ten krok zapewnia, że nie przegap wszelkie uzasadnione zmiany, kiedy nastąpi moment, aby wygenerować pusty migracji.
2. Synchronizowanie z kontrolą źródła.
3. Uruchom **Update-Database** do zastosowania nowej migracji, które zostały zaewidencjonowane innym deweloperom. **Uwaga:** Jeśli nie dostaniesz żadnych ostrzeżeń polecenia Update-Database nie było żadnych nowych migracji od innych deweloperów i trzeba wykonywać dalszych scalania.
4. Uruchom **Update-Database-TargetMigration <drugi_ostatniego_migracji>** (w przykładzie, możemy wykonywano takie rozwiązanie byłoby aktualizacji bazy danych — TargetMigration AddRating). To role bazy danych z powrotem do stanu drugiego ostatnie migracji — skutecznie "bez stosowania" ostatniej migracji z bazy danych. **Uwaga:** ten krok jest wymagany, aby bezpiecznie edytować metadane migracji, ponieważ metadane są również przechowywane w _MigrationsHistoryTable bazy danych. Jest to, dlaczego tej opcji należy używać tylko, jeśli ostatni migracji jest tylko w lokalnej bazie kodu. Jeśli innych baz danych było ostatniej migracji, które są stosowane również masz do nich wycofać, a następnie ponownie zastosuj ostatniej migracji w celu zaktualizowania metadanych.
5. Uruchom **migracji Dodaj <pełną nazwa tym_sygnatura czasowa_z ostatniego migracji >** (w tym przykładzie Firma Microsoft wykonywano powinien to być podobny do **201311062215252 migracji Dodaj_AddReaders**). **Uwaga:** muszą zawierać sygnaturę czasową, aby migracje wie, którą chcesz edytować istniejącej migracji, a nie tworzenia szkieletów nową. Spowoduje to zaktualizowanie metadanych dla ostatniej migracji do dopasowania w bieżącym modelu. Po wykonaniu polecenia, ale jest to dokładnie co chcesz otrzymasz następujące ostrzeżenie. "Tylko kodu projektanta do migracji" 201311062215252_AddReaders został ponownie utworzony szkielet. Aby ponownie utworzyć szkielet całego migracji, należy użyć parametru -Force. "
6. Uruchom **Update-Database** Ponowne zgłoszenie części najnowszych migracji ze zaktualizowanymi metadanymi.
7. Kontynuować tworzenie lub Prześlij do kontroli źródła (po wykonywania z testów jednostkowych oczywiście).

W tym miejscu jest stan dla deweloperów #2 użytkownika lokalnego kodu bazowego po zakończeniu korzystania z tej metody.



Podsumowanie

Istnieją niektóre wyzwania, korzystając z migracji Code First w środowisku pracy dla zespołu. Jednak podstawową wiedzę na temat sposobu działania migracji i niektóre proste podejścia do rozwiązywania konfliktów scalania ułatwiają przewyciążyć te wyzwania.

Podstawowe problem jest nieprawidłowa z metadanych najnowszych migracji. Powoduje to Code First nieprawidłowo wykrywa, że bieżący model i schemat bazy danych nie są zgodne i tworzenia szkieletu niepoprawny kod w następnej migracji. Takiej sytuacji można rozwiązać przez generowanie pustej migracji za pomocą modelu poprawne lub aktualizowanie metadanych w najnowszych migracji.

Najpierw modelu

27.09.2018 • 14 minutes to read • [Edit Online](#)

W tym przewodniku krok po kroku i wideo zawierają wprowadzenie do rozwoju pierwszego modelu używający narzędzia Entity Framework. Model umożliwia najpierw utworzyć nowy model przy użyciu programu Entity Framework Designer, a następnie wygenerować schemat bazy danych z modelem. Model jest przechowywany w pliku EDMX (z rozszerzeniem edmx) i można wyświetlać i edytować w Projektancie Entity Framework. Klasy, które możesz korzystać z aplikacji są generowane automatycznie z pliku EDMX.

Obejrzyj wideo

W tym przewodniku krok po kroku i wideo zawierają wprowadzenie do rozwoju pierwszego modelu używający narzędzia Entity Framework. Model umożliwia najpierw utworzyć nowy model przy użyciu programu Entity Framework Designer, a następnie wygenerować schemat bazy danych z modelem. Model jest przechowywany w pliku EDMX (z rozszerzeniem edmx) i można wyświetlać i edytować w Projektancie Entity Framework. Klasy, które możesz korzystać z aplikacji są generowane automatycznie z pliku EDMX.

Osoba prezentująca: [Rowan Miller](#)

Film wideo: [WMV](#) | [MP4](#) | [WMV \(ZIP\)](#)

Wymagania wstępne

Musisz mieć program Visual Studio 2010 lub Visual Studio 2012 są zainstalowane w tym przewodniku.

Jeśli używasz programu Visual Studio 2010, należy również mieć [NuGet](#) zainstalowane.

1. Tworzenie aplikacji

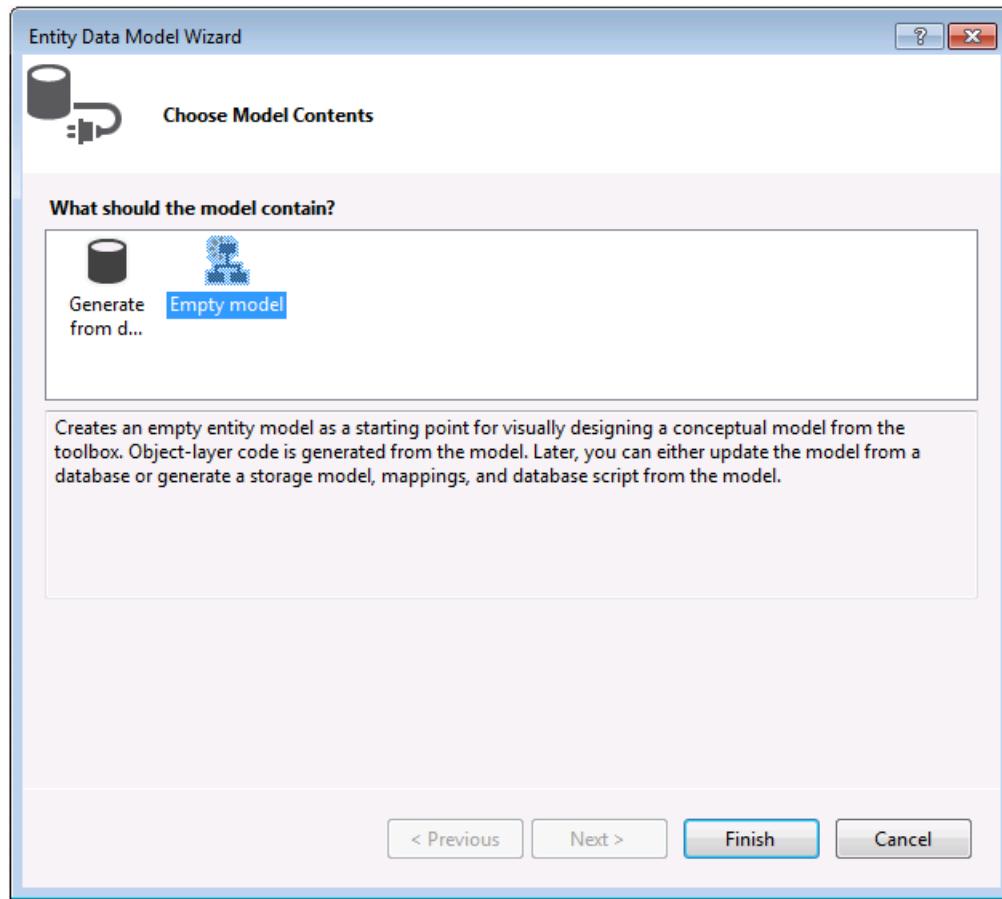
Aby zachować ich prostotę zamierzamy utworzyć aplikację konsoli podstawowe, która używa pierwszego modelu do dostępu do danych:

- Otwórz program Visual Studio
- **Plik —> New -> projektu...**
- Wybierz **Windows** menu po lewej stronie i **aplikacji konsoli**
- Wprowadź **ModelFirstSample** jako nazwę
- Wybierz **OK**

2. Tworzenie modelu

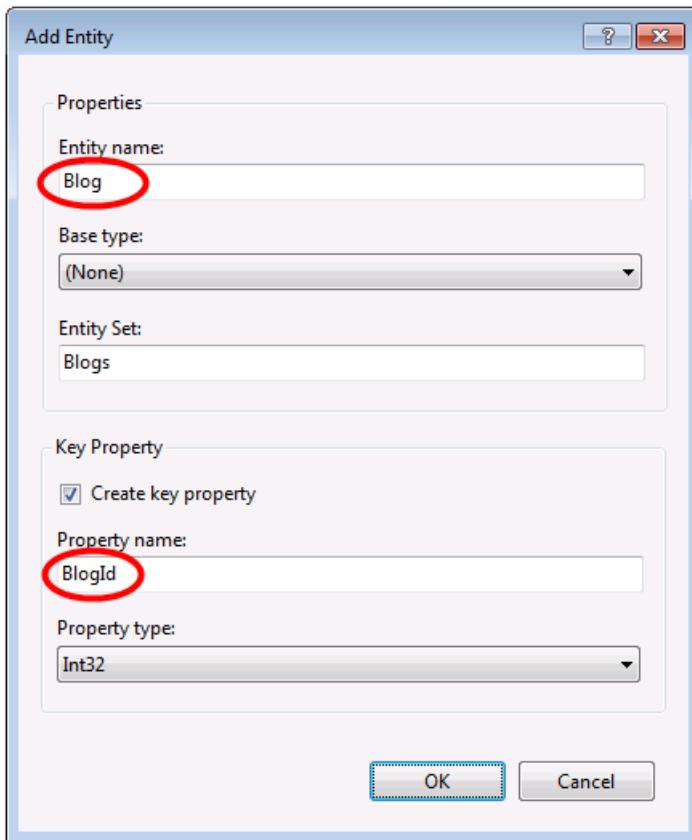
Zamierzamy korzystania z programu Entity Framework Designer, który wchodzi w skład programu Visual Studio, aby utworzyć nasz model.

- **Projekt —> Dodaj nowy element...**
- Wybierz **danych** menu po lewej stronie i następnie **ADO.NET Entity Data Model**
- Wprowadź **BloggingModel** jako nazwę i kliknij przycisk **OK**, zostanie uruchomiony Kreator modelu Entity Data Model
- Wybierz **pusty Model** i kliknij przycisk **Zakończ**



Entity Framework Designer jest otwierany przy użyciu modelu puste. Można teraz rozpoczęć dodawanie jednostek, właściwości i skojarzenia do modelu.

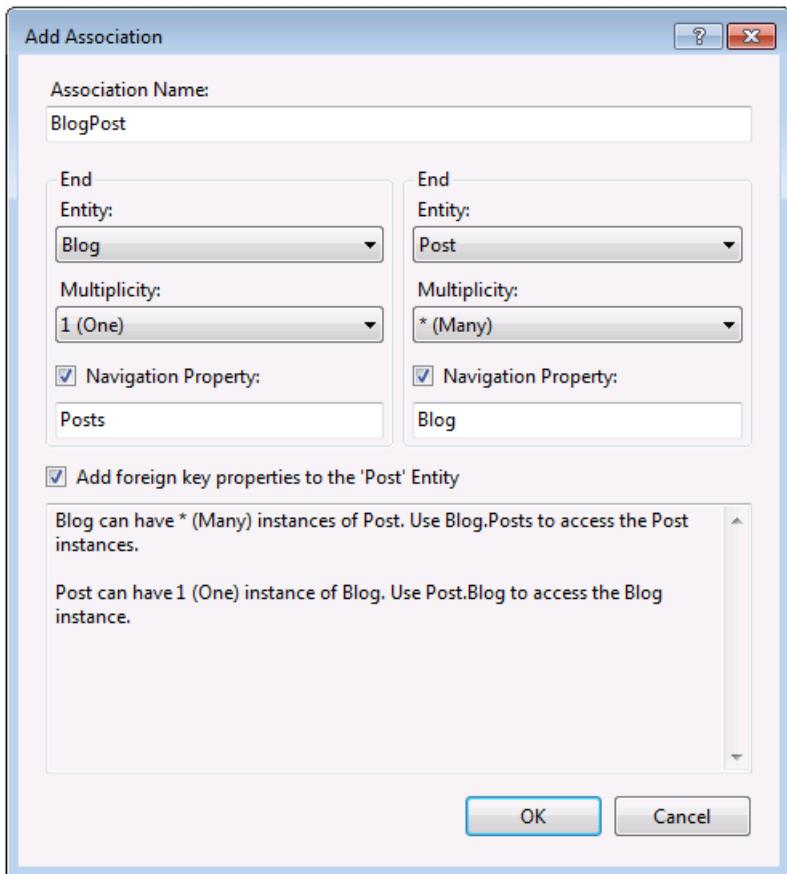
- Kliknij prawym przyciskiem myszy projekt powierzchni i wybierz **właściwości**
- Zmiana okna właściwości **nazwa kontenera jednostki** do **BloggingContext** jest to nazwa pochodnej kontekst, który zostanie wygenerowany dla Ciebie kontekstu reprezentuje sesję z bazą danych, dzięki czemu nam zapytania i Zapisz dane
- Kliknij prawym przyciskiem myszy projekt powierzchni i wybierz **Dodaj nowy -> jednostki...**
- Wprowadź **Blog** jako nazwę podmiotu i **BlogId** jako nazwę klucza i kliknij przycisk **OK**



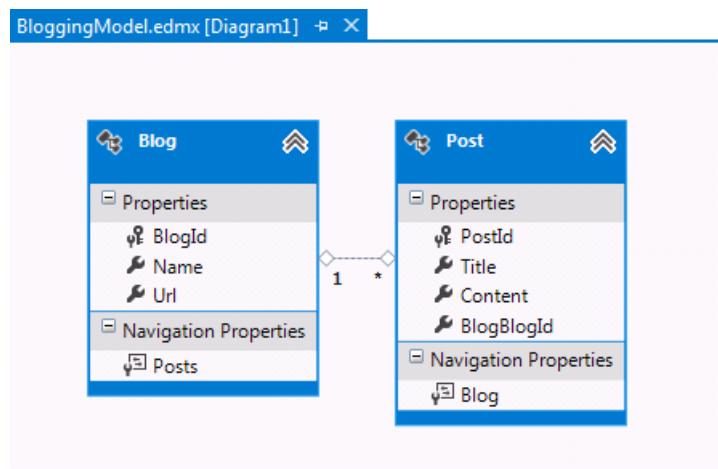
- Kliknij prawym przyciskiem myszy na nową jednostkę na projekt powierzchni i wybierz **Dodaj nowy -> właściwość skalarną**, wprowadź **nazwa** jako nazwę właściwości.
- Powtórz ten proces, aby dodać **adresu Url** właściwości.
- Kliknij prawym przyciskiem myszy **adresu Url** właściwość projekt powierzchni i wybierz **właściwości**, zmiana właściwości **Nullable** ustawienie **True** Dzięki temu można zapisać w blogu do bazy danych bez przypisywania go na adres Url
- Przy użyciu technik, możesz po prostu dzięki modelom uczenia, Dodaj **wpis** jednostki o **PostId** właściwości klucza
- Dodaj **tytuł** i **zawartości** właściwości skalarne można **wpis** jednostki

Skoro już mamy kilka jednostek, nadszedł czas na dodawanie skojarzenia (lub relacji) między nimi.

- Kliknij prawym przyciskiem myszy projekt powierzchni i wybierz **Dodaj nowy -> skojarzenie...**
- Wprowadzić jeden z końców relacji wskazując **Blog** z wartością liczebności równą **jeden** i innego punktu końcowego do **wpis** z wartością liczebności równą **wiele** Oznacza to, że blogu zawiera wiele wpisów i wpis należy do jednego Blog
- Upewnij się, **dodawania właściwości klucza obcego do jednostki "Post"** pole jest zaznaczone, a kliknij **OK**



W efekcie powstał prosty model, który możemy Generuj z bazy danych i umożliwia odczytywanie i zapisywanie danych.



Dodatkowe kroki w programie Visual Studio 2010

Jeśli pracujesz w programie Visual Studio 2010 istnieją pewne dodatkowe czynności, które należy wykonać uaktualnienie do najnowszej wersji programu Entity Framework. Uaktualnienie jest ważne, ponieważ daje ona dostęp do ulepszonej powierzchni interfejsu API, który jest znacznie łatwiejsze do użycia, a także najnowsze poprawki.

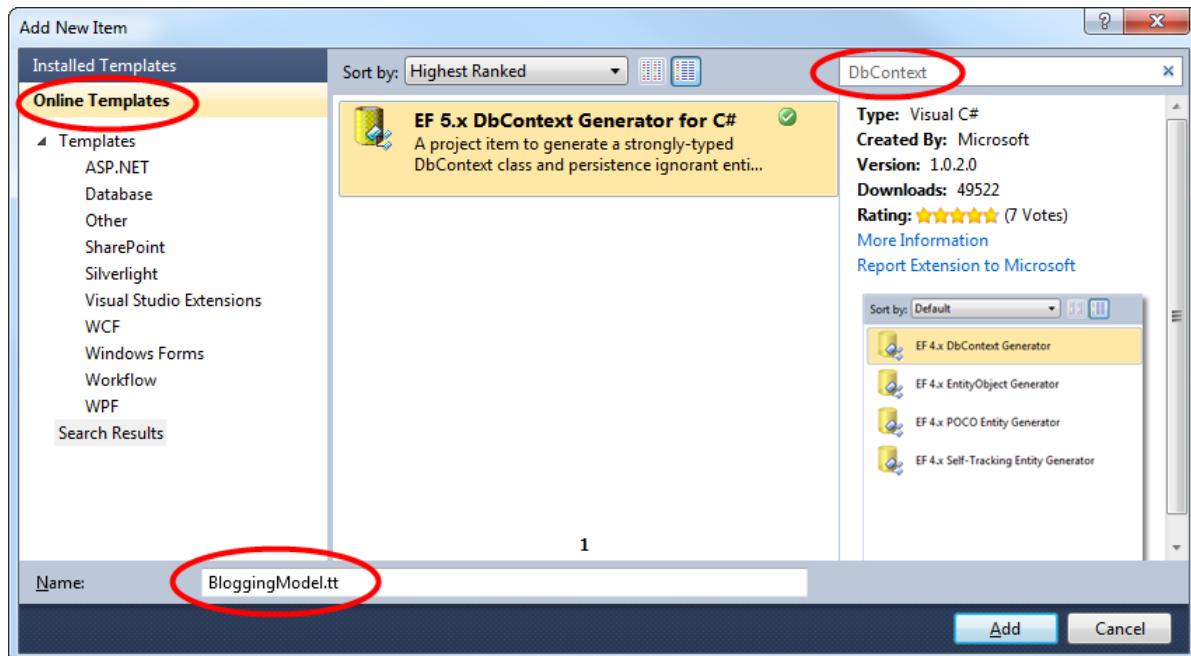
Najpierw up, musimy pobrać najnowszą wersję programu Entity Framework z NuGet.

- **Project —> Zarządzaj pakietami NuGet...** *Jeśli nie masz *Zarządzaj pakietami NuGet... **opcji, należy zainstalować najnowszej wersji pakietu NuGet*
- Wybierz **Online** kartę
- Wybierz **EntityFramework** pakietu
- Kliknij przycisk **instalacji**

Następnie należy zmienić nasz model, aby wygenerować kod, który korzysta z interfejsu API typu DbContext,

który został wprowadzony w nowszych wersjach programu Entity Framework.

- Kliknij prawym przyciskiem myszy na puste miejsce modelu w Projektancie platformy EF, a następnie wybierz pozycję **Dodaj element generowanie kodu...**
- Wybierz **szablonów Online** z menu po lewej stronie i wyszukaj **DbContext**
- Wybierz **EF 5.x Generator DbContext dla języka C#**, wprowadź **BloggingModel** jako nazwę i kliknij przycisk **Dodaj**



3. Generowanie bazy danych

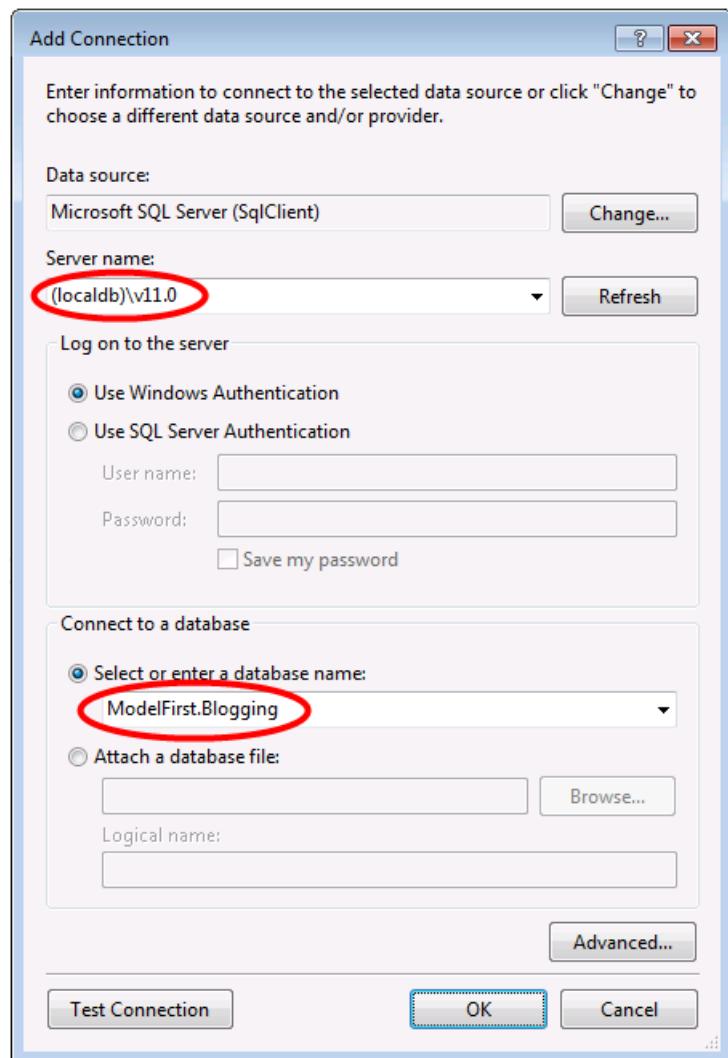
Biorąc pod uwagę nasz model, platformy Entity Framework można obliczyć schemat bazy danych, które pomogą nam do przechowywania i pobierania danych przy użyciu modelu.

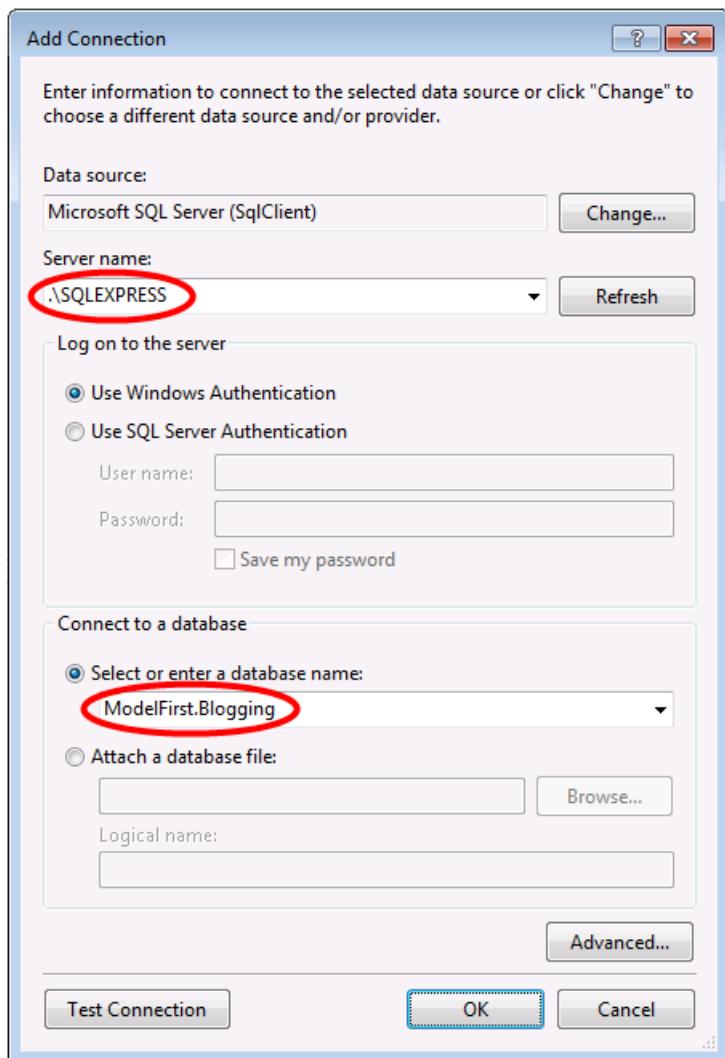
Serwer bazy danych, który został zainstalowany przy użyciu programu Visual Studio różni się zależnie od wersji programu Visual Studio zostały zainstalowane:

- Jeśli używasz programu Visual Studio 2010 zostanie utworzona baza danych programu SQL Express.
- Jeśli używasz programu Visual Studio 2012, a następnie zostanie utworzona **LocalDB** bazy danych.

Rozpoczniemy i wygenerować bazę danych.

- Kliknij prawym przyciskiem myszy projekt powierzchni i wybierz **Generuj z bazy danych z modelu...**
- Kliknij przycisk **nowe połączenie...** a następnie określ LocalDB lub SQL Express, w zależności od instalowanej wersji programu Visual Studio, którego używasz, wprowadź **ModelFirst.Blogging** jako nazwa bazy danych.



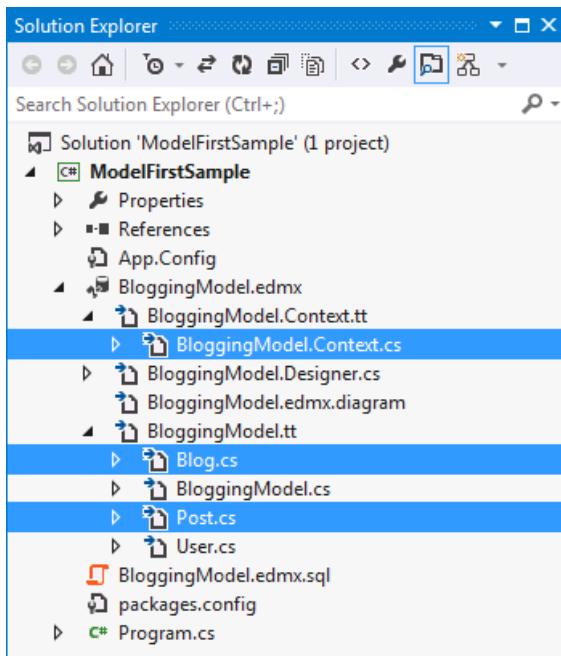


- Wybierz **OK** i uzyskasz, jeśli chcesz utworzyć nową bazę danych, wybierz opcję **tak**
- Wybierz **dalej** i Entity Framework Designer obliczy skryptu, aby utworzyć schemat bazy danych
- Gdy skrypt zostanie wyświetlona, kliknij przycisk **Zakończ** i skrypt zostanie dodany do projektu i otwarte
- Kliknij prawym przyciskiem myszy skrypt i wybierz pozycję **Execute**, zostanie wyświetlony monit, aby określić bazę danych, aby nawiązać połączenie, podaj LocalDB lub SQL Server Express, w zależności od instalowanej wersji programu Visual Studio, którego używasz

4. Odczytywanie i zapisywanie danych

Teraz, gdy model, nadszedł czas na potrzeby dostępu do niektórych danych. Klasy użyjemy na potrzeby dostępu do danych są generowane automatycznie na podstawie pliku EDMX.

Ten zrzut ekranu pochodzi z programu Visual Studio 2012, jeśli używasz programu Visual Studio 2010 *BloggingModel.tt* i *BloggingModel.Context.tt* plików będzie bezpośrednio w ramach projektu, a nie zagnieżdżony w pliku EDMX.



Implementuje metody Main w pliku Program.cs, jak pokazano poniżej. Ten kod tworzy nowe wystąpienie nasz kontekst i używa go do wstawiania nowego bloga. Następnie używa zapytania LINQ, aby pobrać wszystkie blogi z bazy danych uporządkowana w kolejności alfabetycznej według tytułu.

```
class Program
{
    static void Main(string[] args)
    {
        using (var db = new BloggingContext())
        {
            // Create and save a new Blog
            Console.Write("Enter a name for a new Blog: ");
            var name = Console.ReadLine();

            var blog = new Blog { Name = name };
            db.Blogs.Add(blog);
            db.SaveChanges();

            // Display all Blogs from the database
            var query = from b in db.Blogs
                        orderby b.Name
                        select b;

            Console.WriteLine("All blogs in the database:");
            foreach (var item in query)
            {
                Console.WriteLine(item.Name);
            }

            Console.WriteLine("Press any key to exit...");
            Console.ReadKey();
        }
    }
}
```

Możesz teraz uruchomić aplikację i ją przetestować.

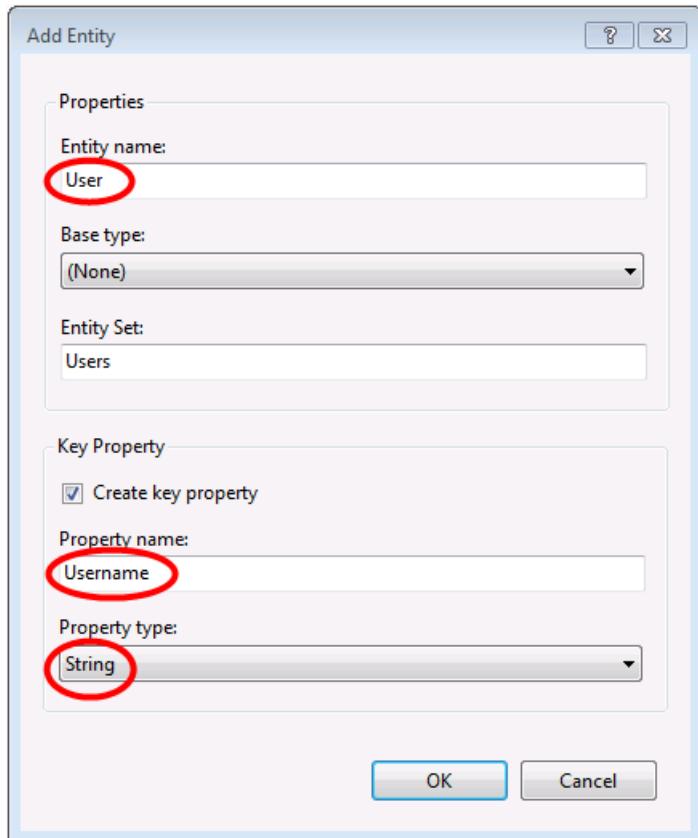
```
Enter a name for a new Blog: ADO.NET Blog
All blogs in the database:
ADO.NET Blog
Press any key to exit...
```

5. Obsługa zmiany modelu

Teraz nadszedł czas, aby wprowadzić pewne zmiany na nasz model, jeśli możemy wprowadzić te zmiany, należy również zaktualizować schemat bazy danych.

Rozpoczniemy przez dodanie nowego obiektu użytkownika w naszym modelu.

- Dodaj nową **użytkownika** nazwa jednostki za pomocą **Username** jako nazwę klucza i **ciąg** jako typ właściwości klucza



- Kliknij prawym przyciskiem myszy **Username** właściwość projekt powierzchni i wybierz **właściwości**, we właściwościach okna zmiany **MaxLength** ustawienie **50 ** To ogranicza dane, które mogą być przechowywane w nazwie użytkownika używana do 50 znaków
- Dodaj **DisplayName** właściwości skalarne **użytkownika** jednostki

W efekcie powstał aktualizowanego modelu i jesteśmy gotowi zaktualizować bazę danych, aby obsłużyć nasz nowy typ jednostki użytkownika.

- Kliknij prawym przyciskiem myszy projekt powierzchni i wybierz **Generuj z bazy danych z modelu...** , Platformy entity Framework obliczy skrypt, aby ponownie utworzyć schemat oparty na aktualizowanego modelu.
- Kliknij przycisk **Zakończ**
- Möže odbierać ostrzeżeń o zastąpienie istniejącego skryptu języka DDL i mapowania i magazynu części modelu, kliknij przycisk **tak** dla obu tych ostrzeżeń
- Zaktualizowano skrypt SQL w celu utworzenia bazy danych jest otwarty
Skrypt, który jest generowany będzie porzucić wszystkie istniejące tabele i następnie ponownie Utwórz schemat od podstaw. To może działać na potrzeby lokalnego programowania, ale nie jest wygodną wypychania zmian do bazy danych, która została już wdrożona. Jeśli potrzebujesz opublikować zmiany w bazie danych, która została już wdrożona, konieczne będzie edytowanie skryptu lub użyj narzędzia do porównywania schematu do obliczania skryptów migracji.
- Kliknij prawym przyciskiem myszy skrypt i wybierz pozycję **Execute**, zostanie wyświetlony monit, aby określić

bazę danych, aby nawiązać połączenie, podaj LocalDB lub SQL Server Express, w zależności od instalowanej wersji programu Visual Studio, którego używasz

Podsumowanie

W tym przewodniku, który przyjrzaliśmy się pierwszy Model programowania, które umożliwiło nam Tworzenie modelu w Projektancie platformy EF, a następnie wygenerować bazę danych z tego modelu. Następnie użyliśmy modelu do odczytu i zapisu pewne dane z bazy danych. Na koniec mamy zaktualizowany modelu i tworzony ponownie schematu bazy danych, zgodny z modelem.

Najpierw bazy danych

27.09.2018 • 11 minutes to read • [Edit Online](#)

W tym przewodniku krok po kroku i wideo zawierają wprowadzenie do tworzenia pierwszej bazy danych przy użyciu platformy Entity Framework. Baza danych najpierw można odtwarzać modelu z istniejącej bazy danych. Model jest przechowywany w pliku EDMX (z rozszerzeniem edmx) i można wyświetlać i edytować w Projektancie Entity Framework. Klasy, które możesz korzystać z aplikacji są generowane automatycznie z pliku EDMX.

Obejrzyj wideo

To wideo zawiera wprowadzenie do tworzenia pierwszej bazy danych przy użyciu platformy Entity Framework. Baza danych najpierw można odtwarzać modelu z istniejącej bazy danych. Model jest przechowywany w pliku EDMX (z rozszerzeniem edmx) i można wyświetlać i edytować w Projektancie Entity Framework. Klasy, które możesz korzystać z aplikacji są generowane automatycznie z pliku EDMX.

Osoba prezentująca: [Rowan Miller](#)

Film wideo: [WMV](#) | [MP4](#) | [WMV \(ZIP\)](#)

Wymagania wstępne

Musisz mieć co najmniej programu Visual Studio 2010 lub Visual Studio 2012 są zainstalowane w tym przewodniku.

Jeśli używasz programu Visual Studio 2010, należy również mieć [NuGet](#) zainstalowane.

1. Utwórz istniejącej bazy danych

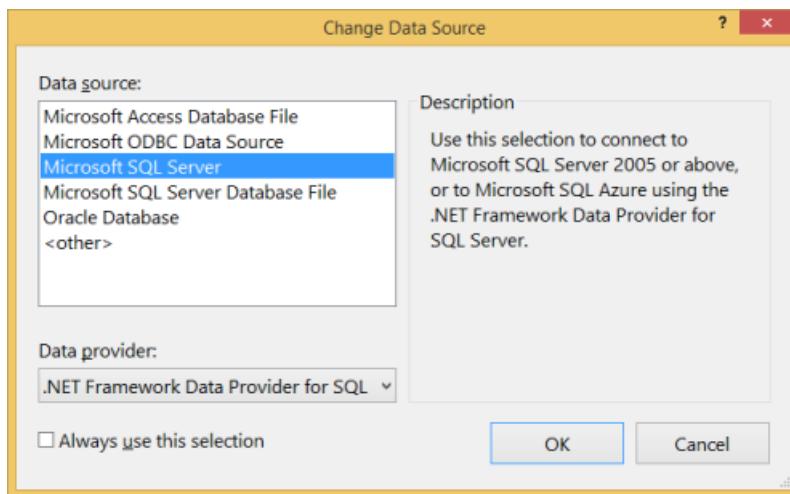
Zazwyczaj przeznaczonych do istniejącej bazy danych, zostanie już utworzony, ale w tym przewodniku należy utworzyć bazy danych w celu uzyskania dostępu.

Serwer bazy danych, który został zainstalowany przy użyciu programu Visual Studio różni się zależnie od wersji programu Visual Studio zostały zainstalowane:

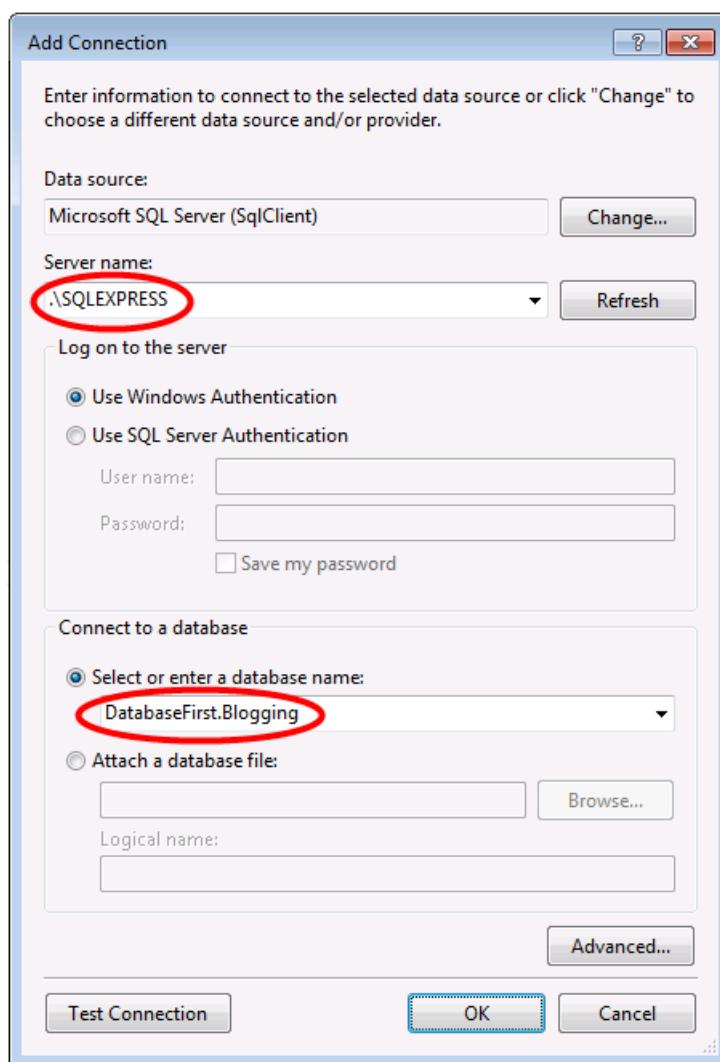
- Jeśli używasz programu Visual Studio 2010 zostanie utworzona baza danych programu SQL Express.
- Jeśli używasz programu Visual Studio 2012, a następnie zostanie utworzona [LocalDB](#) bazy danych.

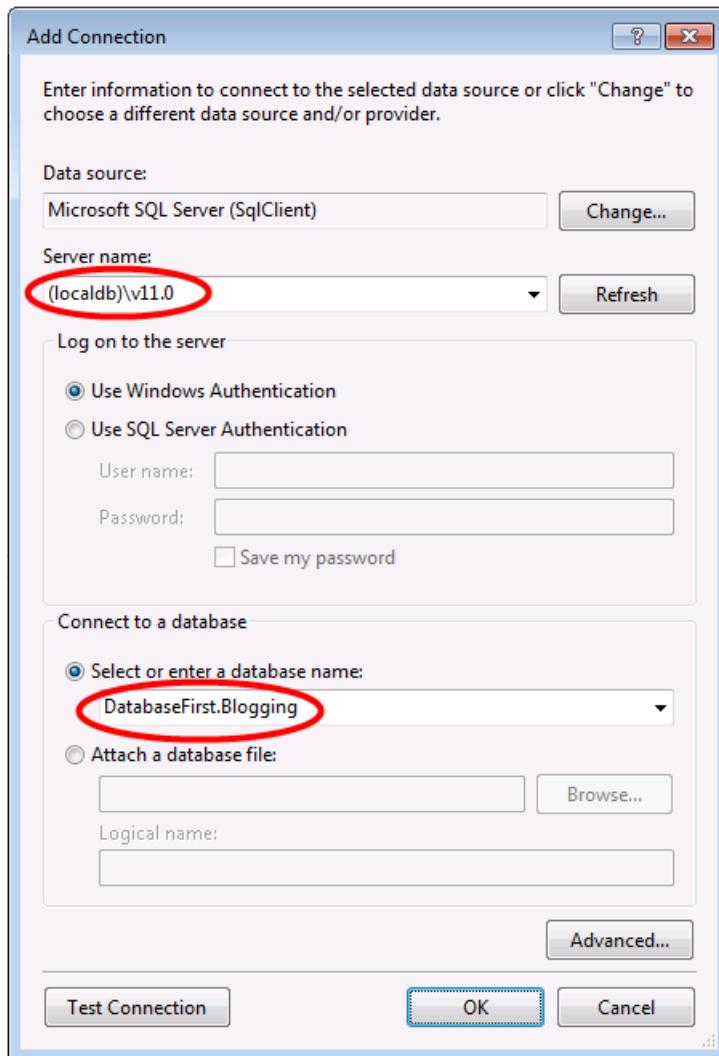
Rozpoczniemy i wygenerować bazę danych.

- Otwórz program Visual Studio
- **Widok —> Eksploratora serwera**
- Kliknij prawym przyciskiem myszy **połączeń danych** -> **Dodaj połączenie...**
- Jeśli nie jest połączona z bazą danych za pomocą Eksploratora serwera przed, musisz wybrać programu Microsoft SQL Server jako źródło danych



- Łączenie się z LocalDB lub SQL Express, w zależności od tego, który z nich został zainstalowany, a następnie wprowadź **DatabaseFirst.Blogging** jako nazwa bazy danych





- Wybierz **OK** i uzyskasz, jeśli chcesz utworzyć nową bazę danych, wybierz opcję **tak**



- Nowe bazy danych będą teraz wyświetlane w Eksploratorze serwera, kliknij prawym przyciskiem myszy na nim i wybierz **nowe zapytanie**
- Skopiuj następujące instrukcje SQL do nowego zapytania, a następnie kliknij prawym przyciskiem myszy, zapytania i wybierz pozycję **wykonania**

```

CREATE TABLE [dbo].[Blogs] (
    [BlogId] INT IDENTITY (1, 1) NOT NULL,
    [Name] NVARCHAR (200) NULL,
    [Url] NVARCHAR (200) NULL,
    CONSTRAINT [PK_dbo.Blogs] PRIMARY KEY CLUSTERED ([BlogId] ASC)
);

CREATE TABLE [dbo].[Posts] (
    [PostId] INT IDENTITY (1, 1) NOT NULL,
    [Title] NVARCHAR (200) NULL,
    [Content] NTEXT NULL,
    [BlogId] INT NOT NULL,
    CONSTRAINT [PK_dbo.Posts] PRIMARY KEY CLUSTERED ([PostId] ASC),
    CONSTRAINT [FK_dbo.Posts_dbo.Blogs_BlogId] FOREIGN KEY ([BlogId]) REFERENCES [dbo].[Blogs] ([BlogId]) ON
DELETE CASCADE
);

```

2. Tworzenie aplikacji

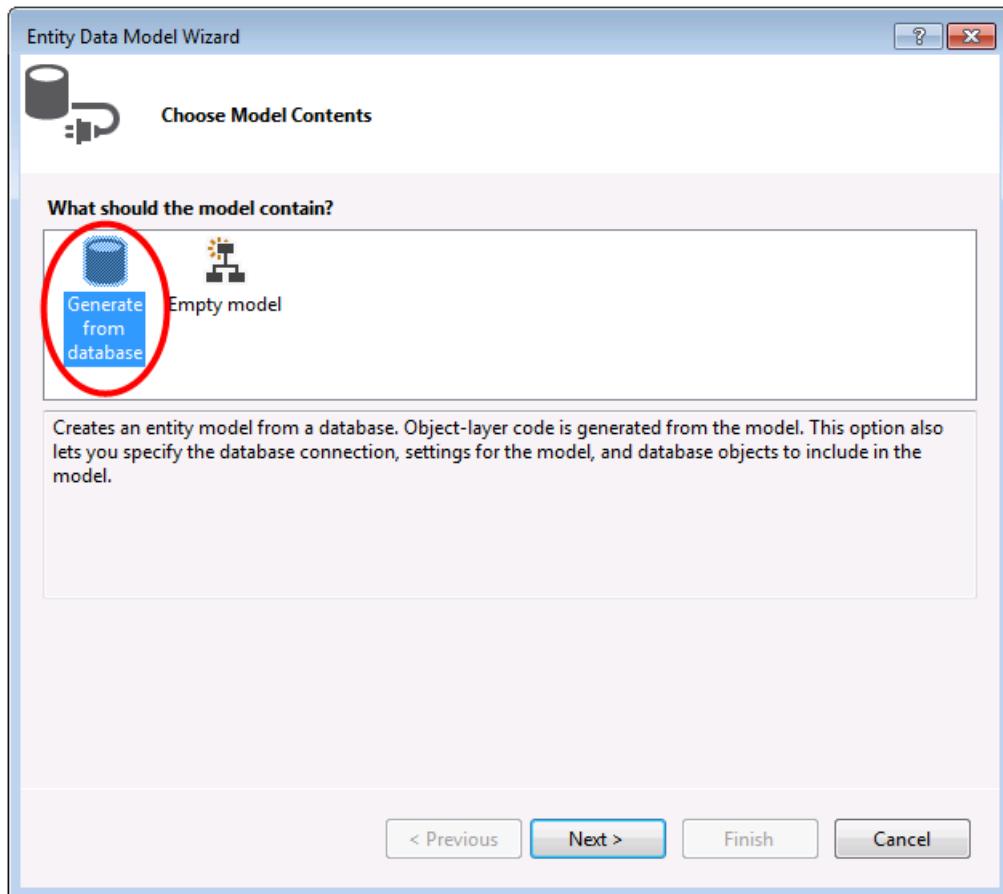
Aby zachować ich prostotę zamierzamy utworzyć aplikację konsoli podstawowe, która używa pierwszej bazy danych do dostępu do danych:

- Otwórz program Visual Studio
- **Plik —> New -> projektu...**
- Wybierz **Windows** menu po lewej stronie i **aplikacji konsoli**
- Wprowadź **DatabaseFirstSample** jako nazwę
- Wybierz **OK**

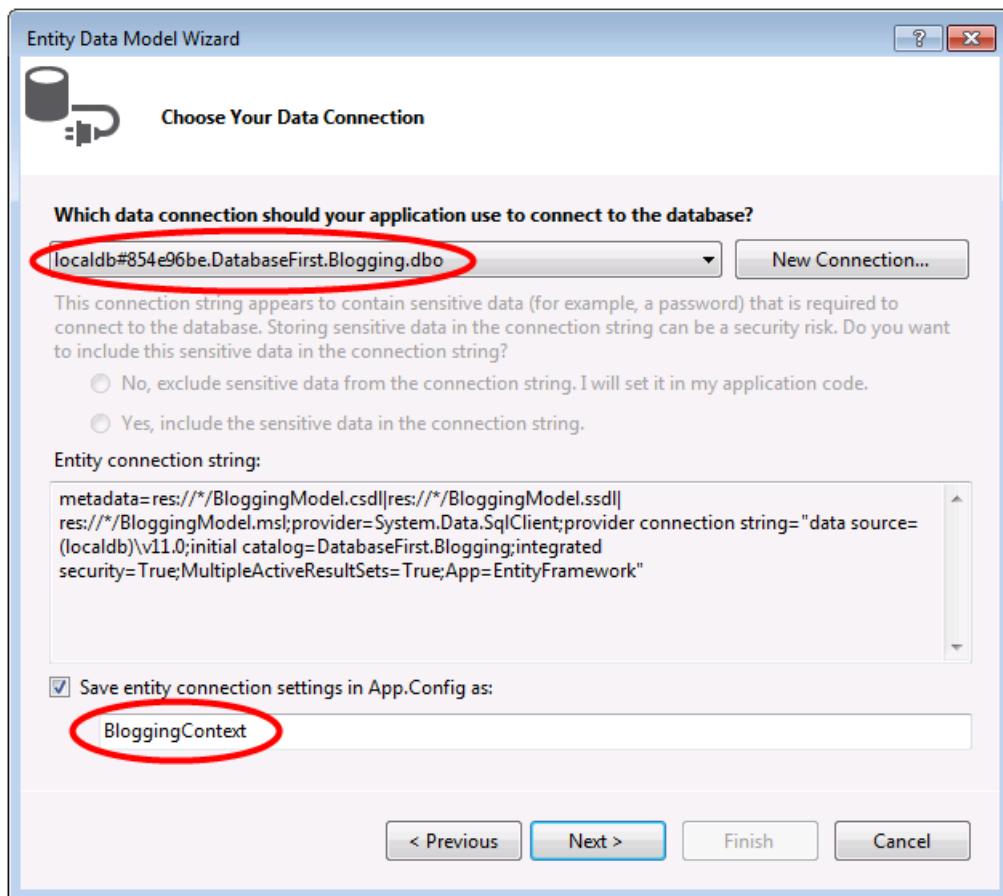
3. Model odtwarzania

Zamierzamy korzystania z programu Entity Framework Designer, który wchodzi w skład programu Visual Studio, aby utworzyć nasz model.

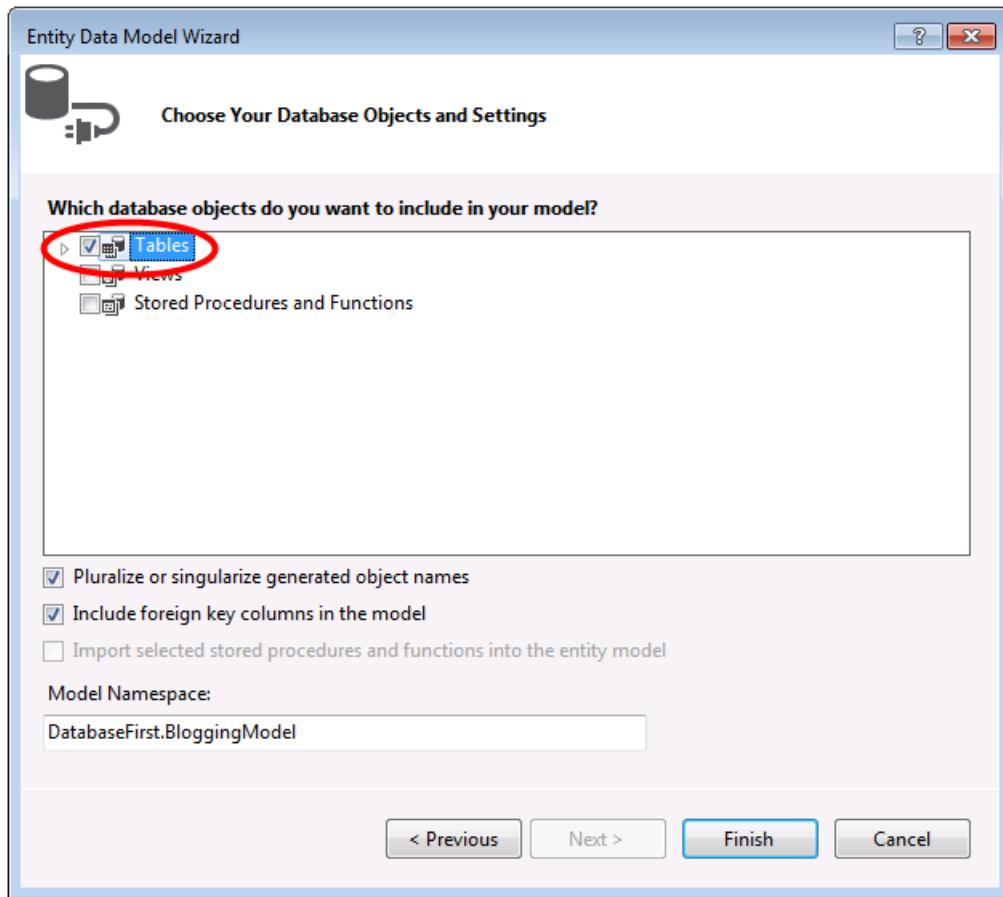
- **Projekt —> Dodaj nowy element...**
- Wybierz **danych** menu po lewej stronie i następnie **ADO.NET Entity Data Model**
- Wprowadź **BloggingModel** jako nazwę i kliknij przycisk **OK**
- Spowoduje to uruchomienie **Kreator modelu Entity Data Model**
- Wybierz **Generuj z bazy danych** i kliknij przycisk **dalej**



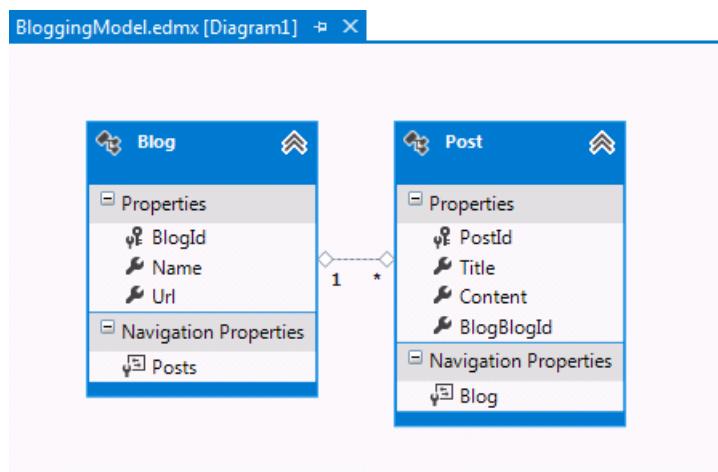
- Wybierz połączenie do bazy danych utworzonej w pierwszej sekcji, wprowadź **BloggingContext** jako nazwa parametrów połączenia i kliknij przycisk **dalej**



- Kliknij pole wyboru obok "Tabele", aby zaimportować wszystkie tabele i kliknij przycisk "Zakończ"



Po zakończeniu procesu odtwarzania nowy model jest dodawane do projektu i otworzona w celu wyświetlania w Projektancie Entity Framework. Dodano również pliku App.config do projektu przy użyciu szczegółów połączenia dla bazy danych.



Dodatkowe kroki w programie Visual Studio 2010

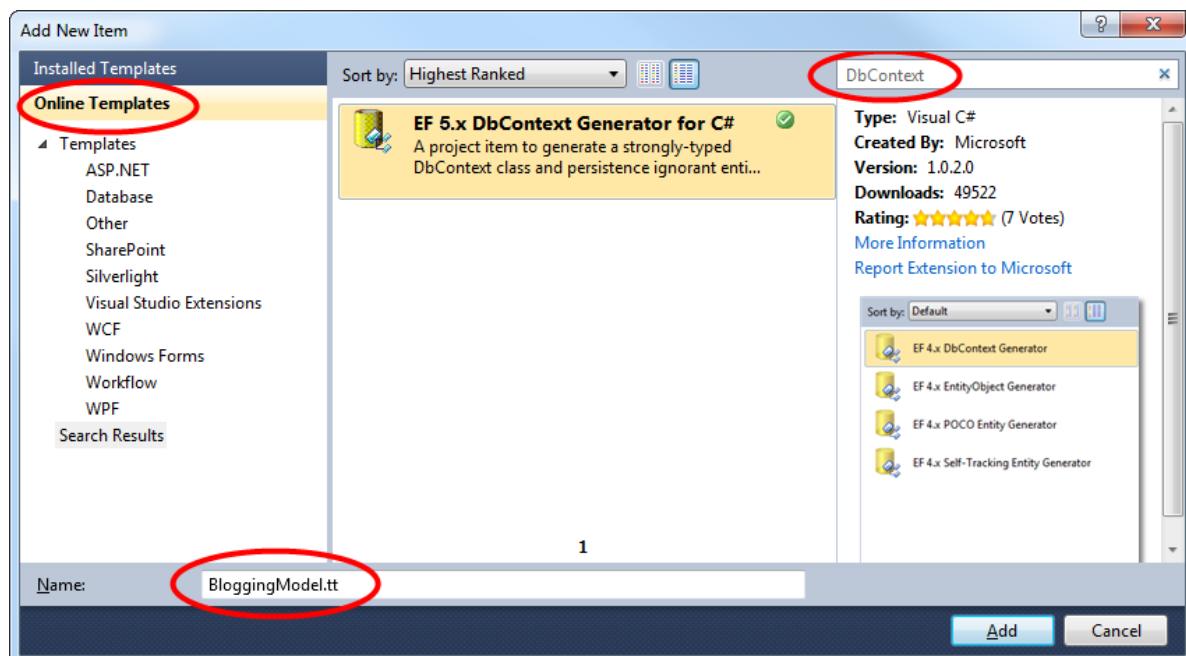
Jeśli pracujesz w programie Visual Studio 2010 istnieją pewne dodatkowe czynności, które należy wykonać uaktualnienie do najnowszej wersji programu Entity Framework. Uaktualnienie jest ważne, ponieważ daje ona dostęp do ulepszonej powierzchni interfejsu API, który jest znacznie łatwiejsze do użycia, a także najnowsze poprawki.

Najpierw up, musimy pobrać najnowszą wersję programu Entity Framework z NuGet.

- **Project —> Zarządzaj pakietami NuGet...** *Jeśli nie masz *Zarządzaj pakietami NuGet... ** opcji, należy zainstalować najnowszej wersji pakietu NuGet*
- Wybierz **Online** kartę
- Wybierz **EntityFramework** pakietu
- Kliknij przycisk **instalacji**

Następnie należy zamienić nasz model, aby wygenerować kod, który korzysta z interfejsu API typu DbContext, który został wprowadzony w nowszych wersjach programu Entity Framework.

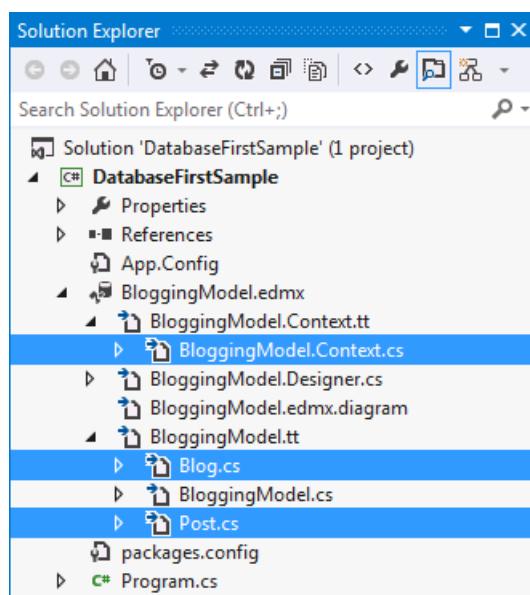
- Kliknij prawym przyciskiem myszy na puste miejsce modelu w Projektancie platformy EF, a następnie wybierz pozycję **Dodaj element generowanie kodu...**
- Wybierz **szablonów Online** z menu po lewej stronie i wyszukaj **DbContext**
- Wybierz **EF 5.x Generator DbContext dla języka C#**, wprowadź **BloggingModel** jako nazwę i kliknij przycisk **Dodaj**



4. Odczytywanie i zapisywanie danych

Teraz, gdy model, nadszedł czas na potrzeby dostępu do niektórych danych. Klasy użyjemy na potrzeby dostępu do danych są generowane automatycznie na podstawie pliku EDMX.

Ten zrzut ekranu pochodzi z programu Visual Studio 2012, jeśli używasz programu Visual Studio 2010 BloggingModel.tt i BloggingModel.Context.tt plików będzie bezpośrednio w ramach projektu, a nie zagnieżdżony w pliku EDMX.



Implementuje metody Main w pliku Program.cs, jak pokazano poniżej. Ten kod tworzy nowe wystąpienie nasz kontekst i używa go do wstawiania nowego bloga. Następnie używa zapytania LINQ, aby pobrać wszystkie blogi z

bazy danych uporządkowana w kolejności alfabetycznej według tytułu.

```
class Program
{
    static void Main(string[] args)
    {
        using (var db = new BloggingContext())
        {
            // Create and save a new Blog
            Console.Write("Enter a name for a new Blog: ");
            var name = Console.ReadLine();

            var blog = new Blog { Name = name };
            db.Blogs.Add(blog);
            db.SaveChanges();

            // Display all Blogs from the database
            var query = from b in db.Blogs
                        orderby b.Name
                        select b;

            Console.WriteLine("All blogs in the database:");
            foreach (var item in query)
            {
                Console.WriteLine(item.Name);
            }

            Console.WriteLine("Press any key to exit...");
            Console.ReadKey();
        }
    }
}
```

Możesz teraz uruchomić aplikację i ją przetestować.

```
Enter a name for a new Blog: ADO.NET Blog
All blogs in the database:
ADO.NET Blog
Press any key to exit...
```

5. Zajmowanie się zmian w bazie danych

Teraz nadszedł czas, aby wprowadzić pewne zmiany na naszych schemat bazy danych, gdy firma Microsoft wprowadza te zmiany, należy również zaktualizować nasz model, aby odzwierciedlać wprowadzone zmiany.

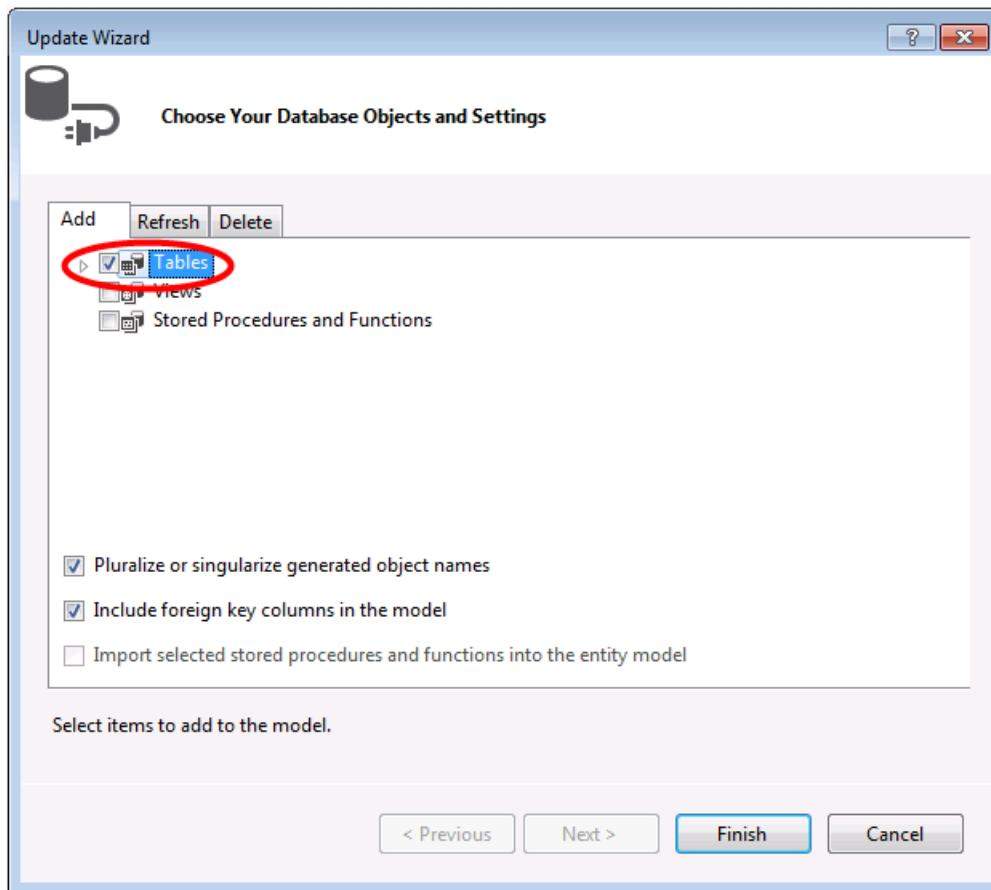
Pierwszym krokiem jest wprowadzić pewne zmiany schematu bazy danych. Zamierzamy dodać tabelę użytkowników ze schematem.

- Kliknij prawym przyciskiem myszy **DatabaseFirst.Blogging** bazy danych w Eksploratorze serwera i wybierz **nowe zapytanie**
- Skopiuj następujące instrukcje SQL do nowego zapytania, a następnie kliknij prawym przyciskiem myszy, zapytania i wybierz pozycję **wykonania**

```
CREATE TABLE [dbo].[Users]
(
    [Username] NVARCHAR(50) NOT NULL PRIMARY KEY,
    [DisplayName] NVARCHAR(MAX) NULL
)
```

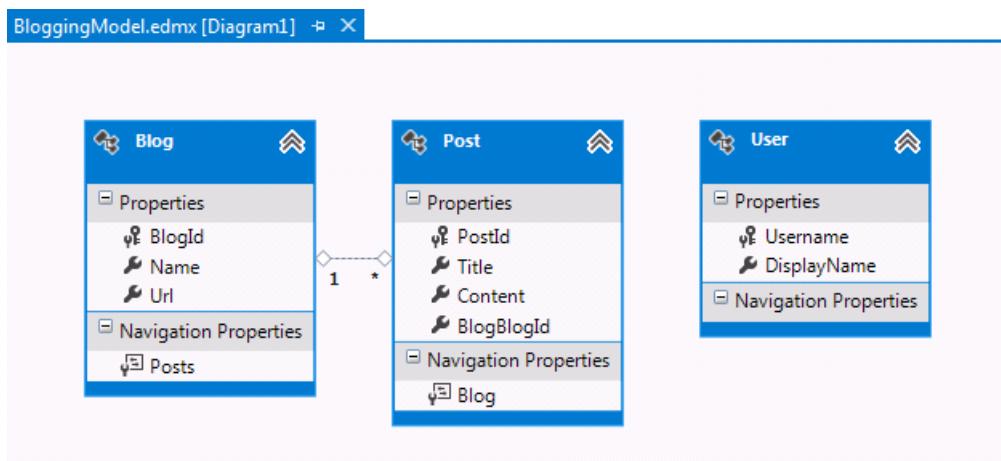
Teraz, gdy schemat jest aktualizowany, nadszedł czas na aktualizowanie modelu z tymi zmianami.

- Kliknij prawym przyciskiem myszy na puste miejsce w modelu w Projektancie platformy EF i wybierz opcję "Aktualizuj modelu z bazy danych...", co spowoduje uruchomienie Kreatora aktualizacji
- Na karcie Dodaj Kreatora aktualizacji zaznacz pole wyboru obok tabel oznacza to, że chcemy dodać nowe tabele ze schematu. *Na karcie odświeżania pokazuje wszystkie istniejące tabele w modelu, który będzie sprawdzany zmian podczas aktualizacji. Usuń kartach wszystkie tabele zostały usunięte ze schematu, które zostaną także usunięte z modelu, w ramach aktualizacji. Informacje dotyczące tych dwóch kart jest wykrywany automatycznie i jest dostępne wyłącznie do celów informacyjnych, nie można zmienić dowolne ustawienia.*



- Kliknij przycisk Zakończ w Kreatorze aktualizacji

Model został zaktualizowany do uwzględnienia nowej jednostki użytkownika, który jest mapowany do tabeli użytkowników, którą dodaliśmy do bazy danych.



Podsumowanie

W tym przewodniku, który przyjrzaliśmy się tworzeniu pierwszej bazy danych, które umożliwiło nam Tworzenie

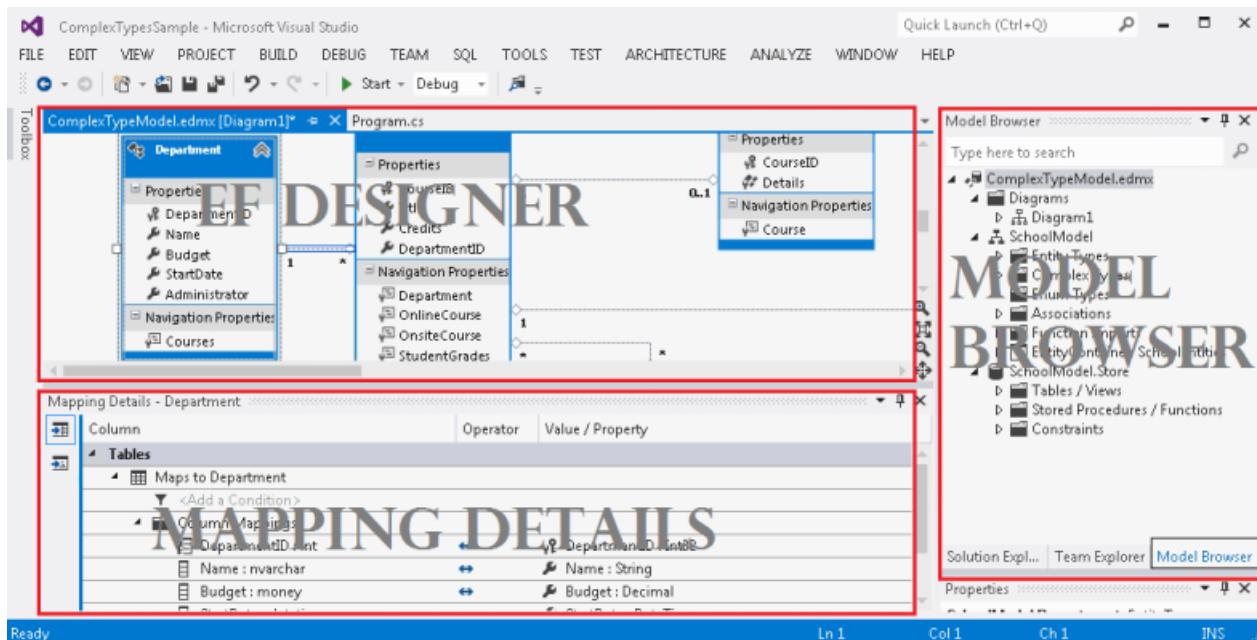
modelu w Projektancie platformy EF, w oparciu o istniejącą bazę danych. Następnie użyliśmy tego modelu do odczytu i zapisu pewne dane z bazy danych. Na koniec Zaktualizowaliśmy modelu aby odzwierciedlić zmiany wprowadzone do schematu bazy danych.

Typy złożone - projektancie platformy EF

13.09.2018 • 12 minutes to read • [Edit Online](#)

W tym temacie pokazano, jak do mapowania typów złożonych z Entity Framework Designer (Projektant EF) oraz sposób wykonywania zapytań w przypadku jednostek, które zawierają właściwości typu złożonego.

Na poniżej ilustracji przedstawiono główne systemu windows, które są używane podczas pracy z projektancie platformy EF.



NOTE

Podczas tworzenia modelu koncepcyjnego ostrzeżenia dotyczące podmiotów niezmapowanych i skojarzenia może pojawić się na liście błędów. Można zignorować te ostrzeżenia, ponieważ po dokonaniu wyboru wygenerować bazę danych z modelem, błędy znikną.

Co to jest typem złożonym

Typy złożone są właściwościami nieskalarnego typów jednostek, które umożliwiają właściwości skalarne, które mają być organizowane w ramach jednostek. Podobnie jak jednostki typy złożone składają się z właściwości skalarne właściwości lub innego typu złożonego.

Podczas pracy z obiektami, które reprezentują typy złożone, należy pamiętać o następujących czynności:

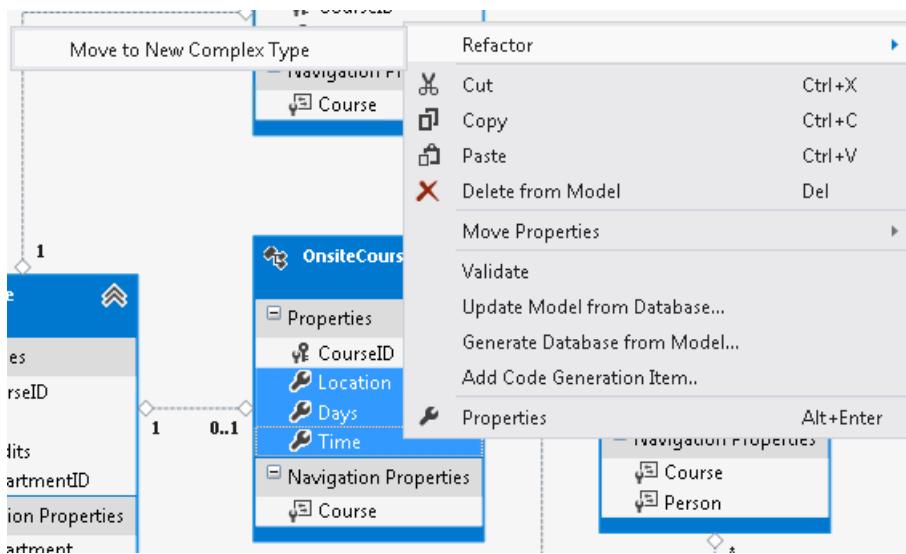
- Typy złożone nie mają kluczy i dlatego nie mogą istnieć niezależnie. Typy złożone mogą istnieć tylko jako właściwości typów jednostek lub innych typów złożonych.
- Typy złożone nie mogą uczestniczyć w skojarzeniach i nie mogą zawierać właściwości nawigacji.
- Właściwości typu złożonego nie mogą być **null**. **InvalidOperationException** występuje, gdy **DbContext.SaveChanges** nosi nazwę i obiekt złożony null zostanie osiągnięty. Właściwości skalarne obiektów złożonych może być **null**.
- Typy złożone nie mogą dziedziczyć z innych typów złożonych.
- Należy zdefiniować typ złożony jako **klasy**.
- EF wykrywa zmiany do elementów członkowskich w obiekcie typu złożonego przy

DbContext.DetectChanges jest wywoływana. Wywołuje platformy Entity Framework **metody DetectChanges** automatycznie, kiedy są wywoływanie następujące składowe: **DbSet.Find**, **DbSet.Local**, **DbSet.Remove**, **DbSet.Add**, **DbSet.Attach**, **DbContext.SaveChanges**, **DbContext.GetValidationErrors**, **DbContext.Entry**, **DbChangeTracker.Entries**.

Refaktoryzacji właściwości jednostki na nowy typ złożony

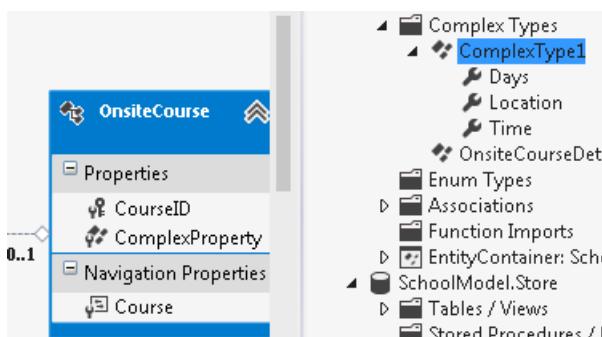
Jeśli masz już jednostki w modelu koncepcyjnego może zajść potrzeba refaktoryzacji niektórych właściwości na właściwość typu złożonego.

Na powierzchni projektowej, wybierz jeden lub więcej właściwości (z wyjątkiem właściwości nawigacji) jednostki, następnie kliknij prawym przyciskiem myszy i wybierz **Refaktoryzuj -> przenieś do nowego typu złożonego**.



Nowy typ złożony z wybranymi właściwościami zostanie dodany do **przeglądarka modeli**. Typ złożony jest nadawana nazwa domyślna.

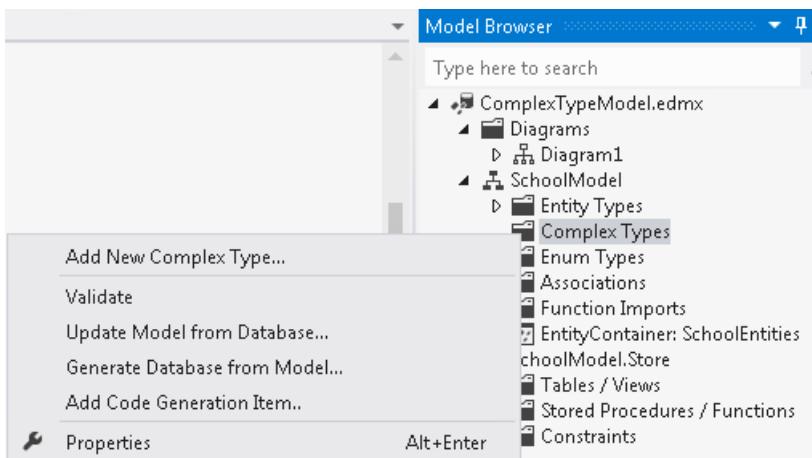
Właściwości złożonej typu nowo utworzony zastępuje wybranych właściwości. Wszystkie mapowania właściwości są zachowywane.



Utwórz nowy typ złożony

Można również utworzyć nowy typ złożony, który nie zawiera właściwości istniejącej jednostki.

Kliknij prawym przyciskiem myszy **typy złożone** folderu w przeglądarce modelu wskaź **typu złożonego działając funkcję AddNew...** . Alternatywnie, można wybrać **typy złożone** folder, a następnie naciśnij klawisz **Wstaw** kluczowe na klawiaturze.



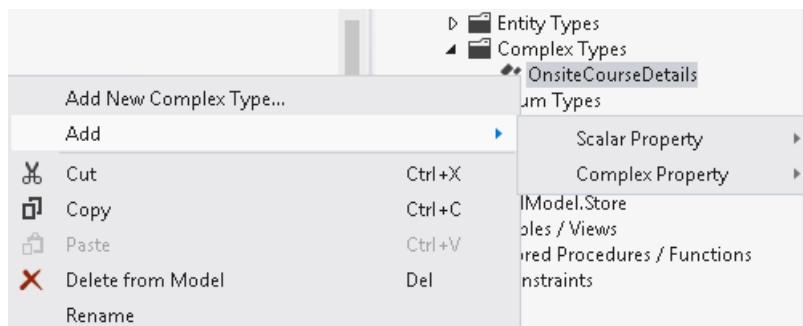
Nowy typ złożony jest dodawany do folderu o domyślnej nazwie. Teraz można dodać właściwości do typu.

Dodaj właściwości do typu złożonego

Typy skalarne lub istniejące typy złożone, może być właściwości typu złożonego. Jednak właściwości typu złożonego nie może mieć odwołania cykliczne. Na przykład typ złożony **OnsiteCourseDetails** nie może mieć właściwość typu złożonego **OnsiteCourseDetails**.

Można dodać właściwość, typ złożony sposoby wymienione poniżej.

- Kliknij prawym przyciskiem myszy typ złożony w przeglądarce modelu, wskaz opcję **Dodaj**, następnie wskaz **właściwość skalarną** lub **właściwość typu złożonego**, następnie wybierz typ żądanej właściwości.
Alternatywnie można wybrać typ złożony, a następnie naciśnij klawisz **Wstaw** kluczowe na klawiaturze.



Nowa właściwość jest dodawany do typu złożonego o domyślnej nazwie.

- LUB —
- Kliknij prawym przyciskiem myszy właściwość jednostek **projektancie platformy EF** powierzchni i wybierz **kopiowania**, kliknij prawym przyciskiem myszy typ złożony, w **przeglądarka modeli** i wybierz **Wklej**.

Zmień nazwę typu złożonego

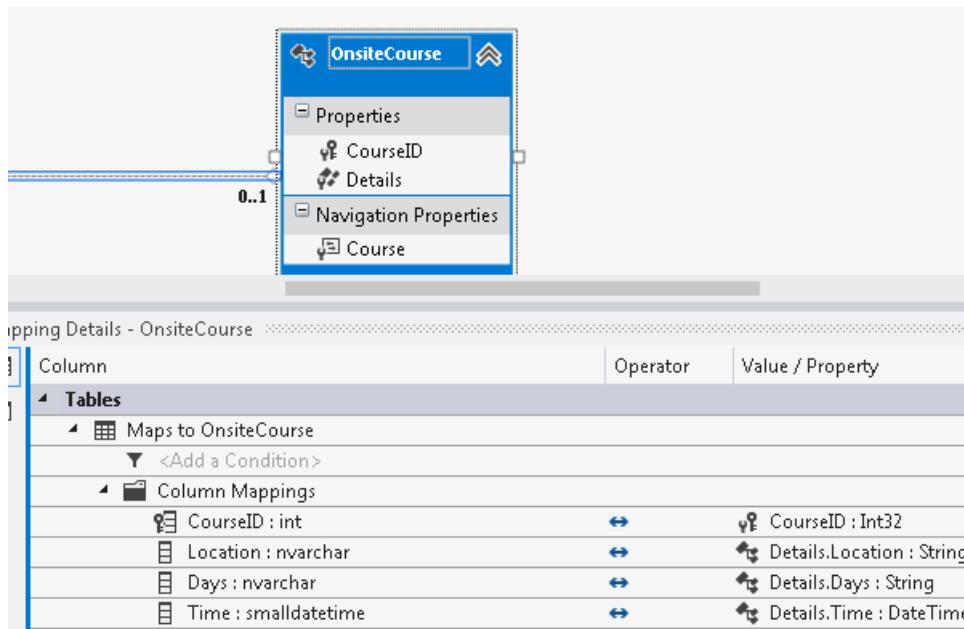
Po zmianie nazwy typu złożonego, wszystkie odwołania do typu są aktualizowane w całym projekcie.

- Wolno kliknij dwukrotnie typu złożonego w **przeglądarka modeli**. Nazwa zostanie wybrana i w trybie edycji.
- LUB —
- Kliknij prawym przyciskiem myszy typ złożony, w **przeglądarka modeli** i wybierz **Zmień nazwę**.
- LUB —
- Wybierz typ złożony w przeglądarce modelu, a następnie naciśnij klawisz F2.

- LUB —
- Kliknij prawym przyciskiem myszy typ złożony, w **przeglądarka modeli** i wybierz **właściwości**. Edytuj nazwę w **właściwości** okna.

Dodawanie istniejącego typu złożonego do jednostki i zamapuj jego właściwości do kolumn tabeli

1. Kliknij prawym przyciskiem myszy jednostkę, wskaż opcję **Dodaj nowej** wybierz **właściwość typu złożonego**. Właściwość typu złożonego o domyślnej nazwie zostanie dodany do jednostki. Domyślny typ (masz wybrane z istniejących typów złożonych) jest przypisywane do właściwości.
2. Przypisz żądany typ właściwości w **właściwości** okna. Po dodaniu właściwość typu złożonego do jednostki, jego właściwości musi być mapowane na kolumny w tabeli.
3. Kliknij prawym przyciskiem myszy typ jednostki na powierzchni projektowej lub w **przeglądarka modeli** i wybierz **mapowania tabel**. Mapowania tabel są wyświetlane w **szczegółowy mapowania** okna.
4. Rozwiń **mapuje <nazwy tabeli>** węzła. A **mapowania kolumn** węzeł jest dostępny.
5. Rozwiń **mapowania kolumn** węzła. Zostanie wyświetlona lista wszystkich kolumn w tabeli. Domyślne właściwości (jeśli istnieje) który mapowania kolumn są wyświetlane w obszarze **wartość/właściwości** nagłówka.
6. Wybierz kolumnę, którą chcesz zamapować, a następnie kliknij prawym przyciskiem myszy odpowiedni **wartość/właściwości** pola. Jest wyświetlana lista rozwijana lista właściwości skalarne.
7. Wybierz odpowiednie właściwości.



8. Powtórz kroki 6 i 7 dla każdej kolumny.

NOTE

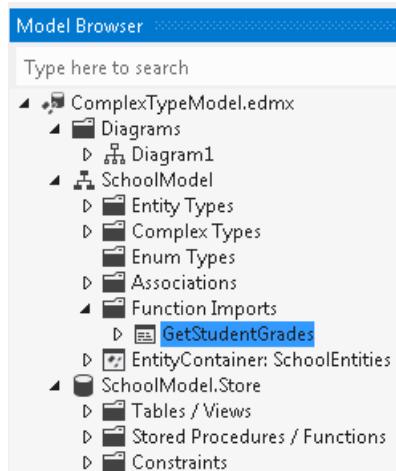
Aby usunąć mapowanie kolumn, wybierz kolumnę, którą chcesz zamapować, a następnie kliknij przycisk **wartość/właściwości** pola. Następnie wybierz **Usuń** z listy rozwijanej.

Importowanie funkcji mapowania typu złożonego

Importy funkcji są oparte na procedury składowane. Aby mapować importowanie funkcji typu złożonego, kolumny

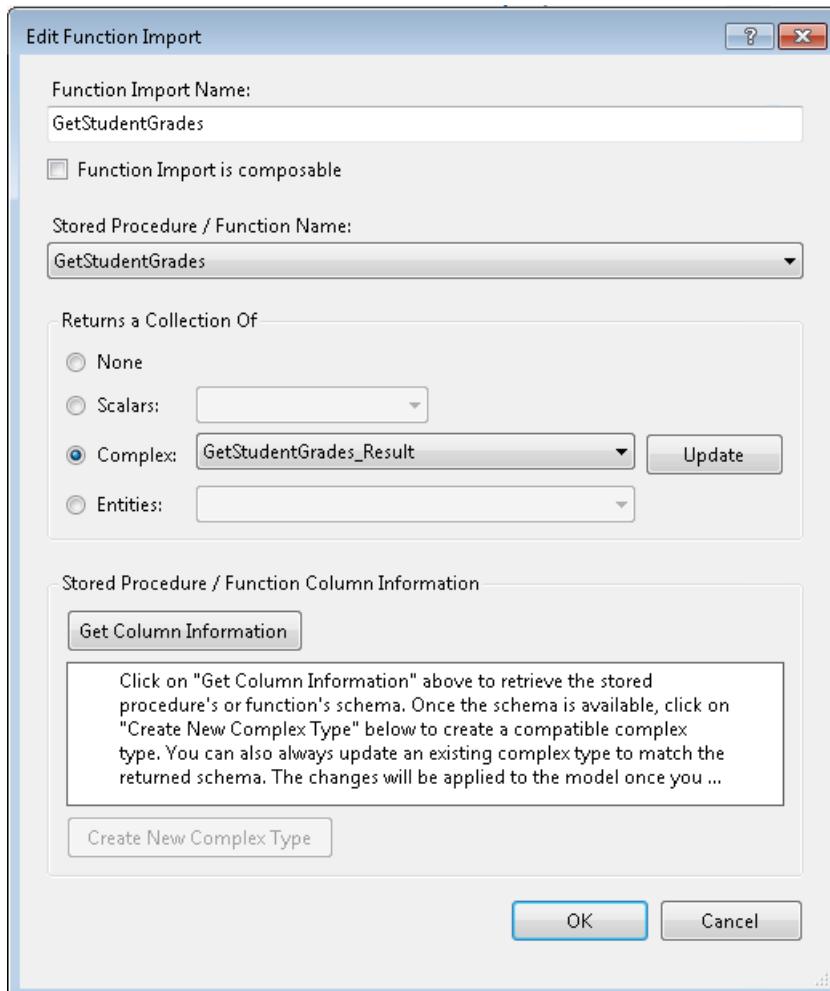
zwracane przez odpowiednie procedury składowanej musi odpowiadać właściwości typu złożonego w liczbie i musi mieć typy magazynu, które są zgodne z typami właściwości.

- Kliknij dwukrotnie importowanych funkcja, która ma być mapowany do typu złożonego.



- Wypełnij ustawienia dla nowego importowanie funkcji w następujący sposób:

- Określ procedury składowanej, dla którego tworzysz importowanie funkcji, w **Nazwa procedury składowanej** pola. To pole jest listy rozwijanej, który wyświetla wszystkie procedury przechowywane w modelu magazynu.
- Określ nazwę funkcji importu w **nazwy importowanie funkcji** pola.
- Wybierz **złożonych** jako zwracany typ, a następnie określ określonych zwracany typ złożony, wybierając odpowiedni typ z listy rozwijanej.



- Kliknij przycisk **OK**. W modelu koncepcyjnym jest tworzony wpis importu funkcji.

Dostosuj kolumny mapowania dla importu — funkcja

- Kliknij prawym przyciskiem myszy importowanie funkcji w przeglądarce modelu, a następnie wybierz pozycję **Importowanie mapowania funkcji. Szczegóły mapowania** oknie wyświetlane wraz z domyślnego mapowania do importu funkcji. Strzałki wskazują mapowania między wartości w kolumnie i wartości właściwości. Domyślnie nazwy kolumn są zakładają się, że taka sama jak nazwy właściwości typu złożonego. Domyślne nazwy kolumn są wyświetlane w kolorze szarym.
- W razie potrzeby zmień nazwy kolumn do dopasowania nazwy kolumn, które są zwracane przez procedurę składowaną, która odnosi się do importu funkcji.

Usuń typ złożony

Jeśli usuniesz to typ złożony, typ zostanie usunięta z modelu koncepcyjnego i mapowania dla wszystkich wystąpień tego typu są usuwane. Odwołania do typu nie są aktualizowane. Na przykład, jeśli określona jednostka ma właściwość typu złożonego typu ComplexType1 i ComplexType1 zostanie usunięty z **przeglądarka modeli**, odpowiednie właściwości jednostki nie jest aktualizowana. Model nie zostanie przeprowadzona Weryfikacja, ponieważ zawiera jednostki, do której odwołuje się do usuniętego typu złożonego. Można zaktualizować lub usunąć odwołania do usuniętych typów złożonych przy użyciu narzędzia Projektant jednostki.

- Kliknij prawym przyciskiem myszy typ złożony w przeglądarce modelu, a następnie wybierz pozycję **Usuń**.
- LUB —
- Wybierz typ złożony w przeglądarce modelu i na klawiaturze naciśnij klawisz Delete.

Zapytań dotyczących jednostek zawierający właściwości typu złożonego

Poniższy kod przedstawia sposób wykonywania zapytania, które zwraca kolekcję jednostek typu obiektów, które zawierają właściwość typu złożonego.

```
using (SchoolEntities context = new SchoolEntities())
{
    var courses =
        from c in context.OnsiteCourses
        order by c.Details.Time
        select c;

    foreach (var c in courses)
    {
        Console.WriteLine("Time: " + c.Details.Time);
        Console.WriteLine("Days: " + c.Details.Days);
        Console.WriteLine("Location: " + c.Details.Location);
    }
}
```

Pomoc techniczna dla wyliczenia — projektancie platformy EF

27.09.2018 • 10 minutes to read • [Edit Online](#)

NOTE

EF5 począwszy tylko — funkcje, interfejsów API itp. z opisem na tej stronie zostały wprowadzone w programie Entity Framework 5. Jeśli używasz starszej wersji, niektóre lub wszystkie informacje, nie ma zastosowania.

W tym przewodniku krok po kroku i wideo pokazuje, jak w typach wyliczeniowych za pomocą programu Entity Framework Designer. Ilustruje też sposób używania typów wyliczeniowych w zapytaniu programu LINQ.

W tym przewodniku używa pierwszego modelu do utworzenia nowej bazy danych, ale może również służyć projektancie platformy EF z [Database First](#) przepływu pracy do mapowania istniejącej bazy danych.

Obsługa typu wyliczeniowego została wprowadzona w Entity Framework 5. Aby korzystać z nowych funkcji, takich jak typy wyliczeniowe, typy danych przestrzennych i funkcji z wartościami przechowywanymi w tabeli, należy wskazać .NET Framework 4.5. Program Visual Studio 2012 jest przeznaczony dla platformy .NET 4.5 domyślnie.

W programie Entity Framework, wyliczenie może mieć następujące typy bazowe: **bajtów**, **Int16**, **Int32**, **Int64**, lub **SByte**.

Obejrzyj wideo

To wideo pokazuje, jak w typach wyliczeniowych za pomocą programu Entity Framework Designer. Ilustruje też sposób używania typów wyliczeniowych w zapytaniu programu LINQ.

Osoba prezentująca: Julia Kornich

Film wideo: [WMV](#) | [MP4](#) | [WMV \(ZIP\)](#)

Wymagania wstępne

Musisz być w wersji programu Visual Studio 2012 Ultimate, Premium, Professional i Web Express zainstalowany w celu przeprowadzenia tego instruktażu.

Konfigurowanie projektu

1. Otwórz program Visual Studio 2012
2. Na **pliku** menu wskaż **New**, a następnie kliknij przycisk **projektu**
3. W okienku po lewej stronie kliknij **Visual C#**, a następnie wybierz pozycję **konsoli** szablonu
4. Wprowadź **EnumEFDesigner** jako nazwę projektu i kliknij przycisk **OK**

Utwórz nowy Model przy użyciu projektanta EF

1. Kliknij prawym przyciskiem myszy nazwę projektu w Eksploratorze rozwiązań, wskaż opcję **Dodaj**, a następnie kliknij przycisk **nowy element**
2. Wybierz **danych** z menu po lewej stronie, a następnie wybierz pozycję **ADO.NET Entity Data Model** w okienku szablonów

3. Wprowadź **EnumTestModel.edmx** nazwę pliku, a następnie kliknij przycisk **Dodaj**
4. Na stronie Kreator modelu Entity Data Model wybierz **pusty Model** w oknie dialogowym Wybierz zawartość modelu
5. Kliknij przycisk **Zakończ**

Zostanie wyświetlona Projektancie jednostki, zapewniającą powierzchnię projektową do edycji modelu.

Kreator wykonuje następujące czynności:

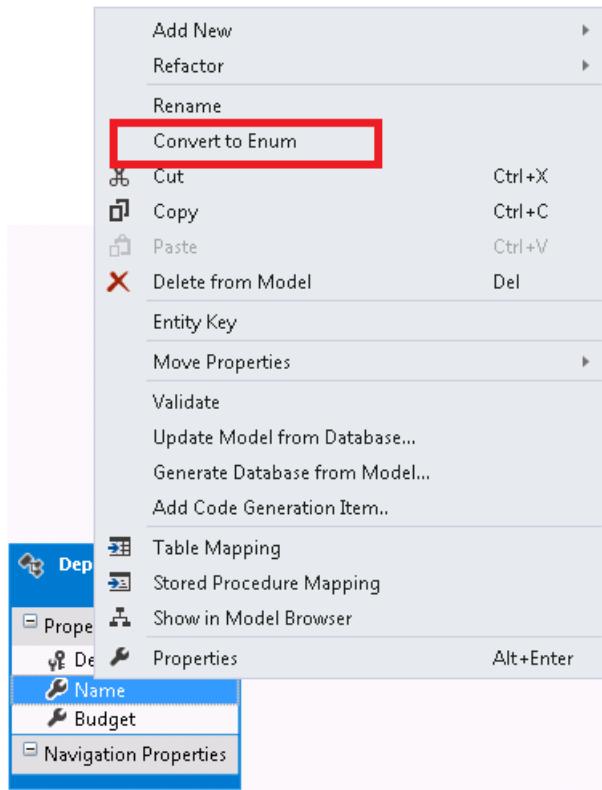
- Generuje plik EnumTestModel.edmx, który definiuje modelu koncepcyjnego, model magazynu i mapowanie między nimi. Ustawia właściwość metadanych artefaktów przetwarzania pliku edmx osadzania w zestawie danych wyjściowych, więc pliki metadanych wygenerowanych Pobierz osadzone w zestawie.
- Dodanie odwołania do następujących zestawów: platformy EntityFramework, System.ComponentModel.DataAnnotations i System.Data.Entity.
- Tworzy pliki EnumTestModel.tt i EnumTestModel.Context.tt i dodaje je w pliku edmx. Te pliki szablonów T4 wygenerować kod, który definiuje typu DbContext pochodnych i POCO typów, które mapują jednostek w modelu edmx.

Dodaj nowy typ jednostki

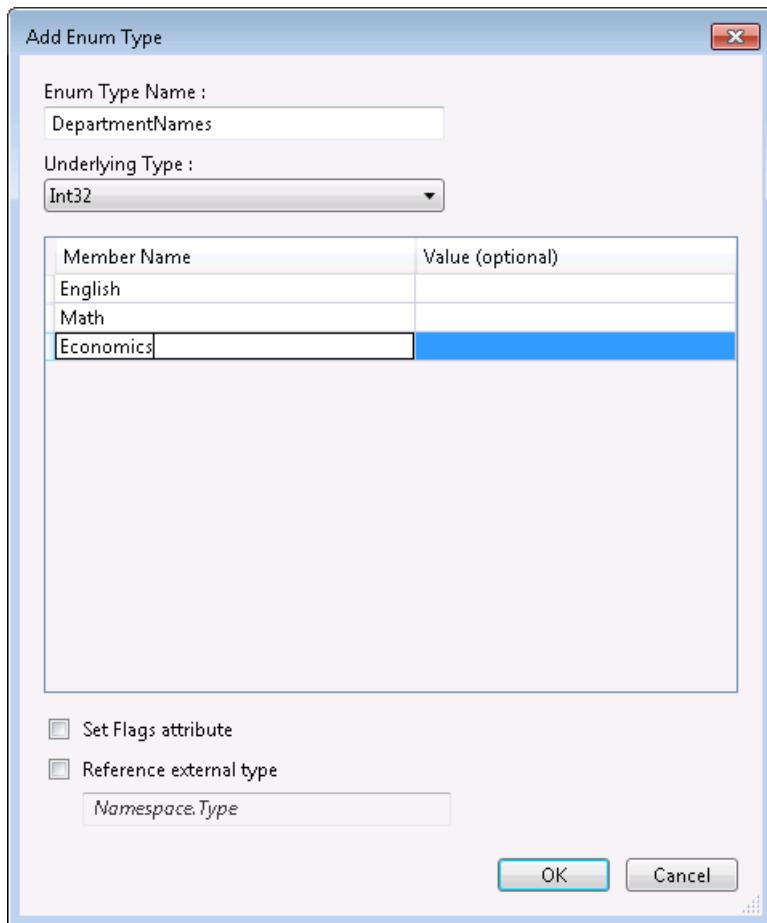
1. Kliknij prawym przyciskiem myszy pusty obszar powierzchni projektu, wybierz opcję **Add -> jednostki**, pojawi się okno dialogowe nowej jednostki
2. Określ **działu** dla typu nazwy i określ **DepartmentID** dla danej nazwy właściwości klucza, pozostaw typ jako **Int32**
3. Kliknij przycisk **OK**
4. Kliknij prawym przyciskiem myszy jednostkę, a następnie wybierz pozycję **Dodaj nowy -> właściwość skalarną**
5. Zmień nazwę nowej właściwości **nazwy**
6. Zmień typ nowej właściwości **Int32** (domyślnie przez nową właściwość jest typu String) Aby zmienić typ, Otwórz okno właściwości, a następnie zmień typ właściwości **Int32**
7. Dodawanie innej skalarne właściwości i zmień jej nazwę na **budżetu**, Zmień typ na **dziesiętna**

Dodaj typ wyliczeniowy

1. W Entity Framework Designer kliknij prawym przyciskiem myszy nazwę właściwości, wybierz **przekonwertować wyliczenia**



2. W **Dodaj wyliczenie** okna dialogowego wpisz **DepartmentNames** dla nazwy typu wyliczenia, Zmień typ podstawowy do **Int32**, a następnie dodaj następujące elementy członkowskie do typu: angielski, Matematyczne i oszczędnościami, jakie daje



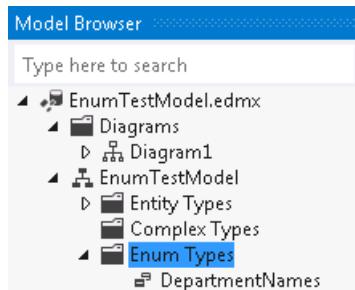
3. Naciśnij klawisz **OK**
4. Zapisz model i skompiluj projekt

NOTE

W przypadku tworzenia, ostrzeżenia dotyczące podmiotów niezmapowanych i skojarzenia może pojawić się na liście błędów. Można zignorować te ostrzeżenia, ponieważ po Wybierzmy wygenerować bazę danych z modelu, błędy znikną.

Jeśli przyjrzymy się w oknie właściwości, zauważysz, że typ właściwości Nazwa została zmieniona na **DepartmentNames** i typ wyliczeniowy nowo dodany została dodana do listy typów.

Po przełączeniu do okna przeglądarki modelu pojawi się, czy typ został również dodany do węzła typach wyliczeniowych.



NOTE

Można również dodać nowe typy wyliczenia z tego okna, klikając prawym przyciskiem myszy i wybierając **Dodaj typ wyliczeniowy**. Po utworzeniu typu pojawi się na liście typów i będzie można skojarzyć z właściwością

Generuj bazę danych z modelu

Teraz możemy wygenerować bazę danych, która jest oparta na modelu.

1. Kliknij prawym przyciskiem myszy pusty obszar w Projektancie jednostki powierzchni i wybierz **Generuj z bazą danych z modelu**
2. Wybierz połączenie danych dialogowym Generuj Kreatora bazy danych jest wyświetlany, kliknij przycisk **nowe połączenie** przycisk Określ **(localdb)\mssqllocaldb** dla nazwy serwera i **EnumTest** bazy danych, a następnie kliknij przycisk **OK**
3. Okno z pytaniem, jeśli chcesz utworzyć nową bazę danych będą się pojawiać, kliknij przycisk **tak**.
4. Kliknij przycisk **dalej** i Kreatora tworzenia bazy danych generuje języka definicji danych (DDL) dla tworzenia bazy danych wygenerowanej języka DDL są wyświetlane w Podsumowanie i okna dialogowego pole Uwaga dotycząca ustawień, który DDL nie zawiera definicji dla tabelę, która mapuje do typu wyliczenia
5. Kliknij przycisk **Zakończ** kliknięcie przycisku Zakończ nie wykonuje skryptu języka DDL.
6. Kreator tworzenia bazy danych wykonuje następujące czynności: Otwiera **EnumTest.edmx.sql** w Edytor T-SQL generuje sekcje schematu i mapowania magazynu EDMX pliku informacji o parametrach połączenia usług AD DS do pliku App.config
7. Kliknij prawym przyciskiem myszy w edytorze języka T-SQL i wybierz pozycję **Execute** nawiązać połączenie z serwerem w oknie dialogowym wyświetlony, podaj informacje o połączeniu z kroku 2, a następnie kliknij polecenie **Connect**
8. Aby wyświetlić wygenerowany schemat, kliknij prawym przyciskiem myszy nazwę bazy danych w Eksploratorze obiektów SQL Server, a następnie wybierz **odświeżania**

Utrwalanie i pobieranie danych

Otwórz plik Program.cs, w którym jest zdefiniowana metoda Main. Dodaj następujący kod do funkcji Main. Ten kod dodaje nowy obiekt działu do kontekstu. Następnie zapisuje dane. Kod wykonuje też zapytanie LINQ, które

zwraca dział, gdzie nazwa jest DepartmentNames.English.

```
using (var context = new EnumTestModelContainer())
{
    context.Departments.Add(new Department{ Name = DepartmentNames.English });

    context.SaveChanges();

    var department = (from d in context.Departments
                      where d.Name == DepartmentNames.English
                      select d).FirstOrDefault();

    Console.WriteLine(
        "DepartmentID: {0} and Name: {1}",
        department.DepartmentID,
        department.Name);
}
```

Skompilować i uruchomić aplikację. Program generuje następujące wyniki:

```
DepartmentID: 1 Name: English
```

Aby wyświetlić dane w bazie danych, kliknij prawym przyciskiem myszy nazwę bazy danych w Eksploratorze obiektów SQL Server, a następnie wybierz **Odśwież**. Kliknięcie prawym przyciskiem myszy tabelę i wybierz pozycję **dane widoku**.

Podsumowanie

W tym przewodniku zobaczyliśmy, jak do mapowania typów wyliczenia przy użyciu programu Entity Framework Designer oraz jak używać wyliczenia w kodzie.

Przestrzenne - projektancie platformy EF

27.09.2018 • 9 minutes to read • [Edit Online](#)

NOTE

EF5 począwszy tylko — funkcje, interfejsów API itp. z opisem na tej stronie zostały wprowadzone w programie Entity Framework 5. Jeśli używasz starszej wersji, niektóre lub wszystkie informacje, nie ma zastosowania.

Przewodnik krok po kroku i wideo pokazuje, jak do mapowania typów przestrzennych z programu Entity Framework Designer. Ilustruje też sposób używania zapytania LINQ można znaleźć odległość między dwiema lokalizacjami.

W tym przewodniku użyje pierwszego modelu do utworzenia nowej bazy danych, ale może również służyć projektancie platformy EF z [Database First](#) przepływu pracy do mapowania istniejącej bazy danych.

Obsługa przestrzenne typu została wprowadzona w programie Entity Framework 5. Należy pamiętać, że aby korzystać z nowych funkcji, takich jak typ przestrzennych, wyliczeń i funkcji z wartościami przechowywanymi w tabeli, należy wskazać .NET Framework 4.5. Program Visual Studio 2012 jest przeznaczony dla platformy .NET 4.5 domyślnie.

Używanie typów danych przestrzennych, należy również użyć dostawcy środowiska Entity Framework, który obsługuje przestrzennych. Zobacz [Obsługa dostawców dla typów przestrzennych](#) Aby uzyskać więcej informacji.

Istnieją dwa typy danych przestrzennych głównego: geometry i położenia geograficznego. Typ danych Geografia przechowuje dane ellipsoidal (na przykład współrzędne geograficzne GPS koordynuje). Typ danych Geometria reprezentuje euklidesowa współrzędnych (płaski).

Obejrzyj wideo

To wideo pokazuje, jak do mapowania typów przestrzennych z programu Entity Framework Designer. Ilustruje też sposób używania zapytania LINQ można znaleźć odległość między dwiema lokalizacjami.

Osoba prezentująca: Julia Kornich

Film wideo: [WMV](#) | [MP4](#) | [WMV \(ZIP\)](#)

Wymagania wstępne

Musisz być w wersji programu Visual Studio 2012 Ultimate, Premium, Professional i Web Express zainstalowany w celu przeprowadzenia tego instruktażu.

Konfigurowanie projektu

1. Otwórz program Visual Studio 2012
2. Na **pliku** menu wskaż **New**, a następnie kliknij przycisk **projektu**
3. W okienku po lewej stronie kliknij **Visual C#**, a następnie wybierz pozycję **konsoli** szablonu
4. Wprowadź **SpatialEFDesigner** jako nazwę projektu i kliknij przycisk **OK**

Utwórz nowy Model przy użyciu projektanta EF

1. Kliknij prawym przyciskiem myszy nazwę projektu w Eksploratorze rozwiązań, wskaż opcję **Dodaj**, a następnie

kliknij przycisk **nowy element**

2. Wybierz **danych** z menu po lewej stronie, a następnie wybierz pozycję **ADO.NET Entity Data Model** w okienku szablonów
3. Wprowadź **UniversityModel.edmx** nazwę pliku, a następnie kliknij przycisk **Dodaj**
4. Na stronie Kreator modelu Entity Data Model wybierz **pusty Model** w oknie dialogowym Wybierz zawartość modelu
5. Kliknij przycisk **Zakończ**

Zostanie wyświetlona Projektancie jednostki, zapewniającą powierzchnię projektową do edycji modelu.

Kreator wykonuje następujące czynności:

- Generuje plik EnumTestModel.edmx, który definiuje modelu koncepcyjnego, model magazynu i mapowanie między nimi. Ustawia właściwość metadanych artefaktów przetwarzania pliku edmx osadzania w zestawie danych wyjściowych, więc pliki metadanych wygenerowanych Pobierz osadzone w zestawie.
- Dodanie odwołania do następujących zestawów: platformy EntityFramework, System.ComponentModel.DataAnnotations i System.Data.Entity.
- Tworzy pliki UniversityModel.tt i UniversityModel.Context.tt i dodaje je w pliku edmx. Te pliki szablonów T4 generowania kodu, który definiuje typu DbContext pochodnych i POCO typów, które mapują jednostek w modelu edmx

Dodaj nowy typ jednostki

1. Kliknij prawym przyciskiem myszy pusty obszar powierzchni projektu, wybierz opcję **Add -> jednostki**, pojawi się okno dialogowe nowej jednostki
2. Określ **University** dla typu nazwy i określ **UniversityID** dla danej nazwy właściwości klucza, pozostaw typ jako **Int32**
3. Kliknij przycisk **OK**
4. Kliknij prawym przyciskiem myszy jednostkę, a następnie wybierz pozycję **Dodaj nowy -> właściwość skalarną**
5. Zmień nazwę nowej właściwości **nazwy**
6. Dodawanie innej skalarne właściwości i zmień jej nazwę na **lokalizacji** Otwórz okno właściwości, a następnie zmień typ nowej właściwości **lokalizacji geograficznej**
7. Zapisz model i skompiluj projekt

NOTE

W przypadku tworzenia, ostrzeżenia dotyczące podmiotów niezmapowanych i skojarzenia może pojawić się na liście błędów. Można zignorować te ostrzeżenia, ponieważ po Wybierzmy wygenerować bazę danych z modelu, błędy znikną.

Generuj bazę danych z modelu

Teraz możemy wygenerować bazę danych, która jest oparta na modelu.

1. Kliknij prawym przyciskiem myszy pusty obszar w Projektancie jednostki powierzchni i wybierz **Generuj z bazy danych z modelu**
2. Wybierz połączenie danych dialogowym Generuj Kreatora bazy danych jest wyświetlany, kliknij przycisk **nowe połączenie** przycisk Określ **(localdb)\mssqllocaldb** dla nazwy serwera i **Uniwersytet** bazy danych, a następnie kliknij przycisk **OK**
3. Okno z pytaniem, jeśli chcesz utworzyć nową bazę danych będą się pojawiać, kliknij przycisk **tak**.
4. Kliknij przycisk **dalej** i Kreatora tworzenia bazy danych generuje języka definicji danych (DDL) dla tworzenia

bazy danych wygenerowanej języka DDL są wyświetlane w Podsumowanie i okna dialogowego pole Uwaga dotycząca ustawień, który DDL nie zawiera definicji dla tabelę, która mapuje do typu wyliczenia

5. Kliknij przycisk **Zakończ** kliknięcie przycisku Zakończ nie wykonuje skryptu języka DDL.
6. Kreator tworzenia bazy danych wykonuje następujące czynności: Otwiera **UniversityModel.edmx.sql** w Edytorze T-SQL generuje sekcje schematu i mapowania magazynu EDMX pliku informacji o parametrach połączenia usług AD DS do pliku App.config
7. Kliknij prawym przyciskiem myszy w edytorze języka T-SQL i wybierz pozycję **Execute** nawiązać połączenie z serwerem w oknie dialogowym wyświetlony, podaj informacje o połączeniu z kroku 2, a następnie kliknij polecenie **Connect**
8. Aby wyświetlić wygenerowany schemat, kliknij prawym przyciskiem myszy nazwę bazy danych w Eksploratorze obiektów SQL Server, a następnie wybierz **odświeżania**

Utrwalanie i pobieranie danych

Otwórz plik Program.cs, w którym jest zdefiniowana metoda Main. Dodaj następujący kod do funkcji Main.

Ten kod dodaje dwa nowe obiekty University do kontekstu. Właściwości przestrzenne są inicjowane za pomocą metody DbGeography.FromText. Punkt lokalizacji geograficznej, reprezentowane jako WellKnownText jest przekazywany do metody. Kod następnie zapisuje dane. Następnie zapytania LINQ, która zwraca obiekt University, gdzie jego lokalizacja jest najbardziej zbliżony do określonej lokalizacji jest tworzony i wykonywane.

```
using (var context = new UniversityModelContainer())
{
    context.Universities.Add(new University()
    {
        Name = "Graphic Design Institute",
        Location = DbGeography.FromText("POINT(-122.336106 47.605049)"),
    });

    context.Universities.Add(new University()
    {
        Name = "School of Fine Art",
        Location = DbGeography.FromText("POINT(-122.335197 47.646711)"),
    });

    context.SaveChanges();

    var myLocation = DbGeography.FromText("POINT(-122.296623 47.640405)");

    var university = (from u in context.Universities
                      orderby u.Location.Distance(myLocation)
                      select u).FirstOrDefault();

    Console.WriteLine(
        "The closest University to you is: {0}.",
        university.Name);
}
```

Skompilować i uruchomić aplikację. Program generuje następujące wyniki:

```
The closest University to you is: School of Fine Art.
```

Aby wyświetlić dane w bazie danych, kliknij prawym przyciskiem myszy nazwę bazy danych w Eksploratorze obiektów SQL Server, a następnie wybierz **Odśwież**. Kliknięcie prawym przyciskiem myszy tabelę i wybierz pozycję **dane widoku**.

Podsumowanie

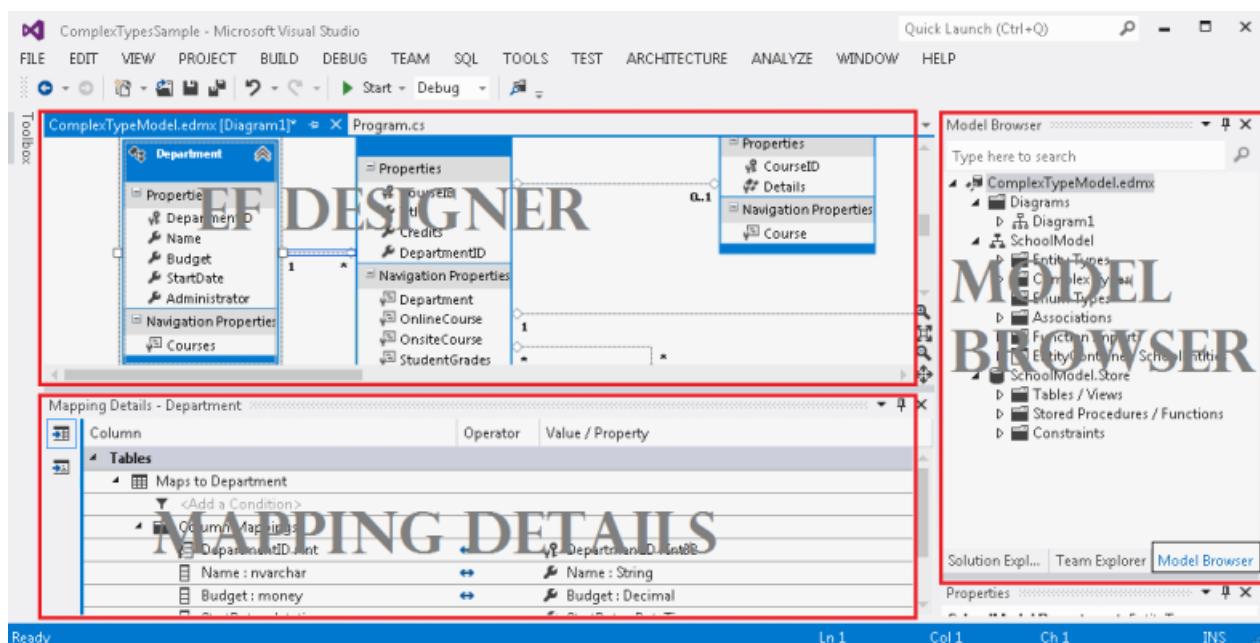
W tym przewodniku zobaczyliśmy, jak do mapowania typów przestrzennych, za pomocą programu Entity Framework Designer oraz jak używać typów przestrzennych w kodzie.

Projektant jednostki dzielenia

13.09.2018 • 7 minutes to read • [Edit Online](#)

Ten poradnik pokazuje jak do mapowania typu encji dwie tabele, modyfikując model przy użyciu narzędzia Entity Framework Designer (Projektant EF). Jednostki można zamapować na wiele tabel, tabele udostępniania wspólny klucz. Pojęcia, które są stosowane do mapowania typu jednostki z dwiema tabelami są łatwo rozszerzyć mapowania typu jednostki w więcej niż dwie tabele.

Na poniżej ilustracji przedstawiono główne systemu windows, które są używane podczas pracy z projektancie platformy EF.



Wymagania wstępne

Program Visual Studio 2012 lub Visual Studio 2010, Ultimate, Premium, Professional lub Web Express edition.

Tworzenie bazy danych

Serwer bazy danych, który został zainstalowany przy użyciu programu Visual Studio różni się zależnie od wersji programu Visual Studio zostały zainstalowane:

- Jeśli używasz programu Visual Studio 2012, a następnie będzie utworzenie bazy danych LocalDB.
- Jeśli używasz programu Visual Studio 2010 zostanie utworzona baza danych programu SQL Express.

Najpierw utworzymy bazę danych z dwoma tabelami, które będziemy połączyć do pojedynczej jednostki.

- Otwórz program Visual Studio
- **Widok —> Eksploratora serwera**
- Kliknij prawym przyciskiem myszy **połączeń danych** -> **Dodaj połączenie...**
- Jeśli nie jest połączona z bazą danych za pomocą Eksploratora serwera, zanim będzie konieczne wybranie **programu Microsoft SQL Server** jako źródło danych
- Łączenie się z LocalDB lub SQL Express, w zależności od tego, który z nich został zainstalowany
- Wprowadź **EntitySplitting** jako nazwa bazy danych
- Wybierz **OK** i uzyskasz, jeśli chcesz utworzyć nową bazę danych, wybierz opcję **tak**

- Nowa baza danych będzie teraz wyświetlany w Eksploratorze serwera
- Jeśli używasz programu Visual Studio 2012
 - Kliknij prawym przyciskiem myszy w bazie danych w Eksploratorze serwera i wybierz **nowe zapytanie**
 - Skopiuj następujące instrukcje SQL do nowego zapytania, a następnie kliknij prawym przyciskiem myszy, zapytania i wybierz pozycję **wykonania**
- Jeśli używasz programu Visual Studio 2010
 - Wybierz **dane —> języka Transact SQL edytor —> nowego połączenia zapytania...**
 - Wprowadź **.\\SQLEXPRESS** jako nazwę serwera i kliknij przycisk **OK**
 - Wybierz **EntitySplitting** bazy danych z listy rozwijanej w górnej części edytora zapytań
 - Skopiuj następujące instrukcje SQL do nowego zapytania, a następnie kliknij prawym przyciskiem myszy, zapytania i wybierz pozycję **wykonaj instrukcję SQL**

```

CREATE TABLE [dbo].[Person] (
[PersonId] INT IDENTITY (1, 1) NOT NULL,
[FirstName] NVARCHAR (200) NULL,
[LastName] NVARCHAR (200) NULL,
CONSTRAINT [PK_Person] PRIMARY KEY CLUSTERED ([PersonId] ASC)
);

CREATE TABLE [dbo].[PersonInfo] (
[PersonId] INT NOT NULL,
[Email] NVARCHAR (200) NULL,
[Phone] NVARCHAR (50) NULL,
CONSTRAINT [PK_PersonInfo] PRIMARY KEY CLUSTERED ([PersonId] ASC),
CONSTRAINT [FK_Person_PersonInfo] FOREIGN KEY ([PersonId]) REFERENCES [dbo].[Person] ([PersonId]) ON DELETE CASCADE
);

```

Tworzenie projektu

- Na **pliku** menu wskaż **New**, a następnie kliknij przycisk **projektu**.
- W okienku po lewej stronie kliknij **Visual C#**, a następnie wybierz pozycję **aplikację Konsolową** szablonu.
- Wprowadź **MapEntityToTablesSample** jako nazwę projektu i kliknij przycisk **OK**.
- Kliknij przycisk **nie** Jeśli zostanie wyświetlony monit, aby zapisać zapytanie SQL utworzonego w pierwszej sekci.

Tworzenie modelu na podstawie bazy danych

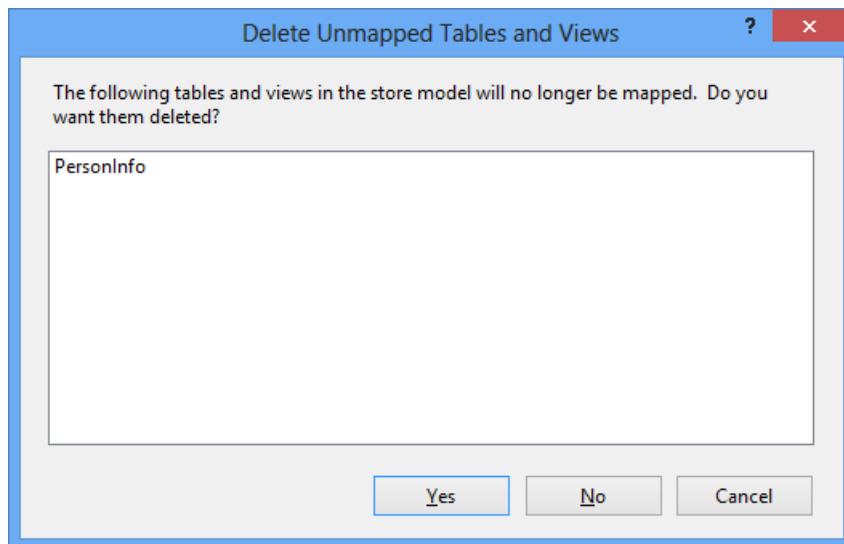
- Kliknij prawym przyciskiem myszy nazwę projektu w Eksploratorze rozwiązań, wskaż opcję **Dodaj**, a następnie kliknij przycisk **nowy element**.
- Wybierz **danych** z menu po lewej stronie, a następnie wybierz pozycję **ADO.NET Entity Data Model** w okienku szablonów.
- Wprowadź **MapEntityToTablesModel.edmx** nazwę pliku, a następnie kliknij przycisk **Dodaj**.
- W oknie dialogowym Wybierz zawartość modelu, wybierz **Generuj z bazy danych**, a następnie kliknij przycisk **dalej**.
- Wybierz **EntitySplitting** połączenia z listy w dół i kliknij przycisk **dalej**.
- W oknie dialogowym Wybierz obiekty bazy danych, zaznacz pole wyboru obok pozycji **tabel** węzła. Spowoduje to dodanie wszystkich tabel z **EntitySplitting** bazy danych do modelu.
- Kliknij przycisk **Zakończ**.

Zostanie wyświetlona Projektancie jednostki, zapewniającą powierzchnię projektową do edycji modelu.

Mapuj jednostki z dwiema tabelami

W tym kroku zaktualizujemy **osoby** typu jednostki, łączyć dane z **osoby** i **PersonInfo** tabel.

- Wybierz **E-mail** i **Phone** właściwości ** PersonInfo ** jednostki, a następnie naciśnij klawisz **Ctrl + X** kluczy.
- Wybierz pozycję ** osoby ** jednostki, a następnie naciśnij klawisz **klawisze Ctrl + V** kluczy.
- Na powierzchni projektowej wybierz **PersonInfo** jednostki, a następnie naciśnij klawisz **Usuń** przycisku na klawiaturze.
- Kliknij przycisk **nie** po wyświetleniu monitu, jeśli chcesz usunąć **PersonInfo** tabelę z modelu, firma Microsoft zamiar zamapować na **osoby** jednostki.



Następne kroki wymagają **szczegóły mapowania** okna. Jeśli nie widzisz tego okna, kliknij prawym przyciskiem myszy projekt powierzchni i wybierz **szczegóły mapowania**.

- Wybierz **osoby** typu jednostki i kliknij przycisk **<Dodaj tabelę lub widok>** w **szczegóły mapowania** okna.
- Wybierz pozycję ** PersonInfo ** z listy rozwijanej. **Szczegóły mapowania** okno zostanie zaktualizowana przy użyciu domyślnego mapowania kolumn, są w dobrym stanie w naszym scenariuszu.

Osoby typ jednostki jest obecnie zmapowane do **osoby** i **PersonInfo** tabel.

A screenshot of the "Mapping Details - Person" dialog box. It has a tree view on the left and a table view on the right. The tree view shows "Tables" expanded, with "Maps to Person" and "Maps to PersonInfo" selected. The table view shows column mappings between "Person" and "PersonInfo" tables. For "Maps to Person", columns PersonId, FirstName, and LastName are mapped to PersonId, FirstName, and LastName respectively. For "Maps to PersonInfo", columns PersonId, Email, and Phone are mapped to PersonId, Email, and Phone respectively. The "Maps to PersonInfo" section is highlighted with a blue selection bar.

Użyj modelu

- Wklej następujący kod dla metody Main.

```

using (var context = new EntitySplittingEntities())
{
    var person = new Person
    {
        FirstName = "John",
        LastName = "Doe",
        Email = "john@example.com",
        Phone = "555-555-5555"
    };

    context.People.Add(person);
    context.SaveChanges();

    foreach (var item in context.People)
    {
        Console.WriteLine(item.FirstName);
    }
}

```

- Skompilować i uruchomić aplikację.

Poniższe instrukcje języka T-SQL były wykonywane w bazie danych, w wyniku działania tej aplikacji.

- Następujące dwa **Wstaw** instrukcje zostały wykonane w wyniku wykonania z kontekstu. SaveChanges(). Przyjmują, dane z **osoby** jednostki i podziel go między **osoby** i **PersonInfo** tabeli.

⚡ ADO.NET: Execute Reader "insert [dbo].[Person]([FirstName], [L
The command text "insert [dbo].[Person]([FirstName],
[LastName])
values (@0, @1)
select [PersonId]
from [dbo].[Person]
where @@ROWCOUNT > 0 and [PersonId] = scope_identity()
was executed on connection "data source=(localdb)
\v11.0;initial catalog=EntitySplitting;integrated
security=True;MultipleActiveResultSets=True;App=EntityFram
ework", building a SqlDataReader.
Thread: Main Thread [2052]
Related views: [Locals](#) [Call Stack](#)

⚡ ADO.NET: Execute NonQuery "insert [dbo].[PersonInfo]([Person
The command text "insert [dbo].[PersonInfo]([PersonId],
[Email], [Phone])
values (@0, @1, @2)
" was executed on connection "data source=(localdb)
\v11.0;initial catalog=EntitySplitting;integrated
security=True;MultipleActiveResultSets=True;App=EntityFram
ework", returning the number of rows affected.
Thread: Main Thread [2052]
Related views: [Locals](#) [Call Stack](#)

- Następujące **wybierz** zostało wykonane w wyniku wyliczanie osób w bazie danych. Łączy ona dane z **osoby** i **PersonInfo** tabeli.

⚡ ADO.NET: Execute Reader "SELECT [Extent1].[PersonId] AS [Per
The command text "SELECT
[Extent1].[PersonId] AS [PersonId],
[Extent2].[FirstName] AS [FirstName],
[Extent2].[LastName] AS [LastName],
[Extent1].[Email] AS [Email],
[Extent1].[Phone] AS [Phone]
FROM [dbo].[PersonInfo] AS [Extent1]
INNER JOIN [dbo].[Person] AS [Extent2] ON [Extent1].[PersonId] = [Extent2].[PersonId]" was executed on connection
"data source=(localdb)\v11.0;initial
catalog=EntitySplitting;integrated
security=True;MultipleActiveResultSets=True;App=EntityFram
ework", building a SqlDataReader.
Thread: Main Thread [2052]
Related views: [Locals](#) [Call Stack](#)

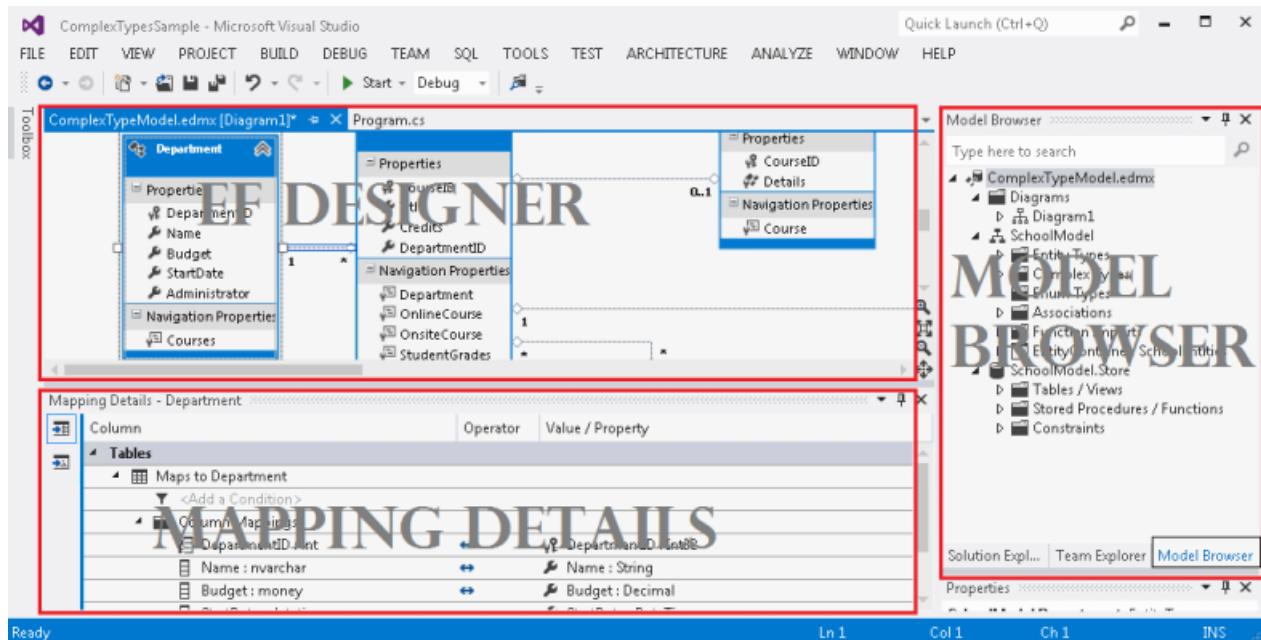
Tabela projektanta dzielenia

13.09.2018 • 7 minutes to read • [Edit Online](#)

Ten poradnik pokazuje jak mapować wielu typów jednostek do pojedynczej tabeli, modyfikując model przy użyciu narzędzia Entity Framework Designer (Projektant EF).

Jeden powód, dla którego chcesz użyć tabeli podział opóźnia ładowanie niektórych właściwości podczas korzystania z opóźnieniem, ładowania do ładowania obiektów. Można oddzielić właściwości, które mogą zawierać bardzo duże ilości danych na osobne jednostki i tylko załadować je, gdy jest to wymagane.

Na poniżej ilustracji przedstawiono główne systemu windows, które są używane podczas pracy z projektancie platformy EF.



Wymagania wstępne

W celu wykonania instrukcji w tym przewodniku potrzebne są następujące elementy:

- Najnowszą wersję programu Visual Studio.
- [Przykładowej bazy danych School](#).

Konfigurowanie projektu

Ten przewodnik korzysta z programu Visual Studio 2012.

- Otwórz program Visual Studio 2012.
- Na **pliku** menu wskaż **New**, a następnie kliknij przycisk **projektu**.
- W okienku po lewej stronie kliknij Visual C#, a następnie wybierz szablon Aplikacja konsoli.
- Wprowadź **TableSplittingSample** jako nazwę projektu i kliknij przycisk **OK**.

Tworzenie modelu, w oparciu o bazę danych School

- Kliknij prawym przyciskiem myszy nazwę projektu w Eksploratorze rozwiązań, wskaż opcję **Dodaj**, a następnie kliknij przycisk **nowy element**.

- Wybierz **danych** z menu po lewej stronie, a następnie wybierz pozycję **ADO.NET Entity Data Model** w okienku szablonów.
- Wprowadź **TableSplittingModel.edmx** nazwę pliku, a następnie kliknij przycisk **Dodaj**.
- W oknie dialogowym Wybierz zawartość modelu, wybierz **Generuj z bazy danych**, a następnie kliknij przycisk **dalej**.
- Kliknij przycisk nowe połączenie. W oknie dialogowym właściwości połączenia, wprowadź nazwę serwera (na przykład **(localdb)\mssqllocaldb**), wybierz metodę uwierzytelniania, wpisz **School** nazwy bazy danych, a następnie Kliknij przycisk **OK**. Okno dialogowe Wybierz połączenie danych jest aktualizowana ustawieniem połączenia bazy danych.
- W oknie dialogowym Wybierz obiekty bazy danych należy ujawniać **tabel** węzła i wyboru **osoby** tabeli. Spowoduje to dodanie określonej tabeli, aby **School** modelu.
- Kliknij przycisk **Zakończ**.

Zostanie wyświetlona Projektancie jednostki, zapewniającą powierzchnię projektową do edycji modelu. Wszystkie obiekty, które wybrano w **wybierz obiekty bazy danych** okno dialogowe, są dodawane do modelu.

Dwie jednostki mapy do pojedynczej tabeli

W tej sekcji zostanie podzielona **osoby** jednostki do dwóch jednostek i zamapować je do jednej tabeli.

NOTE

Osoby jednostka nie zawiera żadnych właściwości, które mogą zawierać dużą ilość danych; jest używany tylko jako przykład.

- Kliknij prawym przyciskiem myszy pusty obszar powierzchni projektu, wskaż opcję **Dodaj nowej** kliknij przycisk **jednostki. Nowa jednostka** pojawi się okno dialogowe.
- Typ **HireInfo** dla **nazwa jednostki** i **PersonID** dla **właściwości klucza** nazwy.
- Kliknij przycisk **OK**.
- Nowy typ jednostki jest tworzone i wyświetlane na powierzchni projektowej.
- Wybierz **DataZatrudnienia** właściwość **osoby** typu jednostki, a następnie naciśnij klawisz **Ctrl + X** kluczy.
- Wybierz **HireInfo** jednostki, a następnie naciśnij klawisz **klawisze Ctrl + V** kluczy.
- Tworzenie skojarzenia między **osoby** i **HireInfo**. Aby to zrobić, kliknij prawym przyciskiem myszy pusty obszar powierzchni projektu, wskaż opcję **Dodaj nowej** kliknij przycisk **skojarzenie**.
- **Dodawanie skojarzenia** pojawi się okno dialogowe. **PersonHireInfo** domyślnie zostanie podana nazwa.
- Określać liczebność **1(One)** po obu stronach relacji.
- Naciśnij klawisz **OK**.

Następny krok wymaga **szczegóły mapowania** okna. Jeśli nie widzisz tego okna, kliknij prawym przyciskiem myszy projekt powierzchni i wybierz **szczegóły mapowania**.

- Wybierz **HireInfo** typu jednostki i kliknij przycisk **<Dodaj tabelę lub widok>** w **szczegóły mapowania** okna.
- Wybierz **osoby** z **<Dodaj tabelę lub widok>** pole listy rozwijanej. Lista zawiera tabele lub widoki, które można mapować wybranej jednostki. Odpowiednie właściwości powinny być mapowane domyślnie.

Column	Ope...	Value / Property
PersonID : int	<input type="button" value="↔"/>	PersonID : Int32
LastName : nvarchar	<input type="button" value="↔"/>	
FirstName : nvarchar	<input type="button" value="↔"/>	
HireDate : datetime	<input type="button" value="↔"/>	HireDate : DateTime
EnrollmentDate : datetime	<input type="button" value="↔"/>	
Discriminator : nvarchar	<input type="button" value="↔"/>	

- Wybierz **PersonHireInfo** skojarzenia na powierzchni projektowej.
- Kliknij prawym przyciskiem myszy asocjacji na projekt powierzchni i wybierz **właściwości**.
- W **właściwości** wybierz **ograniczenia referencyjne** właściwości i kliknij przycisk z wielokropkiem.
- Wybierz **osoby z jednostki** listy rozwijanej.
- Naciśnij klawisz **OK**.

Użyj modelu

- Wklej następujący kod dla metody Main.

```

using (var context = new SchoolEntities())
{
    Person person = new Person()
    {
        FirstName = "Kimberly",
        LastName = "Morgan",
        Discriminator = "Instructor",
    };

    person.HireInfo = new HireInfo()
    {
        HireDate = DateTime.Now
    };

    // Add the new person to the context.
    context.People.Add(person);

    // Insert a row into the Person table.
    context.SaveChanges();

    // Execute a query against the Person table.
    // The query returns columns that map to the Person entity.
    var existingPerson = context.People.FirstOrDefault();

    // Execute a query against the Person table.
    // The query returns columns that map to the Instructor entity.
    var hireInfo = existingPerson.HireInfo;

    Console.WriteLine("{0} was hired on {1}",
        existingPerson.LastName, hireInfo.HireDate);
}

```

- Skompilować i uruchomić aplikację.

Poniższe instrukcje języka T-SQL zostały wykonane na **School** bazy danych w wyniku działania tej aplikacji.

- Następujące **Wstaw** zostało wykonane w wyniku wykonania z kontekstu. SaveChanges() i łączy dane z

osoby i HireInfo jednostek

⚡ **ADO.NET:** Execute Reader "insert [dbo].[Person]([LastName], [FirstName], [HireDate], [EnrollmentDate], [Discriminator]) values (@0, @1, @2, null, @3) select [PersonID] from [dbo].[Person] where @@ROWCOUNT > 0 and [PersonID] = scope_identity()" was executed on connection "data source=(localdb)\v11.0;initial catalog=School;integrated security=True;MultipleActiveResultSets=True;App=EntityFramework", building a SqlDataReader.

- Następujące **wybierz** zostało wykonane w wyniku wykonania z kontekstu `People.FirstOrDefault()` i wybiera tylko potrzebne kolumny mapowane na **osoby**

⚡ **ADO.NET:** Execute Reader "SELECT TOP (1) [c].[PersonID], [c].[LastName] AS [LastName], [c].[FirstName] AS [FirstName], [c].[EnrollmentDate] AS [EnrollmentDate], [c].[Discriminator] AS [Discriminator] FROM [dbo].[Person] AS [c]" was executed on connection "data source=(localdb)\v11.0;initial catalog=School;integrated security=True;MultipleActiveResultSets=True;App=EntityFramework", building a SqlDataReader.

- Następujące **wybierz** zostało wykonane w wyniku uzyskiwania dostępu do `existingPerson.Instructor` właściwość nawigacji z wybiera tylko potrzebne kolumny, które są mapowane na **HireInfo**

⚡ **ADO.NET:** Execute Reader "SELECT [Extent1].[PersonID], [Extent1].[HireDate] FROM [dbo].[Person] AS [Extent1] WHERE [Extent1].[PersonID] = @EntityKeyValue1" was executed on connection "data source=(localdb)\v11.0;initial catalog=School;integrated security=True;MultipleActiveResultSets=True;App=EntityFramework", building a SqlDataReader.

Dziedziczenie TPH projektanta

13.09.2018 • 10 minutes to read • [Edit Online](#)

Ten przewodnik krok po kroku pokazano, jak zaimplementować Tabela wg hierarchii (TPH) dziedziczenia w model konceptyjny za pomocą funkcji Entity Framework Designer (Projektant EF). Dziedziczenie TPH używa jednej tabeli bazy danych do przechowywania danych dla wszystkich typów jednostek w hierarchii dziedziczenia.

W ramach tego przewodnika firma Microsoft będzie zmapowana tabeli osób do trzech typów jednostek: osoba (typ podstawowy), uczniów (pochodzi od osoby) i przez instruktorów (pochodzi od osoby). Utworzymy model konceptyjny z bazy danych (Database First) i następnie zmienić model, który ma implementują dziedziczenie TPH przy użyciu projektanta EF.

Można zamapować na dziedziczenie TPH, za pomocą funkcji First modelu, ale trzeba napisać własne przepływu pracy generowanie bazy danych, którą jest złożony. Następnie można przypisać ten przepływ pracy ma **generowanie bazy danych w przepływie pracy** właściwości w Projektancie platformy EF. Alternatywą łatwiej jest używać Code First.

Inne opcje dziedziczenia

Tabela wg typu (TPT) jest inny typ dziedziczenia, w którym oddzielnego tabel w bazie danych są mapowane do jednostek, które uczestniczą w dziedziczeniu. Aby uzyskać informacje o mapowaniu tabeli na typ dziedziczenia z projektancie platformy EF, zobacz [dziedziczenia TPT projektancie platformy EF](#).

Dziedziczenie typu tabeli na konkretny (TPC) i wzory mieszane dziedziczenia są obsługiwane przez środowisko uruchomieniowe programu Entity Framework, ale nie są obsługiwane w Projektancie platformy EF. Jeśli chcesz użyć TPC lub mieszane dziedziczenia, masz dwie opcje: Użyj Code First lub ręczna Edycja pliku EDMX. Jeśli użytkownik chce pracować z pliku EDMX, okno Szczegóły mapowania, które zostaną wprowadzone w "trybie awaryjnym" i nie można zmienić mapowania za pomocą projektanta.

Wymagania wstępne

W celu wykonania instrukcji w tym przewodniku potrzebne są następujące elementy:

- Najnowszą wersję programu Visual Studio.
- [Przykładowej bazy danych School](#).

Konfigurowanie projektu

- Otwórz program Visual Studio 2012.
- Wybierz **plikach > New -> projektu**
- W okienku po lewej stronie kliknij **Visual C#**, a następnie wybierz pozycję **konsoli** szablonu.
- Wprowadź **TPHDBFirstSample** jako nazwę.
- Wybierz **OK**.

Tworzenie modelu

- Kliknij prawym przyciskiem myszy nazwę projektu w Eksploratorze rozwiązań, a następnie wybierz pozycję **Add -> nowy element**.
- Wybierz **danych** z menu po lewej stronie, a następnie wybierz pozycję **ADO.NET Entity Data Model** w okienku szablonów.

- Wprowadź **TPHModel.edmx** nazwę pliku, a następnie kliknij przycisk **Dodaj**.
- W oknie dialogowym Wybierz zawartość modelu, wybierz **Generuj z bazy danych**, a następnie kliknij przycisk **dalej**.
- Kliknij przycisk **nowe połączenie**. W oknie dialogowym właściwości połączenia, wprowadź nazwę serwera (na przykład **(localdb)\mssqllocaldb**), wybierz metodę uwierzytelniania, wpisz **School** nazwy bazy danych, a następnie Kliknij przycisk **OK**. Okno dialogowe Wybierz połączenie danych jest aktualizowana ustawienie połączenia bazy danych.
- W oknie dialogowym Wybierz obiekty bazy danych w węźle tabel, wybierz **osoby** tabeli.
- Kliknij przycisk **Zakończ**.

Zostanie wyświetlona Projektancie jednostki, zapewniającą powierzchnię projektową do edycji modelu. Wszystkie obiekty, które zostały wybrane w oknie dialogowym Wybierz obiekty bazy danych są dodawane do modelu.

Oznacza to sposób, w jaki **osoby** tabela wygląda w bazie danych.

	Name	Data Type	Allow Nulls
#o	PersonID	int	<input type="checkbox"/>
	LastName	nvarchar(50)	<input type="checkbox"/>
	FirstName	nvarchar(50)	<input type="checkbox"/>
	HireDate	datetime	<input checked="" type="checkbox"/>
	EnrollmentDate	datetime	<input checked="" type="checkbox"/>
	Discriminator	nvarchar(50)	<input type="checkbox"/>
			<input type="checkbox"/>

Implementowanie Tabela wg hierarchii dziedziczenia

Osoby tabela ma **dyskryminatora** kolumny, która może mieć jedną z dwóch wartości: "Student" i "Instruktora". W zależności od wartości **osoby** tabeli zostaną zmapowane do **ucznów** jednostki lub **przez instruktorów** jednostki. **Osoby** tabela ma także dwie kolumny **DataZatrudnienia** i **EnrollmentDate**, które muszą być **dopuszczającego wartość null** ponieważ osoba nie może być dla uczniów i pod kierunkiem instruktora w tym samym czasie, (a przynajmniej nie w tym przewodniku).

Dodaj nowe jednostki

- Dodaj nową jednostkę. Aby to zrobić, kliknij prawym przyciskiem myszy pusty obszar powierzchni projektu programu Entity Framework Designer, a następnie wybierz **Add ->jednostki**.
- Typ **przez instruktorów** dla **nazwa jednostki** i wybierz **osoby** z listy rozwijanej dla **typ podstawowy**.
- Kliknij przycisk **OK**.
- Dodaj inną nową jednostkę. Typ **ucznów** dla **nazwa jednostki** i wybierz **osoby** z listy rozwijanej dla **typ podstawowy**.

Dwóch nowych typów jednostki zostały dodane do powierzchni projektowej. Strzałka wskazuje z nowych typów jednostek do **osoby** typu jednostki; oznacza to, że **osoby** jest typem podstawowym dla nowych typów jednostek.

- Kliknij prawym przyciskiem myszy **DataZatrudnienia** właściwość **osoby** jednostki. Wybierz **Wytnij** (lub użyj klawisza Ctrl-X).
- Kliknij prawym przyciskiem myszy **przez instruktorów** jednostki, a następnie wybierz pozycję **Wklej** (lub użyj klawisza Ctrl-V).
- Kliknij prawym przyciskiem myszy **DataZatrudnienia** właściwości i wybierz pozycję **właściwości**.
- W **właściwości** oknie **Nullable** właściwości **false**.
- Kliknij prawym przyciskiem myszy **EnrollmentDate** właściwość **osoby** jednostki. Wybierz **Wytnij** (lub użyj klawisza Ctrl-X).

- Kliknij prawym przyciskiem myszy **uczniów** jednostki, a następnie wybierz pozycję **Wklej (lub użyj klawiszy Ctrl-V klucza)**.
- Wybierz **EnrollmentDate** właściwości i ustaw **Nullable** właściwości **false**.
- Wybierz **osoby** typu jednostki. W **właściwości** oknie jego **abstrakcyjne** właściwości **true**.
- Usuń **dyskryminatora** właściwość **osoby**. Powodów, dla którego należy usunąć jest szczegółowo opisane w poniższej sekcji.

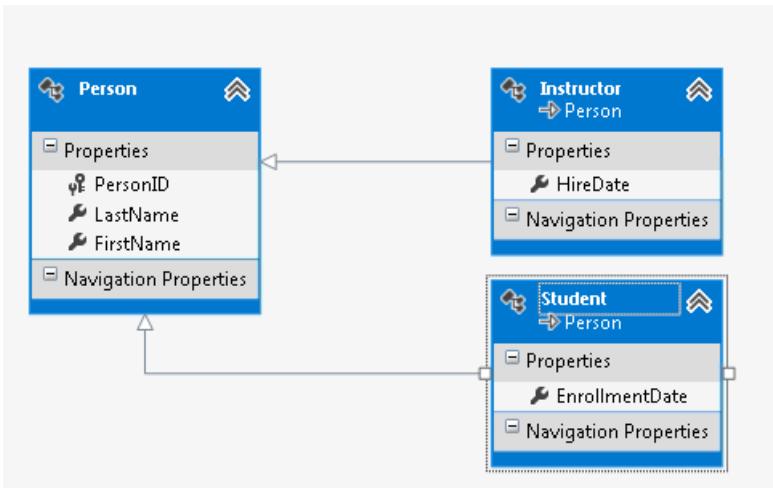
Mapuj jednostki

- Kliknij prawym przyciskiem myszy **przez instruktorów** i wybierz **mapowania tabeli**. W oknie Szczegóły mapowania wybrano jednostki przez instruktorów.
- Kliknij przycisk **<Dodaj tabelę lub widok>** w **szczegóły mapowania** okna. **<Dodaj tabelę lub widok>** pole staje się listy rozwijanej tabel lub widoków, które można mapować wybranej jednostki.
- Wybierz **osoby** z listy rozwijanej.
- **Szczegóły mapowania** okno jest aktualizowane przy użyciu domyślnego mapowania kolumn oraz opcję dodawania warunku.
- Kliknij pozycję **<Dodaj warunek>**. **<Dodaj warunek>** pole staje się listy rozwijanej, kolumn, dla których można ustawić warunki.
- Wybierz **dyskryminatora** z listy rozwijanej.
- W **Operator** kolumny **szczegóły mapowania** wybierz = z listy rozwijanej.
- W **wartość/właściwości** wpisz **przez instruktorów**. Wynik końcowy powinien wyglądać następująco:

Column	Operator	Value / Property
Tables		
Maps to Person	=	Instructor
When Discriminator		
<Add a Condition>		
Column Mappings		
HireDate : datetime	↔	HireDate : DateTime
EnrollmentDate : datetime	↔	
Discriminator : nvarchar	↔	

- Powtórz te kroki dla **uczniów** typu jednostki, ale Utwórz warunek równa **uczniów** wartość. *Dlatego Chcieliśmy, aby usunąć dyskryminatora właściwość, jest, ponieważ nie można zamapować kolumny tabeli więcej niż jeden raz. Ta kolumna będzie służyć warunkowego mapowania, więc nie można używać właściwości mapowania także. Jedynym sposobem może służyć w obu przypadkach, gdy warunek będzie używać Is Null lub Is Not Null porównania.*

Tabela wg hierarchii dziedziczenia jest teraz implementowane.



Użyj modelu

Otwórz **Program.cs** pliku gdzie **Main** metoda jest zdefiniowana. Wklej następujący kod do **Main** funkcji. Kod wykonuje trzy zapytania. Pierwsze zapytanie powoduje zwrócenie wszystkich **osoby** obiektów. Drugie zapytanie używa **OfType** metodę, aby zwrócić **przez instruktorów** obiektów. Trzecie zapytanie używa **OfType** metodę, aby zwrócić **uczniów** obiektów.

```

using (var context = new SchoolEntities())
{
    Console.WriteLine("All people:");
    foreach (var person in context.People)
    {
        Console.WriteLine("    {0} {1}", person.FirstName, person.LastName);
    }

    Console.WriteLine("Instructors only: ");
    foreach (var person in context.People.OfType<Instructor>())
    {
        Console.WriteLine("    {0} {1}", person.FirstName, person.LastName);
    }

    Console.WriteLine("Students only: ");
    foreach (var person in context.People.OfType<Student>())
    {
        Console.WriteLine("    {0} {1}", person.FirstName, person.LastName);
    }
}

```

Dziedziczenie TPT projektanta

13.09.2018 • 6 minutes to read • [Edit Online](#)

Ten przewodnik krok po kroku pokazano, jak zaimplementować Tabela wg typu (TPT) dziedziczenia w modelu przy użyciu narzędzia Entity Framework Designer (Projektant EF). Tabela wg typu dziedziczenia używa osobnej tabeli w bazie danych do przechowywania danych dla właściwości dziedziczonych i właściwości klucza dla każdego typu w hierarchii dziedziczenia.

W tym przewodniku, firma Microsoft będzie zmapowana **kurs** (typ podstawowy) **OnlineCourse** (pochodzi z kursu) i **OnsiteCourse** (pochodzi od klasy **kurs**) jednostki do tabel z takich samych nazwach. Utworzymy model z bazy danych i następnie zmienić model, który ma implementują dziedziczenie TPT.

Można również uruchomić z pierwszego modelu i następnie wygenerować bazę danych z modelu. Projektancie platformy EF używa strategii TPT domyślnie, a więc wszystkie dziedziczenia w modelu będą mapowane do oddzielnych tabel.

Inne opcje dziedziczenia

Tabela wg hierarchii (TPH) jest inny typ dziedziczenia, w której jedna baza danych tabela jest używana do przechowywania danych dla wszystkich typów jednostek w hierarchii dziedziczenia. Aby uzyskać informacje o mapowaniu Tabela wg hierarchii dziedziczenia z Projektanta obiektów, zobacz [dziedziczenia TPH projektancie platformy EF](#).

Należy pamiętać, że tabeli na konkretny typ dziedziczenia (TPC) i dziedziczenie mieszane modeli są obsługiwane przez środowisko uruchomieniowe programu Entity Framework, ale nie są obsługiwane w Projektancie platformy EF. Jeśli chcesz użyć TPC lub mieszane dziedziczenia, masz dwie opcje: Użyj Code First lub ręczna Edycja pliku EDMX. Jeśli użytkownik chce pracować z pliku EDMX, okno Szczegóły mapowania, które zostaną wprowadzone w "trybie awaryjnym" i nie można zmienić mapowania za pomocą projektanta.

Wymagania wstępne

W celu wykonania instrukcji w tym przewodniku potrzebne są następujące elementy:

- Najnowszą wersję programu Visual Studio.
- [Przykładowej bazy danych School](#).

Konfigurowanie projektu

- Otwórz program Visual Studio 2012.
- Wybierz **plikach > New -> projektu**
- W okienku po lewej stronie kliknij **Visual C#**, a następnie wybierz pozycję **konsoli** szablonu.
- Wprowadź **TPTDBFirstSample** jako nazwę.
- Wybierz **OK**.

Tworzenie modelu

- Kliknij prawym przyciskiem myszy projekt w Eksploratorze rozwiązań, a następnie wybierz pozycję **Add -> nowy element**.
- Wybierz **danych** z menu po lewej stronie, a następnie wybierz pozycję **ADO.NET Entity Data Model** w okienku szablonów.

- Wprowadź **TPTModel.edmx** nazwę pliku, a następnie kliknij przycisk **Dodaj**.
- W oknie dialogowym Wybierz zawartość modelu, wybierz pozycję ** Generuj z bazy danych, a następnie kliknij przycisk **dalej**.
- Kliknij przycisk **nowe połączenie**. W oknie dialogowym właściwości połączenia, wprowadź nazwę serwera (na przykład **(localdb)\mssqllocaldb**), wybierz metodę uwierzytelniania, wpisz **School** nazwy bazy danych, a następnie Kliknij przycisk **OK**. Okno dialogowe Wybierz połączenie danych jest aktualizowana ustawieniem połączenia bazy danych.
- W oknie dialogowym Wybierz obiekty bazy danych w węźle tabel, wybierz **działu, kursu, OnlineCourse i OnsiteCourse** tabel.
- Kliknij przycisk **Zakończ**.

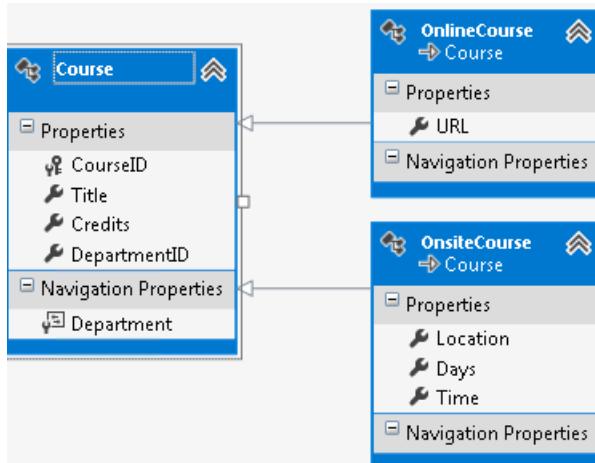
Zostanie wyświetlona Projektancie jednostki, zapewniającą powierzchnię projektową do edycji modelu. Wszystkie obiekty, które zostały wybrane w oknie dialogowym Wybierz obiekty bazy danych są dodawane do modelu.

Implementowanie dziedziczenia tabelę według typu

- Na powierzchni projektowej kliknij prawym przyciskiem myszy **OnlineCourse** typu jednostki, a następnie wybierz **właściwości**.
- W **właściwości** okna, ustaw właściwość typu **Base kurs**.
- Kliknij prawym przyciskiem myszy **OnsiteCourse** typu jednostki, a następnie wybierz **właściwości**.
- W **właściwości** okna, ustaw właściwość typu **Base kurs**.
- Kliknij prawym przyciskiem myszy skojarzenia (wiersz) między **OnlineCourse i kurs** typów jednostek. Wybierz **usunięte z modelu**.
- Kliknij prawym przyciskiem myszy skojarzenie między **OnsiteCourse i kurs** typów jednostek. Wybierz **usunięte z modelu**.

Firma Microsoft usunie teraz **CourseID** właściwość **OnlineCourse i OnsiteCourse** ponieważ te klasy dziedziczą **CourseID z kurs** typ podstawowy.

- Kliknij prawym przyciskiem myszy **CourseID** właściwość **OnlineCourse** typu jednostki, a następnie wybierz **usunięte z modelu**.
- Kliknij prawym przyciskiem myszy **CourseID** właściwość **OnsiteCourse** typu jednostki, a następnie wybierz **usunięte z modelu**
- Tabela wg typu dziedziczenia jest teraz implementowane.



Użyj modelu

Otwórz **Program.cs** pliku gdzie **Main** metoda jest zdefiniowana. Wklej następujący kod do **Main** funkcji. Kod wykonuje trzy zapytania. Pierwsze zapytanie powoduje zwrócenie wszystkich **kursów** związane z określonej dział. Drugie zapytanie używa **OfType** metodę, aby zwrócić **OnlineCourses** związane z określonej dział. Zwraca trzecie

zapytanie **OnsiteCourses**.

```
using (var context = new SchoolEntities())
{
    foreach (var department in context.Departments)
    {
        Console.WriteLine("The {0} department has the following courses:",
                          department.Name);

        Console.WriteLine("    All courses");
        foreach (var course in department.Courses )
        {
            Console.WriteLine("        {0}", course.Title);
        }

        foreach (var course in department.Courses.
                  OfType<OnlineCourse>())
        {
            Console.WriteLine("        Online - {0}", course.Title);
        }

        foreach (var course in department.Courses.
                  OfType<OnsiteCourse>())
        {
            Console.WriteLine("        Onsite - {0}", course.Title);
        }
    }
}
```

Procedury składowane projektanta zapytań

27.09.2018 • 5 minutes to read • [Edit Online](#)

Ten przewodnik krok po kroku pokazano, jak używać Entity Framework Designer (Projektant EF) do zimportowania procedur składowanych do modelu, a następnie wywołać importowanych procedur składowanych do pobierania wyników.

Należy pamiętać, że Code First platformy nie obsługuje mapowania do procedury składowanej lub funkcji. Jednak można wywoływać procedury składowanej lub funkcji za pomocą metody System.Data.Entity.DbSet.SqlQuery. Na przykład:

```
var query = context.Products.SqlQuery("EXECUTE [dbo].[GetAllProducts]");
```

Wymagania wstępne

W celu wykonania instrukcji w tym przewodniku potrzebne są następujące elementy:

- Najnowszą wersję programu Visual Studio.
- [Przykładowej bazy danych School](#).

Konfigurowanie projektu

- Otwórz program Visual Studio 2012.
- Wybierz **plikach > New -> projektu**
- W okienku po lewej stronie kliknij **Visual C#**, a następnie wybierz pozycję **konsoli** szablonu.
- Wprowadź **EFwithSProcsSample** jako nazwę.
- Wybierz **OK**.

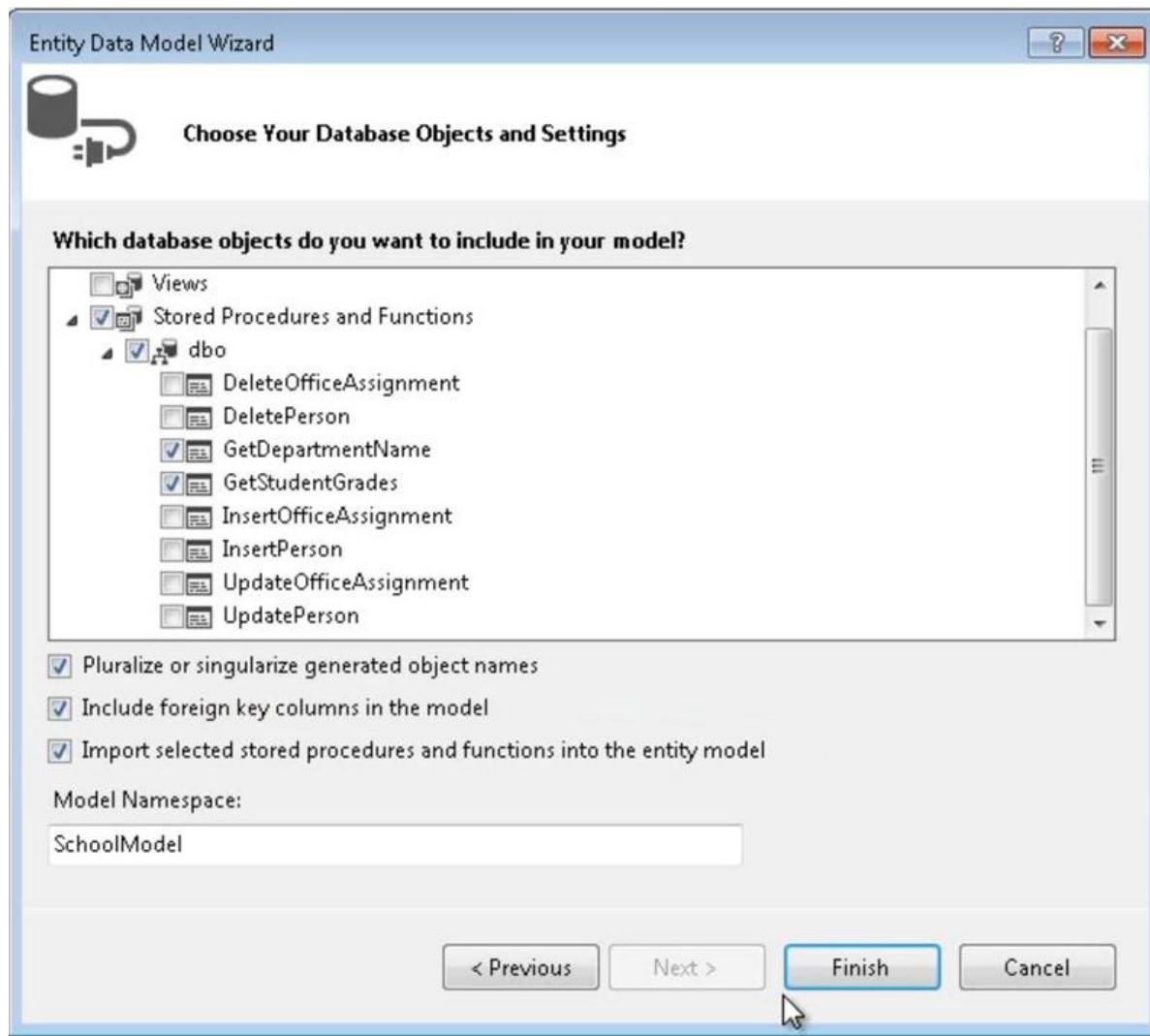
Tworzenie modelu

- Kliknij prawym przyciskiem myszy projekt w Eksploratorze rozwiązań i wybierz **Add -> nowy element**.
- Wybierz **danych** z menu po lewej stronie, a następnie wybierz pozycję **ADO.NET Entity Data Model** w okienku szablonów.
- Wprowadź **EFwithSProcsModel.edmx** nazwę pliku, a następnie kliknij przycisk **Dodaj**.
- W oknie dialogowym Wybierz zawartość modelu, wybierz **Generuj z bazy danych**, a następnie kliknij przycisk **dalej**.
- Kliknij przycisk **nowe połączenie**.

W oknie dialogowym właściwości połączenia, wprowadź nazwę serwera (na przykład **(localdb)\mssqllocaldb**), wybierz metodę uwierzytelniania, wpisz **School** nazwy bazy danych, a następnie Kliknij przycisk **OK**.

Okno dialogowe Wybierz połączenie danych jest aktualizowana ustawienie połączenia bazy danych.

- W oknie dialogowym Wybierz obiekty bazy danych sprawdź **tabel** pole wyboru, aby wszystkie tabele. Zaznacz również następujących procedur składowanych w obszarze **procedur przechowywanych i funkcji** węzła: **GetStudentGrades** i **GetDepartmentName**.



Począwszy od programu Visual Studio 2012, projektancie platformy EF obsługuje importu zbiorczego procedur składowanych. **Importowanie wybranych procedur przechowywanych i funkcji do modelu entity** jest zaznaczone domyślnie.

- Kliknij przycisk **Zakończ**.

Domyślnie kształt wynik każdego importowanego procedura składowana lub funkcja, która zwraca więcej niż jedną kolumnę automatycznie stanie się nowy typ złożony. W tym przykładzie chcemy zamapować wyniki **GetStudentGrades** funkcja **StudentGrade** jednostki oraz wyniki **GetDepartmentName** do **Brak** (**Brak** jest wartością domyślną).

Importowanie funkcji zwracać typ jednostki kolumny zwarcane przez odpowiednie procedury składowanej muszą dokładnie odpowiadać właściwości skalarne typu jednostki zwracanego. Importowanie funkcji może również zwracać kolekcji typów prostych, typów złożonych lub brak wartości.

- Kliknij prawym przyciskiem myszy projekt powierzchni i wybierz **przeglądarka modeli**.
- W **przeglądarka modeli**, wybierz opcję **Importy funkcji**, a następnie kliknij dwukrotnie **GetStudentGrades** funkcji.
- W oknie dialogowym Edytowanie importowanie funkcji, wybierz **jednostek** i wybierz polecenie **StudentGrade**.

Importowanie funkcji jest konfigurowalna pole wyboru w górnej części **Importy funkcji** okno dialogowe umożliwia mapowanie konfigurowalna funkcji. Jeśli zaznaczysz to pole tylko konfigurowalna funkcje (zwracających tabelę), będą wyświetlane w **procedury składowanej / nazwa funkcji** listy rozwijanej. Jeśli to pole wyboru nie jest zaznaczone, tylko funkcje — konfigurowalna będą wyświetlane na liście.

Użyj modelu

Otwórz **Program.cs** pliku gdzie **Main** metoda jest zdefiniowana. Dodaj następujący kod do funkcji Main.

Kod wywołuje dwie procedury składowane: **GetStudentGrades** (zwraca **StudentGrades** dla określonego **StudentId**) i **GetDepartmentName** (zwraca nazwę działu parametr wyjściowy).

```
using (SchoolEntities context = new SchoolEntities())
{
    // Specify the Student ID.
    int studentId = 2;

    // Call GetStudentGrades and iterate through the returned collection.
    foreach (StudentGrade grade in context.GetStudentGrades(studentId))
    {
        Console.WriteLine("StudentID: {0}\tSubject={1}", studentId, grade.Subject);
        Console.WriteLine("Student grade: " + grade.Grade);
    }

    // Call GetDepartmentName.
    // Declare the name variable that will contain the value returned by the output parameter.
    ObjectParameter name = new ObjectParameter("Name", typeof(String));
    context.GetDepartmentName(1, name);
    Console.WriteLine("The department name is {0}", name.Value);

}
```

Skompilować i uruchomić aplikację. Program generuje następujące wyniki:

```
StudentID: 2
Student grade: 4.00
StudentID: 2
Student grade: 3.50
The department name is Engineering
```

Parametry wyjściowe

Jeśli używane są parametry wyjściowe, ich wartości nie będzie dostępne, dopóki wyniki zostały całkowicie odczytane. Jest to spowodowane bazowego zachowanie obiekt DbDataReader, zobacz [podczas pobierania danych przy użyciu elementu DataReader](#) Aby uzyskać więcej informacji.

Procedury składowane CUD projektanta

13.09.2018 • 9 minutes to read • [Edit Online](#)

Ten przewodnik krok po kroku pokazano, jak mapować tworzenia\Wstawianie, aktualizowanie i usuwanie operacji (CUD) typu jednostki do procedur składowanych, za pomocą funkcji Entity Framework Designer (Projektant EF). Domyślnie platforma Entity Framework automatycznie generuje instrukcje SQL dla operacji CUD, ale można również mapować procedur składowanych do tych operacji.

Należy pamiętać, że Code First platformy nie obsługuje mapowania do procedury składowanej lub funkcji. Jednak można wywoływać procedury składowanej lub funkcji za pomocą metody System.Data.Entity.DbSet.SqlQuery. Na przykład:

```
var query = context.Products.SqlQuery("EXECUTE [dbo].[GetAllProducts]");
```

Uwagi dotyczące mapowania operacji CUD do procedur składowanych

Mapowanie operacji CUD do procedur składowanych, następujące kwestie:

- Jeśli mapowanie jeden operacje CUD do procedury składowanej, mapować wszystkie z nich. Jeśli nie należy mapować wszystkie trzy, niezamapowane operacji zakończy się niepowodzeniem, jeśli wykonywane i **UpdateException** zostanie zgłoszony.
- Każdy parametr procedury składowanej musi być mapowane do właściwości jednostki.
- Jeśli serwer generuje wartość klucza podstawowego dla wstawionego wiersza, możesz zamapować tę wartość do właściwości klucza jednostki. W poniższym przykładzie **InsertPerson** procedura składowana ma zwracać jako część zestawu wyników procedury składowanej nowo utworzony klucz podstawowy. Klucz podstawowy jest mapowany na klucz jednostki (**PersonID**) przy użyciu <**Dodaj powiązań wyników**> funkcji projektancie platformy EF.
- Zapisane wywołania procedur są mapowane 1:1 jednostki w modelu koncepcyjnym. Na przykład w przypadku zaimplementowania Hierarchia dziedziczenia w modelu koncepcyjnym i następnie mapy CUD procedur składowanych dla **nadrzędnego** (podstawowy) i **podrzędnych** jednostki (pochodnego), zapisywanie **Podrzędnych** zmian będzie wywoływać tylko **podrzędnych** przez procedury składowane, nie spowodują uruchomienia **nadrzędnego** użytkownika przechowywane wywołań procedur.

Wymagania wstępne

W celu wykonania instrukcji w tym przewodniku potrzebne są następujące elementy:

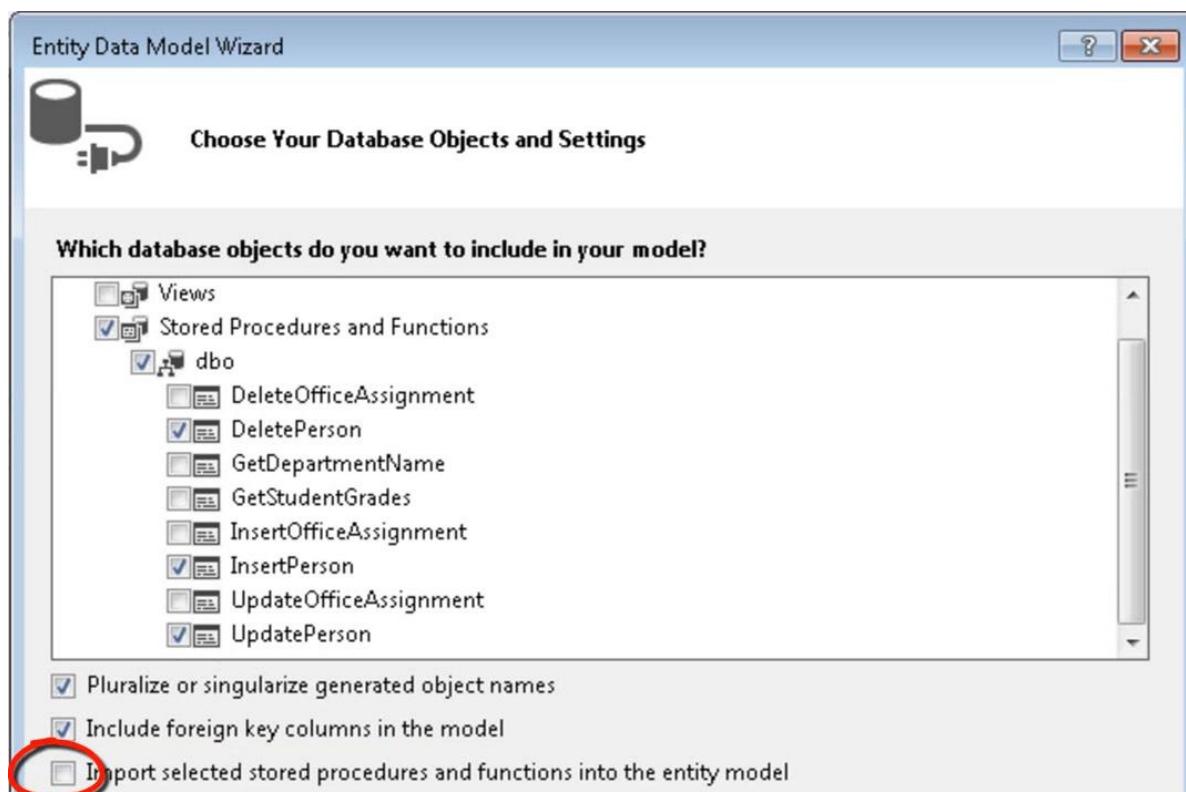
- Najnowszą wersję programu Visual Studio.
- [Przykładowej bazy danych School](#).

Konfigurowanie projektu

- Otwórz program Visual Studio 2012.
- Wybierz **plikach> New -> projektu**
- W okienku po lewej stronie kliknij **Visual C#**, a następnie wybierz pozycję **konsoli** szablonu.
- Wprowadź **CUDSProcsSample** jako nazwę.
- Wybierz **OK**.

Tworzenie modelu

- Kliknij prawym przyciskiem myszy nazwę projektu w Eksploratorze rozwiązań, a następnie wybierz pozycję **Add -> nowy element**.
- Wybierz **danych** z menu po lewej stronie, a następnie wybierz pozycję **ADO.NET Entity Data Model** w okienku szablonów.
- Wprowadź **CUDSProcs.edmx** nazwę pliku, a następnie kliknij przycisk **Dodaj**.
- W oknie dialogowym Wybierz zawartość modelu, wybierz **Generuj z bazy danych**, a następnie kliknij przycisk **dalej**.
- Kliknij przycisk **nowe połączenie**. W oknie dialogowym właściwości połączenia, wprowadź nazwę serwera (na przykład **(localdb)\mssqllocaldb**), wybierz metodę uwierzytelniania, wpisz **School** nazwy bazy danych, a następnie Kliknij przycisk **OK**. Okno dialogowe Wybierz połączenie danych jest aktualizowana ustawienie połączenia bazy danych.
- W polu Wybierz obiekty bazy danych okna dialogowego w polu **tabel** węzeł **osoby** tabeli.
- Zaznacz również następujących procedur składowanych w obszarze **procedur przechowywanych i funkcji** węzła: **DeletePerson**, **InsertPerson**, i **UpdatePerson**.
- Począwszy od programu Visual Studio 2012, projektancie platformy EF obsługuje importu zbiorczego procedur składowanych. **Importowanie wybranych procedur przechowywanych i funkcji do modelu jednostki** jest zaznaczone domyślnie. Ponieważ w tym przykładzie firma Microsoft ma procedur składowanych, które Wstawianie, aktualizowanie i usuwanie typów jednostek, firma Microsoft nie chcesz zaimportować je, a następnie będzie Usuń zaznaczenie tego pola wyboru.



- Kliknij przycisk **Zakończ**. Zostanie wyświetlona projektancie platformy EF, oferujący powierzchnię projektową do edycji modelu.

Mapuj jednostki osoby do procedur składowanych

- Kliknij prawym przyciskiem myszy **osoby** typu jednostki, a następnie wybierz **mapowanie procedur**

przechowywanych.

- Procedura składowana mapowania są wyświetlane w **szczegóły mapowania** okna.
- Kliknij przycisk <**wybierz opcję Wstaw funkcja**>. Pole staje się listy rozwijanej, procedur składowanych w modelu magazynu, które mogą być mapowane na typy jednostek w modelu koncepcyjnym. Wybierz **InsertPerson** z listy rozwijanej.
- Domyślne mapowania między parametrów procedury składowanej i właściwości obiektu są wyświetlane. Należy pamiętać, że strzałki wskazują kierunek mapowania: wartości właściwości są dostarczane do parametrów procedury składowanej.
- Kliknij przycisk <**Dodaj powiązanie wynik**>.
- Typ **NewPersonID**, nazwa parametru zwrócony przez **InsertPerson** procedury składowanej. Upewnij się, że nie wpisz spacji wiodących i końcowych.
- Naciśnij klawisz **wprowadź**.
- Domyślnie **NewPersonID** jest mapowany na klucz jednostki **PersonID**. Należy pamiętać, że strzałka wskazuje kierunek mapowania: wartość kolumny wynik jest dostarczany do właściwości.

Mapping Details - Person					
Parameter / Column	Operator	Property	Use Original...	Rows Affected	Parameter
Functions					
Insert Using InsertPerson					
Parameters					
@ LastName : nvarchar	←	LastName : String			
@ FirstName : nvarchar	←	FirstName : String			
@ HireDate : datetime	←	HireDate : DateTime			
@ EnrollmentDate : datetime	←	EnrollmentDate : DateTime			
@ Discriminator : nvarchar	←	Discriminator : String			
Result Column Bindings					
NewPersonID	→	PersonID : Int32			

- Kliknij przycisk <**wybierz funkcję Update**> i wybierz **UpdatePerson** z wyświetlonej listy rozwijanej.
- Domyślne mapowania między parametrów procedury składowanej i właściwości obiektu są wyświetlane.
- Kliknij przycisk <**funkcji wybierz pozycję Usuń**> i wybierz **DeletePerson** z wyświetlonej listy rozwijanej.
- Domyślne mapowania między parametrów procedury składowanej i właściwości obiektu są wyświetlane.

Wstawianie, aktualizowanie i usuwanie operacji **osoby** typu jednostki są teraz mapowane na procedury składowane.

Jeśli chcesz włączyć współbieżności sprawdzania podczas aktualizowania lub usuwania jednostki za pomocą procedur składowanych, użyj jednej z następujących opcji:

- Użyj **dane wyjściowe** parametru, aby zwrócić liczbę wierszy z procedury składowanej i sprawdź, których to dotyczy **parametr na wiersze** pole wyboru obok nazwy parametru. Jeśli wartość zwracana wynosi zero, gdy operacja jest wywoływana, **OptimisticConcurrencyException** zostanie zgłoszony.
- Sprawdź **Użyj oryginalnej wartości** pole wyboru obok właściwości, którą chcesz używać sprawdzania współbieżności. Próba aktualizacji wartości właściwości, która pierwotnie została odczytana z bazy danych będzie używany podczas zapisywania danych z powrotem do bazy danych. Jeśli wartość jest niezgodna z wartością w bazie danych **OptimisticConcurrencyException** zostanie zgłoszony.

Użyj modelu

Otwórz **Program.cs** pliku gdzie **Main** metoda jest zdefiniowana. Dodaj następujący kod do funkcji Main.

Ten kod tworzy nową **osoby** obiektu, następnie aktualizuje obiekt, a następnie usuwa obiekt.

```

using (var context = new SchoolEntities())
{
    var newInstructor = new Person
    {
        FirstName = "Robyn",
        LastName = "Martin",
        HireDate = DateTime.Now,
        Discriminator = "Instructor"
    }

    // Add the new object to the context.
    context.People.Add(newInstructor);

    Console.WriteLine("Added {0} {1} to the context.",
        newInstructor.FirstName, newInstructor.LastName);

    Console.WriteLine("Before SaveChanges, the PersonID is: {0}",
        newInstructor.PersonID);

    // SaveChanges will call the InsertPerson sproc.
    // The PersonID property will be assigned the value
    // returned by the sproc.
    context.SaveChanges();

    Console.WriteLine("After SaveChanges, the PersonID is: {0}",
        newInstructor.PersonID);

    // Modify the object and call SaveChanges.
    // This time, the UpdatePerson will be called.
    newInstructor.FirstName = "Rachel";
    context.SaveChanges();

    // Remove the object from the context and call SaveChanges.
    // The DeletePerson sproc will be called.
    context.People.Remove(newInstructor);
    context.SaveChanges();

    Person deletedInstructor = context.People.
        Where(p => p.PersonID == newInstructor.PersonID).
        FirstOrDefault();

    if (deletedInstructor == null)
        Console.WriteLine("A person with PersonID {0} was deleted.",
            newInstructor.PersonID);
}

```

- Skompilować i uruchomić aplikację. Program generuje następujące dane wyjściowe *

NOTE

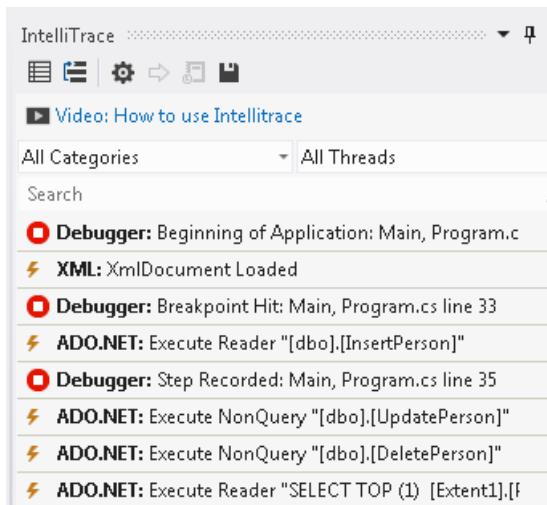
PersonID został wygenerowany automatycznie przez serwer, więc najprawdopodobniej zobaczysz liczbę różnych *

```

Added Robyn Martin to the context.
Before SaveChanges, the PersonID is: 0
After SaveChanges, the PersonID is: 51
A person with PersonID 51 was deleted.

```

Jeśli pracujesz z programu Visual Studio w wersji Ultimate, umożliwia Intellitrace za pomocą debugera zobaczyć instrukcje SQL, które są wykonywane.



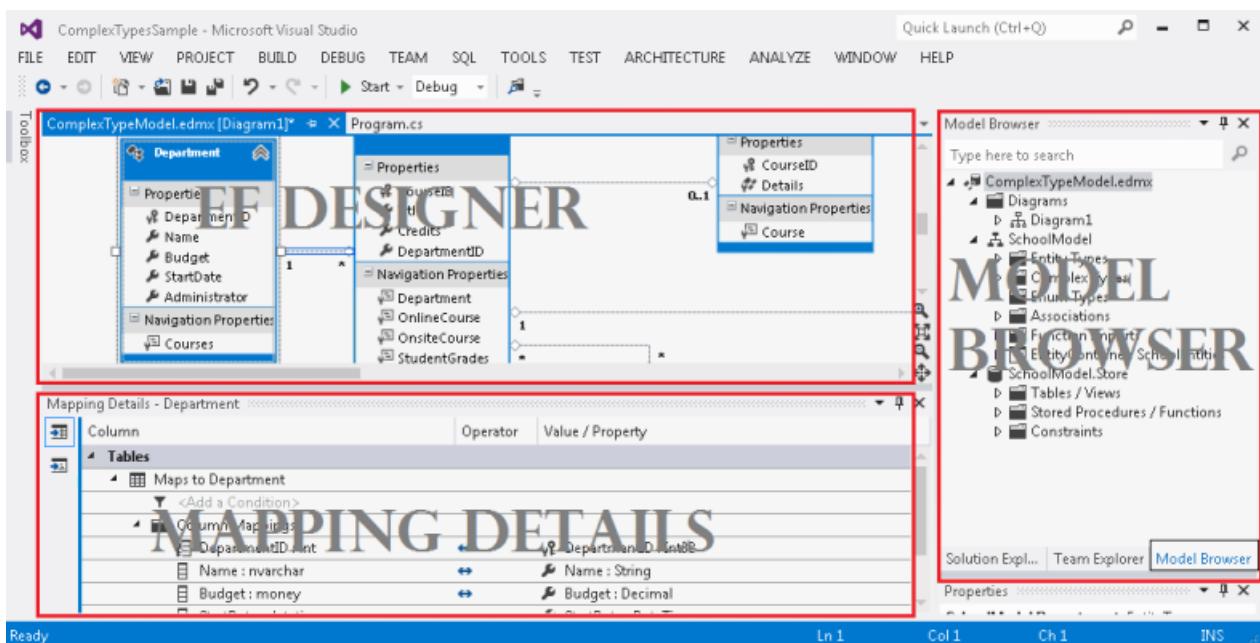
Relacje — projektancie platformy EF

13.09.2018 • 9 minutes to read • [Edit Online](#)

NOTE

Ta strona zawiera informacje o konfigurowaniu relacji w modelu przy użyciu projektanta EF. Aby uzyskać ogólne informacje dotyczące relacji w EF i uzyskiwania dostępu i manipulowanie danymi za pomocą relacji, zobacz [relacje & właściwości nawigacji](#).

Skojarzenia definiowania relacji między typami encji w modelu. W tym temacie pokazano, jak mapować skojarzenia z Entity Framework Designer (Projektant EF). Na poniżej ilustracji przedstawiono główne systemu windows, które są używane podczas pracy z projektancie platformy EF.



NOTE

Podczas tworzenia modelu koncepcyjnego ostrzeżenia dotyczące podmiotów niezmapowanych i skojarzenia może pojawić się na liście błędów. Można zignorować te ostrzeżenia, ponieważ po dokonaniu wyboru wygenerować bazę danych z modelu, błędy znikną.

Omówienie skojarzenia

Podczas projektowania modelu przy użyciu projektanta EF plik edmx reprezentuje model. W pliku edmx **skojarzenia** element definiuje relację między dwoma typami encji. Skojarzenia należy określić typy jednostek, które są zaangażowane w relacji i możliwa liczba typów jednostek na każdym końcu relacji, który jest znany jako liczebności. Liczebność elementu end skojarzenia mogą mieć wartość równą jeden (1), zero lub jeden (od 0 do 1) lub wielu (*). Informacja ta jest określona w dwóch podrzędny **zakończenia** elementów.

W czasie wykonywania wystąpienia typu jednostki na jednym końcu asocjacji jest możliwy za pośrednictwem właściwości nawigacji lub klucze obce (Jeśli użytkownik chce udostępnić klucze obce w jednostkach). Za pomocą kluczy obcych widoczne, relacji między jednostkami odbywa się za pomocą **ReferentialConstraint** elementu (element podrzędny **skojarzenia** elementu). Zalecane jest, należy zawsze udostępnić klucze obce w przypadku relacji w jednostkach.

NOTE

W wielu do wielu (*:*) klucze obce nie można dodać do jednostki. W *:* relacji, informacji o skojarzeniu odbywa się za pomocą tworzenie niezależnych obiektów.

Aby uzyskać informacje o elementach CSDL (**ReferentialConstraint**, **skojarzenia**, itp.) zobacz [Specyfikacja CSDL](#).

Tworzenie i usuwanie skojarzenia

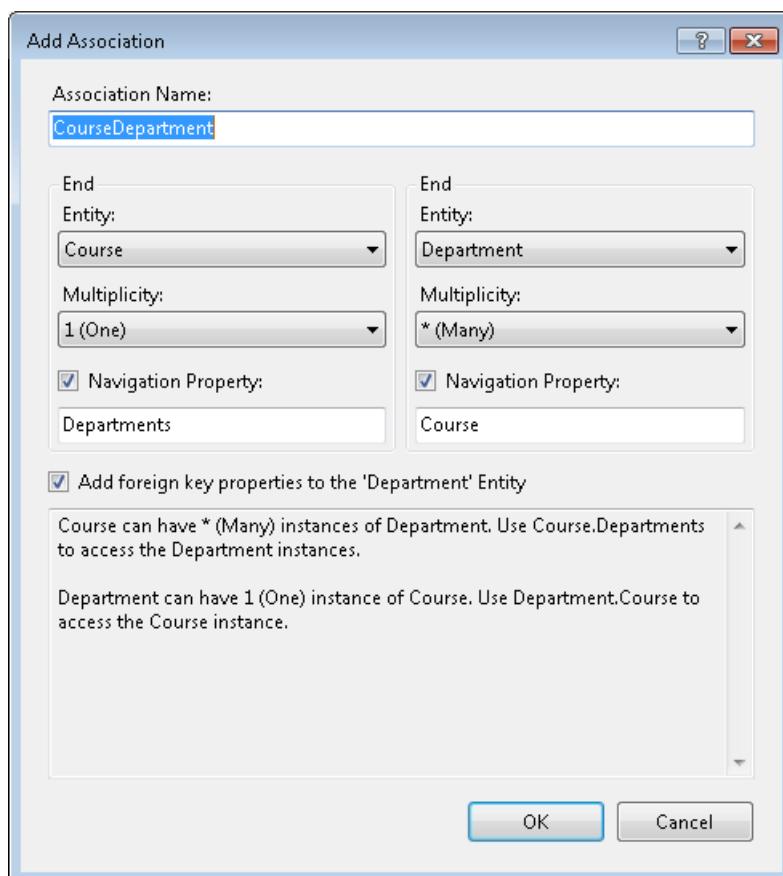
Tworzenie skojarzenia z aktualizacjami projektancie platformy EF modelu zawartości pliku edmx. Po utworzeniu skojarzenia, należy utworzyć mapowania dla skojarzenia (zostało to omówione w dalszej części tego tematu).

NOTE

W tej sekcji założono, dodano już jednostkami, z którymi chcesz utworzyć skojarzenie między do modelu.

Aby utworzyć skojarzenie

1. Kliknij prawym przyciskiem myszy pusty obszar powierzchni projektu, wskaź opcję **Dodaj nowej** wybierz **skojarzenie...**.
2. Wypełnij ustawienia dla skojarzenia w **Dodawanie skojarzenia** okna dialogowego.



NOTE

Użytkownik może nie dodać właściwości nawigacji lub właściwości klucza obcego z jednostkami: końcach asocjacji, czyszcząc ** właściwość nawigacji ** i ** dodać właściwości klucza obcego do <Nazwa typu jednostki> jednostki ** pola wyboru. Jeśli dodasz tylko jedną właściwość nawigacji, stowarzyszenia będą traversable tylko w jednym kierunku. Jeśli dodasz żadnych właściwości nawigacji, użytkownik musi dodać właściwości klucza obcego w celu uzyskania dostępu do jednostek na końcach asocjacji.

3. Kliknij przycisk **OK**.

Aby usunąć skojarzenie

Aby usunąć czy skojarzenie, jedną z następujących czynności:

- Kliknij prawym przyciskiem myszy skojarzenia w Projektancie platformy EF powierzchni i wybierz **Usuń**.
- LUB —
- Wybierz jeden lub więcej skojarzeń, a następnie naciśnij klawisz **DELETE**.

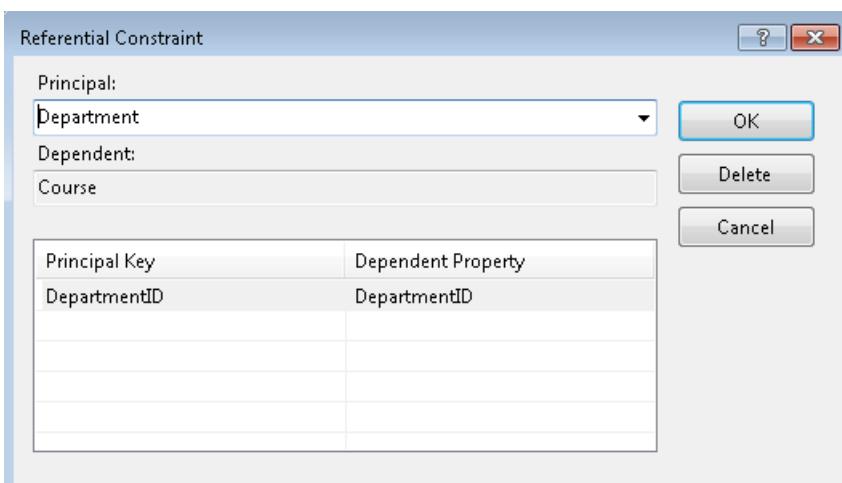
Zawierają właściwości klucza obcego w jednostkach (ograniczenia referencyjne)

Zalecane jest, należy zawsze udostępnić klucze obce w przypadku relacji w jednostkach. Platformy Entity Framework używa ograniczenia referencyjnego do identyfikowania, że właściwość działa jako klucz obcy relacji.

Gdy zaznaczono **właściwości klucza obcego, aby dodać <Nazwa typu jednostki> jednostki** pola wyboru przy tworzeniu relacji, to ograniczenie referencyjne została dodana dla Ciebie.

Gdy używasz projektancie platformy EF umożliwiają dodawanie lub edytowanie ograniczenia referencyjnego projektancie platformy EF dodaje lub modyfikuje **ReferentialConstraint** element CSDL zawartość pliku edmx.

- Kliknij dwukrotnie skojarzenia, które chcesz edytować. **Ograniczenia referencyjnego** pojawi się okno dialogowe.
- Z **jednostki** listy rozwijanej wybierz jednostkę główną w ograniczeniu referencyjnym. Właściwości klucza jednostki są dodawane do **klucz jednostki** listy w oknie dialogowym.
- Z **zależne** listy rozwijanej wybierz jednostki zależne w ograniczeniu referencyjnym.
- Dla każdego klucza podmiotu zabezpieczeń, kluczu zależnych, wybierz odpowiedni klucz zależne z listy rozwijanej w **zależne klucz** kolumny.



- Kliknij przycisk **OK**.

Tworzenie i edytowanie mapowania skojarzenia

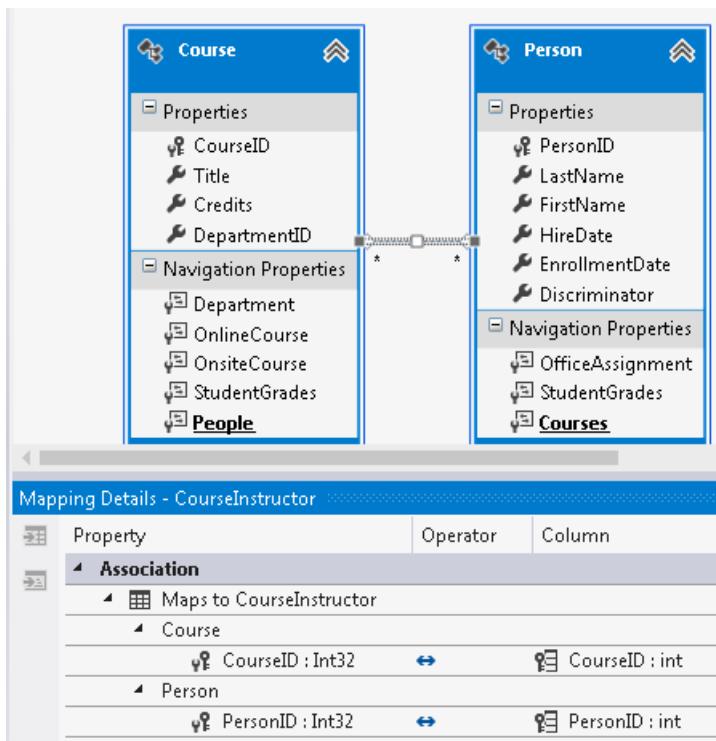
Można określić sposób mapowania bazy danych w skojarzenie **szczegóły mapowania** okna projektancie platformy EF.

NOTE

Można mapować tylko szczegółów dotyczących skojarzeń, które nie mają określone ograniczenia referencyjnego. Jeśli określono ograniczenia referencyjnego właściwości klucza obcego znajduje się w jednostce i szczegóły mapowania można użyć tej jednostki na kolumny klucza obcego mapuje do kontroli.

Utwórz mapowanie skojarzenia

- Kliknij prawym przyciskiem myszy skojarzenie w projekt powierzchni i wybierz **mapowania tabeli**. Spowoduje to wyświetlenie mapowanie skojarzenia w **szczegóły mapowania** okna.
- Kliknij przycisk **Dodaj tabelę lub widok**. Listy rozwijanej pojawia się, który zawiera wszystkie tabele w modelu magazynu.
- Wybierz tabelę, do której będzie zmapowana skojarzenia. **Szczegóły mapowania** okno wyświetla obu końcach asocjacji i właściwości klucza dla typu jednostki, w każdej **zakończenia**.
- Dla każdej właściwości klucza, kliknij przycisk **kolumny** pola, a następnie wybierz kolumnę, do którego będzie zmapowana właściwości.



Edytuj mapowanie skojarzenia

- Kliknij prawym przyciskiem myszy skojarzenie w projekt powierzchni i wybierz **mapowania tabeli**. Spowoduje to wyświetlenie mapowanie skojarzenia w **szczegóły mapowania** okna.
- Kliknij przycisk **mapuje <nazwy tabeli>**. Listy rozwijanej pojawia się, który zawiera wszystkie tabele w modelu magazynu.
- Wybierz tabelę, do której będzie zmapowana skojarzenia. **Szczegóły mapowania** okno wyświetla obu końcach asocjacji i właściwości klucza dla typu jednostki, na każdym końcu.
- Dla każdej właściwości klucza, kliknij przycisk **kolumny** pola, a następnie wybierz kolumnę, do którego będzie zmapowana właściwości.

Edytuj i Usuń właściwości nawigacji

Właściwości nawigacji są właściwościami skrótów, które są używane do lokalizowania jednostek na końcach asocjacji w modelu. Podczas tworzenia skojarzenia między dwoma typami jednostki można utworzyć właściwości nawigacji.

Aby edytować właściwości nawigacji

- Wybierz właściwość nawigacji na powierzchni projektanta EF. W programie Visual Studio wyświetlane są informacje dotyczące właściwości nawigacji **właściwości** okna.
- Zmień ustawienia właściwości **właściwości** okna.

Można usunąć właściwości nawigacji

- Jeśli klucze obce nie są widoczne na typy jednostek w modelu koncepcyjnym, usunięcie właściwości nawigacji może spowodować odpowiedniego skojarzenia traversable tylko w jednym kierunku lub nie traversable wcale.
- Kliknij prawym przyciskiem myszy właściwość nawigacji w Projektancie platformy EF powierzchni i wybierz **Usuń**.

Wiele diagramów na modelu

27.09.2018 • 7 minutes to read • [Edit Online](#)

NOTE

EF5 poczawszy tylko — funkcje, interfejsów API itp. z opisem na tej stronie zostały wprowadzone w programie Entity Framework 5. Jeśli używasz starszej wersji, niektóre lub wszystkie informacje, nie ma zastosowania.

Ten film wideo i stronie pokazuje, jak podzielić modelu wiele diagramów przy użyciu narzędzia Entity Framework Designer (Projektant EF). Można użyć tej funkcji, gdy model staje się zbyt duży, aby wyświetlić lub edytować.

We wcześniejszych wersjach programu projektancie platformy EF jednym diagramie można mieć tylko dla pliku EDMX. Począwszy od programu Visual Studio 2012, umożliwia projektancie platformy EF Podziel plik EDMX na wiele diagramów.

Obejrzyj wideo

To wideo pokazuje, jak podzielić modelu wiele diagramów przy użyciu narzędzia Entity Framework Designer (Projektant EF). Można użyć tej funkcji, gdy model staje się zbyt duży, aby wyświetlić lub edytować.

Osoba prezentująca: Julia Kornich

Film wideo: [WMV](#) | [MP4](#) | [WMV \(ZIP\)](#)

Omówienie projektanta EF

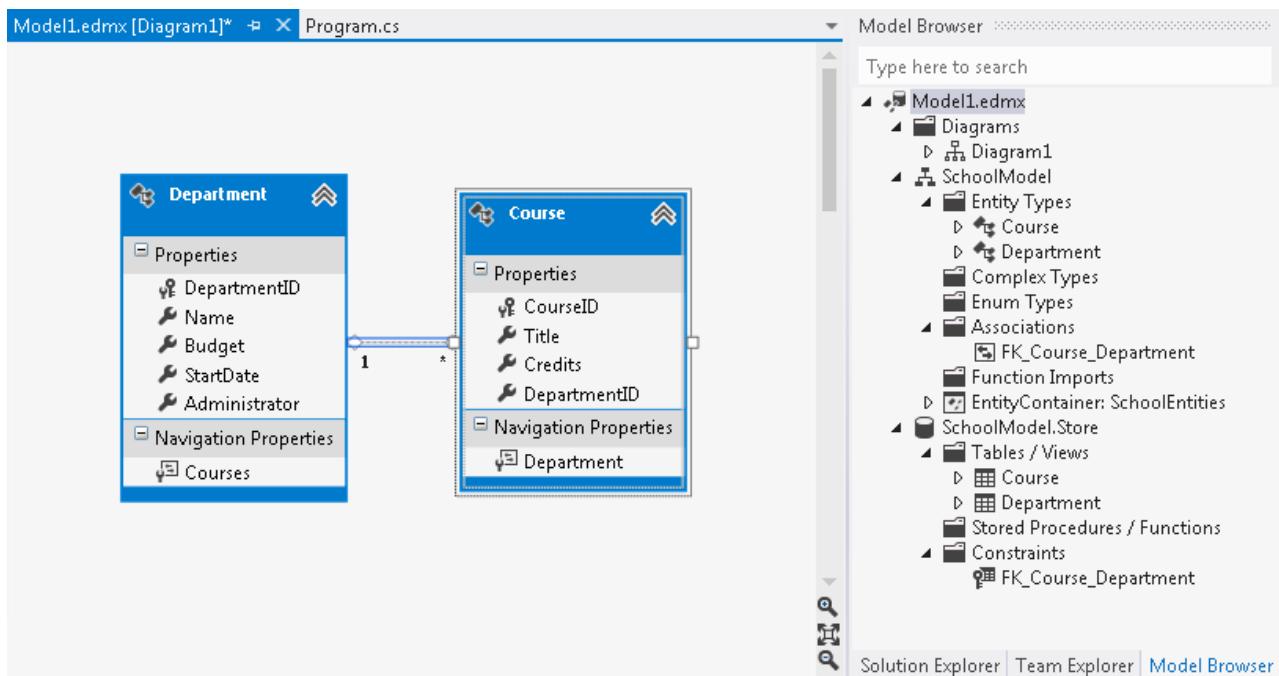
Podczas tworzenia modelu przy użyciu Kreatora modelu danych jednostki w Projektancie platformy EF plik edmx jest utworzony i dodany do rozwiązania. Ten plik definiuje kształt jednostek i sposobu mapowania ich na bazie danych.

W Projektancie platformy EF składa się z następujących składników:

- Powierzchnia projektowania wizualnego służące do edycji modelu. Tworzenia, modyfikowania lub usuwania jednostki i skojarzenia.
- A **przeglądarka modeli** okna, który zawiera widoki drzewa modelu. Jednostek i ich skojarzenia, które znajdują się w obszarze *[ModelName]* folderu. Tabele bazy danych i ograniczeń, które znajdują się w obszarze *[ModelName]. Store* folder.
- A **szczegółы mapowania** okna do wyświetlania i edytowania mapowania. Typy jednostek i skojarzenia można mapować do bazy danych tabel, kolumn i procedur składowanych.

Po zakończeniu działania Kreator modelu Entity Data Model automatycznie otwieranie okna powierzchni projektowania wizualnego. Jeśli przeglądarka modelu nie jest widoczna, kliknij prawym przyciskiem myszy główny projekt powierzchni i wybierz **przeglądarka modeli**.

Poniższy zrzut ekranu przedstawia plik edmx otwarty w Projektancie platformy EF. Na zrzucie ekranu przedstawiono powierzchnię projektowania wizualnego (do lewej) i **przeglądarka modeli** okno (z prawej strony).



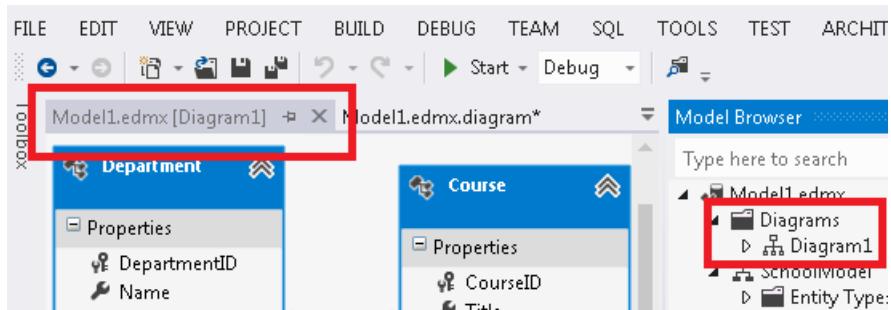
Aby cofnąć operacji w Projektancie platformy EF, kliknij przycisk klawisze Ctrl + Z.

Praca z diagramami

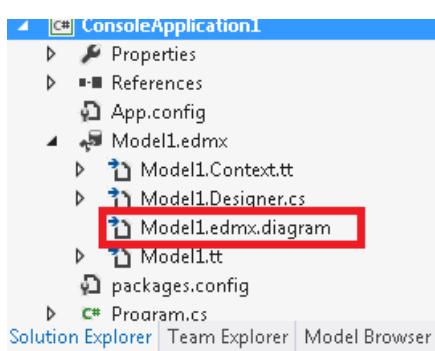
Domyślnie w Projektancie platformy EF tworzy jeden diagram o nazwie Diagram1. W przypadku diagramu z dużą liczbą jednostek i skojarzenia będzie najczęściej chcesz chcesz podzielić logicznie. Począwszy od programu Visual Studio 2012, można wyświetlić model koncepcyjny w wiele diagramów.

W miarę dodawania nowych diagramów pojawiają się w folderze diagramy w oknie przeglądarki modelu. Aby zmienić nazwę diagramu: Wybierz diagram w oknie przeglądarki modelu, kliknij jeden raz na nazwę i wpisz nową nazwę. Możesz również kliknąć prawym przyciskiem myszy nazwę diagramu i wybierz **Zmień nazwę**.

Nazwa diagramu jest wyświetlany obok nazwy pliku edmx w edytorze programu Visual Studio. Na przykład Model1.edmx[Diagram1].



Zawartość diagramy (ksztalt i kolor encji i skojarzeń) jest przechowywana w. edmx.diagram pliku. Aby wyświetlić ten plik, wybierz w Eksploratorze rozwiązań i rozwijania pliku edmx.



Nie należy edytować edmx.diagram ręcznie, plik zawartości tego pliku może być zastąpiona przez projektancie platformy EF.

Dzielenie jednostek i skojarzenia do nowego diagramu

Możesz wybrać jednostek na diagramie istniejących (naciśnij i przytrzymaj klawisz Shift, aby wybrać wiele jednostek). Kliknij prawym przyciskiem myszy i wybierz pozycję **przenieść do nowego diagramu**. Zostanie utworzony nowy diagram i wybranych obiektów i ich skojarzenia zostaną przeniesione do diagramu.

Alternatywnie, kliknij prawym przyciskiem myszy folder diagramy w przeglądarce modelu i wybierz **Dodaj nowy Diagram**. Można następnie przeciągnij i upuść jednostek jest dostępna z folderu typów jednostek w przeglądarce modelu na powierzchnię projektu.

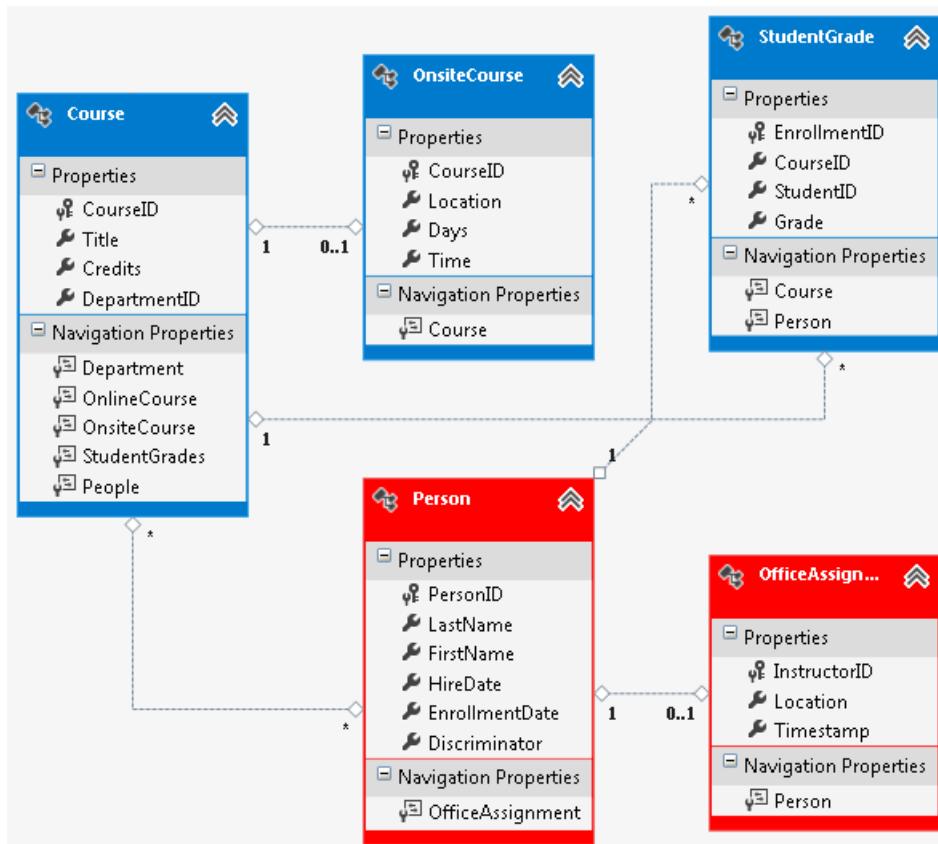
Można również Wytnij lub Kopiuj jednostki (przy użyciu klawiszy Ctrl-X lub Ctrl-C) z jednym diagramie i wkleić (przy użyciu klawisza Ctrl-V) na drugim. Diagram, w której wklejasz jednostki już zawiera jednostkę o takiej samej nazwie, Nowa jednostka zostanie utworzona i dodana do modelu. Na przykład: Diagram2 zawiera jednostkę działu. Następnie możesz wkleić innego działu, na Diagram2. Jednostka Department1 zostanie utworzony i dodany do modelu koncepcyjnego.

Aby dołączyć powiązanych jednostek na diagramie, kliknij rick jednostki i wybierz **obejmują powiązane**. Spowoduje to kopią powiązanych jednostek i skojarzenia na diagramie określony.

Zmiana koloru jednostek

Oprócz dzieląc modelu na wiele diagramów, można również zmienić kolory jednostek.

Aby zmienić kolor, wybierz jednostki (lub wiele jednostek) na powierzchni projektowej. Następnie kliknij prawym przyciskiem myszy i wybierz **właściwości**. W oknie dialogowym właściwości wybierz **kolor wypełnienia** właściwości. Określanie koloru za pomocą prawidłową nazwą koloru (na przykład kolor czerwony) albo prawidłowy RGB, (na przykład, 255, 128, 128).



Podsumowanie

W tym temacie przyjrzelimy się jak podzielić modelu wiele diagramów, a także jak określić innego koloru dla jednostki za pomocą programu Entity Framework Designer.

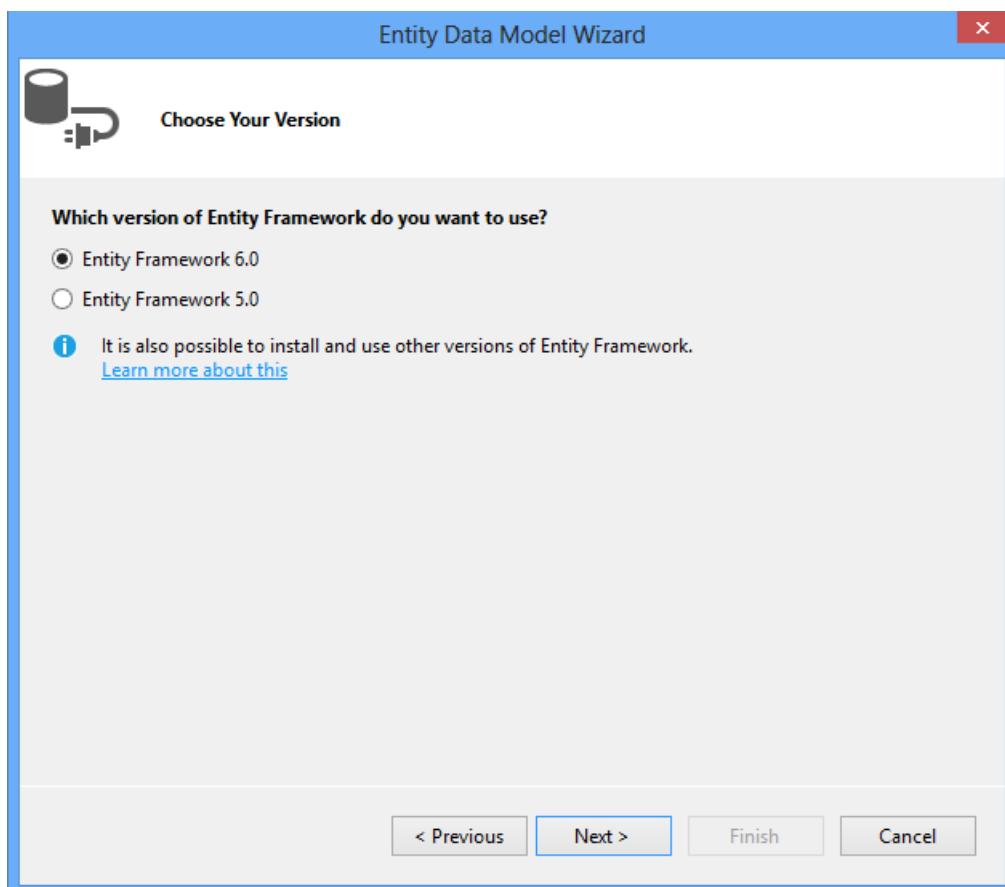
Wybieranie jednostki wersję środowiska uruchomieniowego EF projektanta modeli

13.09.2018 • 3 minutes to read • [Edit Online](#)

NOTE

EF6 począwszy tylko — funkcje, interfejsów API itp. z opisem na tej stronie zostały wprowadzone w programie Entity Framework 6. Jeśli używasz starszej wersji, niektóre lub wszystkie informacje, nie ma zastosowania.

Począwszy od platformy EF6 dodano następujący ekran projektancie platformy EF, aby możliwe było wybrać wersję środowiska uruchomieniowego, który ma pod kątem podczas jego tworzenia. Ekran pojawi się na najnowszą wersję programu Entity Framework nie jest już zainstalowany w projekcie. Jeśli już jest zainstalowana najnowsza wersja będzie używany tylko domyślnie.



Określanie wartości docelowej EF6.x

Możesz wybrać platformę EF6 z ekranu wybierz wersję usługi, aby dodać środowiska uruchomieniowego platformy EF6 do projektu. Po dodaniu EF6 należy zatrzymać, widzisz ten ekran w bieżącym projekcie.

EF6 zostanie wyłączona, jeśli masz już starszą wersję programu EF zainstalowany (ponieważ nie może wskazać wiele wersji środowiska uruchomieniowego, w tym samym projekcie). Jeśli opcja EF6 nie jest włączona w tym miejscu, wykonaj następujące kroki, aby uaktualnić projekt do EF6:

1. Kliknij prawym przyciskiem myszy projekt w Eksploratorze rozwiązań i wybierz **Zarządzaj pakietami NuGet...**
2. Wybierz **aktualizacji**

3. Wybierz **EntityFramework** (Upewnij się, będzie ona konieczność jej zaktualizowania do wersji mają)
4. Kliknij przycisk **aktualizacji**

Określanie wartości docelowej EF5.x

Możesz wybrać EF5 z ekranu wybierz wersję usługę, aby dodać środowiska uruchomieniowego EF5 do projektu. Po dodaniu EF5 nadal zobaczysz ekran z opcją EF6 wyłączone.

Jeśli masz EF4.x wersję środowiska uruchomieniowego już zainstalowanego zostanie wyświetcone tej wersji na liście ekranu, a nie EF5 EF. W takiej sytuacji można przeprowadzić uaktualnienie do EF5 wykonując następujące czynności:

1. Wybierz **Tools -> Menedżer pakietów biblioteki -> Konsola Menedżera pakietów**
2. Uruchom **EntityFramework instalacji pakietu-wersja 5.0.0**

Określanie wartości docelowej EF4.x

Można zainstalować środowisko uruchomieniowe EF4.x do projektu wykonując następujące czynności:

1. Wybierz **Tools -> Menedżer pakietów biblioteki -> Konsola Menedżera pakietów**
2. Uruchom **EntityFramework instalacji pakietu — w wersji 4.3.0**

Szablonów generowania kodu projektanta

27.09.2018 • 13 minutes to read • [Edit Online](#)

Podczas tworzenia modelu przy użyciu programu Entity Framework Designer klas i kontekst pochodna są generowane automatycznie dla Ciebie. Oprócz generowania kodu domyślne oferujemy są również szablony, których można dostosować program code, który pobiera wygenerowany. Te szablony stanowią one szablonów tekstowych T4, dzięki czemu możesz dostosowywać szablony, jeśli to konieczne.

Kod, który pobiera wygenerowany domyślnie, zależy od wersji programu Visual Studio Tworzenie modelu w:

- Modele utworzone w programie Visual Studio 2012 i 2013 generuje prosty klas obiektów POCO i kontekście, który pochodzi z uproszczonego DbContext.
- Modele utworzone w programie Visual Studio 2010, spowoduje wygenerowanie klas jednostek, które wynikają z EntityObject i kontekstu, która pochodzi z obiektu ObjectContext.

NOTE

Firma Microsoft zaleca, przełączanie do szablonu DbContext Generator po dodaniu modelu.

Ta strona obejmuje dostępnych szablonów i następnie zawiera instrukcje dotyczące dodawania szablonu do modelu.

Dostępne szablony

Poniższe szablony są dostarczane przez zespół programu Entity Framework:

DbContext Generator

Ten szablon generuje prosty klas obiektów POCO i kontekstu, która pochodzi od typu DbContext przy użyciu platformy EF6. To jest szablon zalecana, chyba że masz powód, aby użyć jednego z innych szablonów wymienionych poniżej. Jest również szablon generowania kodu, Pobierz domyślnie, jeśli używane są nowe wersje programu Visual Studio (Visual Studio 2013 lub nowszy): po utworzeniu nowego modelu ten szablon jest używany domyślnie i pliki T4 (.tt) zostały zagnieździone w pliku edmx.

Starsze wersje programu Visual Studio

- **Program Visual Studio 2012:** można pobrać **EF 6.x DbContextGenerator** szablonów, musisz zainstalować najnowszą **Entity Framework Tools for Visual Studio** — zobacz [pobrać jednostki Framework](#) strony, aby uzyskać więcej informacji.
- **Program Visual Studio 2010: EF 6.x DbContextGenerator** szablony nie są dostępne dla programu Visual Studio 2010.

Generator DbContext dla platformy EF 5.x

Jeśli używasz starszej wersji pakietu EntityFramework NuGet (po jednej z wersji głównej 5) należy użyć **EF 5.x DbContext Generator** szablonu.

Jeśli używasz programu Visual Studio 2013 lub 2012 ten szablon jest już zainstalowana.

Jeśli używasz programu Visual Studio 2010 będzie konieczne wybranie **Online** karcie podczas dodawania szablonu, aby go pobrać z galerii Visual Studio. Alternatywnie można zainstalować szablonu bezpośrednio z galerii programu Visual Studio wcześniej. Ponieważ szablony są wliczone w nowszych wersjach programu Visual Studio w wersji w galerii można zainstalować tylko w programie Visual Studio 2010.

- [EF 5.x DbContext Generator dla języka C#](#)

- [EF 5.x DbContext Generator dla witryn sieci Web w języku C#](#)
- [EF 5.x DbContext Generator VB.NET](#)
- [EF 5.x DbContext Generator dla witryn sieci Web VB.NET](#)

Generator DbContext dla platformy EF 4.x

Jeśli używasz starszej wersji pakietu EntityFramework NuGet (po jednym z głównych wersji 4) należy użyć **EF 4.x DbContext Generator** szablonu. To można znaleźć w **Online** karcie podczas dodawania szablonu lub szablonu można zainstalować bezpośrednio z galerii programu Visual Studio wcześniej.

- [EF 4.x DbContext Generator dla języka C#](#)
- [EF 4.x DbContext Generator dla witryn sieci Web w języku C#](#)
- [EF 4.x DbContext Generator VB.NET](#)
- [EF 4.x DbContext Generator dla witryn sieci Web VB.NET](#)

EntityObject Generator

Ten szablon spowoduje wygenerowanie klas jednostek, które wynikają z EntityObject i kontekstu, która pochodzi z obiektu ObjectContext.

NOTE

Należy wziąć pod uwagę przy użyciu generatora DbContext

DbContext Generator jest teraz zalecane szablon dla nowych aplikacji. DbContext Generator wykorzystuje prostsze API DbContext. EntityObject Generator jest nadal dostępne do obsługi istniejących aplikacji.

Visual Studio 2010, 2012 & 2013

Będzie konieczne wybranie **Online** karcie podczas dodawania szablonu, aby go pobrać z galerii Visual Studio. Alternatywnie można zainstalować szablonu bezpośrednio z galerii programu Visual Studio wcześniej.

- [EF 6.x EntityObject Generator dla języka C#](#)
- [EF 6.x EntityObject Generator dla witryn sieci Web w języku C#](#)
- [EF 6.x EntityObject Generator VB.NET](#)
- [EF 6.x EntityObject Generator dla witryn sieci Web VB.NET](#)

Generator EntityObject dla platformy EF 5.x

Jeśli używasz programu Visual Studio 2012 lub 2013 będzie konieczne wybranie **Online** karcie podczas dodawania szablonu, aby go pobrać z galerii Visual Studio. Alternatywnie można zainstalować szablonu bezpośrednio z galerii programu Visual Studio wcześniej. Ponieważ szablony są wliczone w programie Visual Studio 2010 wersji w galerii można zainstalować tylko w programie Visual Studio 2012 & 2013.

- [EF 5.x EntityObject Generator dla języka C#](#)
- [EF 5.x EntityObject Generator dla witryn sieci Web w języku C#](#)
- [EF 5.x EntityObject Generator VB.NET](#)
- [EF 5.x EntityObject Generator dla witryn sieci Web VB.NET](#)

Wystarczy ObjectContext generowania kodu bez konieczności edytowania szablonu możesz [powrócić do generowania kodu EntityObject](#).

Jeśli używasz programu Visual Studio 2010, ten szablon jest już zainstalowana. Jeśli tworzysz nowy model w programie Visual Studio 2010, ten szablon jest używany przez domyślny, ale .tt, w których pliki nie są uwzględniane w projekcie. Jeśli chcesz dostosować szablon, należy dodać go do projektu.

Śledzenie własnym Generator jednostki (set)

Ten szablon generuje klas jednostek Self-Tracking i kontekstu, która pochodzi z obiektu ObjectContext. W aplikacji EF kontekst jest odpowiedzialny za śledzenie zmian w jednostkach. Jednak w scenariuszach N-warstwowa kontekst może nie być dostępne w warstwie, która modyfikuje jednostki. Jednostki własnym śledzeniem pomóc śledzić zmiany w dowolnej warstwy. Aby uzyskać więcej informacji, zobacz [jednostek Self-Tracking](#).

NOTE

WKLEJ szablon niezalecane

Nie zalecamy już przy użyciu szablonu Wklej kod w nowej aplikacji, jej nadal będzie dostępna do obsługi istniejących aplikacji. Odwiedź stronę [artykułu odłączone jednostki](#) innych opcji, firma Microsoft zaleca w scenariuszach N-warstwowej.

NOTE

Dostępna jest wersja 6.x nie EF, Wklej kod szablonu.

NOTE

Nie ma żadnych wersji programu Visual Studio 2013 szablonu Wklej kod.

Visual Studio 2012

Jeśli używasz programu Visual Studio 2012 będzie konieczne wybranie **Online** karcie podczas dodawania szablonu, aby go pobrać z galerii Visual Studio. Alternatywnie można zainstalować szablon bezpośrednio z galerii programu Visual Studio wcześniej. Ponieważ szablony są zawarte w Visual Studio 2010 w wersji w galerii można zainstalować tylko w programie Visual Studio 2012.

- [EF 5.x WKLEJ Generator dla języka C#](#)
- [EF 5.x WKLEJ Generator dla witryn sieci Web w języku C#](#)
- [Generator WKLEJ 5.x EF VB.NET](#)
- [EF 5.x WKLEJ Generator dla witryn sieci Web VB.NET](#)

Program Visual Studio 2010 **

Jeśli używasz programu Visual Studio 2010, ten szablon jest już zainstalowana.

Obiektów POCO Generator jednostki

Ten szablon generuje klas obiektów POCO i kontekstu, która pochodzi z obiektu ObjectContext

NOTE

Należy wziąć pod uwagę przy użyciu generatora DbContext

DbContext Generator jest teraz zalecane szablon Generowanie klas POCO w nowych aplikacji. DbContext Generator korzysta z nowego interfejsu API typu DbContext i może generować prostsze POCO klasy. Generator jednostki obiektów POCO jest nadal dostępne do obsługi istniejących aplikacji.

NOTE

Nie ma żadnych EF 5.x i 6.x EF wersję szablonu Wklej kod.

NOTE

Nie ma żadnych wersji programu Visual Studio 2013 szablonu POCO.

Program Visual Studio 2012 & programu Visual Studio 2010

Będzie konieczne wybranie **Online** karcie podczas dodawania szablonu, aby go pobrać z galerii Visual Studio. Alternatywnie można zainstalować szablonu bezpośrednio z galerii programu Visual Studio wcześniej.

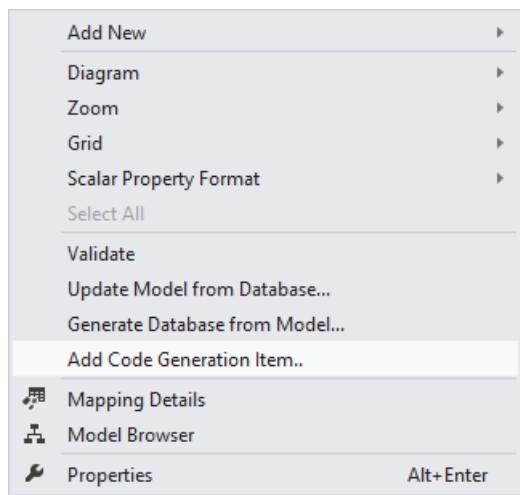
- [EF 4.x POCO Generator dla języka C#](#)
- [EF 4.x POCO Generator dla witryn sieci Web w języku C#](#)
- [EF 4.x POCO Generator VB.NET](#)
- [EF 4.x POCO Generator dla witryn sieci Web VB.NET](#)

Co to są szablony "Witryny sieci Web"

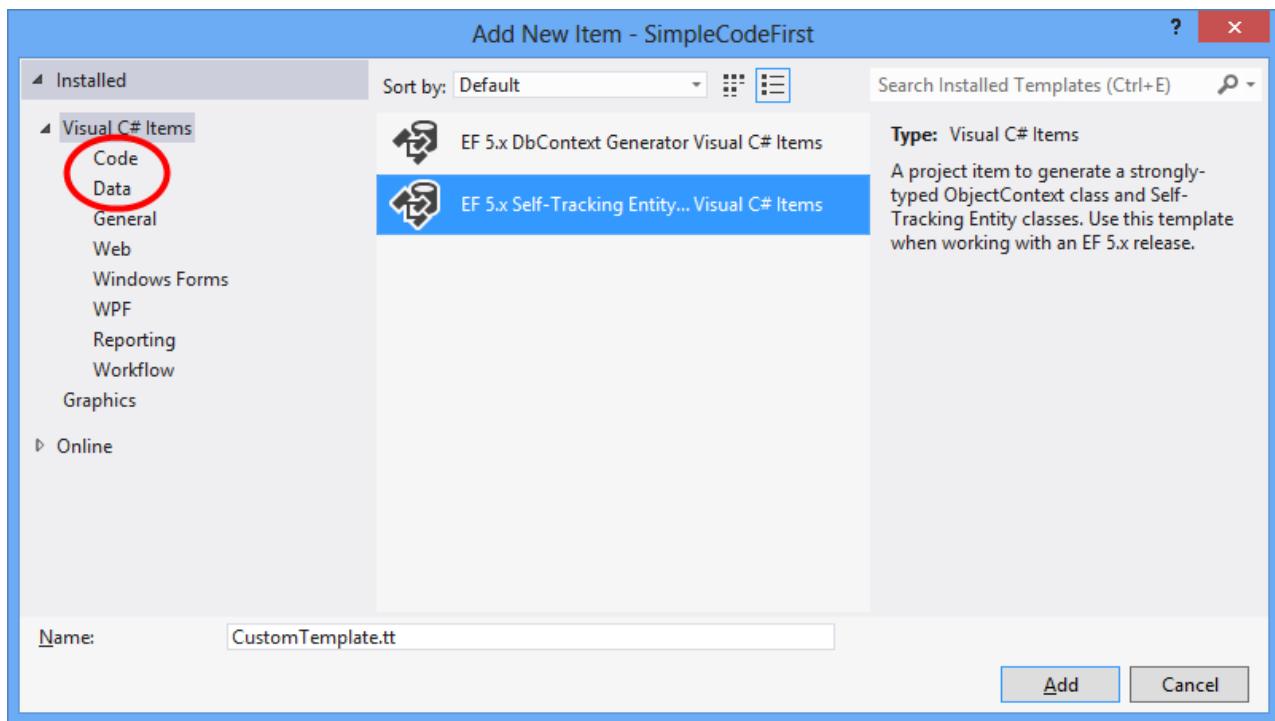
Szablony "Witryny sieci Web" (na przykład **EF 5.x Generator DbContext dla języka C# witryn sieci Web**) są przeznaczone do użytku w projektów witryny sieci Web utworzone za pomocą **pliku —> New -> witryny sieci Web...**. Są one różne od aplikacji sieci Web utworzone za pomocą **pliku —> New -> projektu...**, które używają szablonów standardowych. Firma Microsoft oferuje osobnymi szablonami, ponieważ system szablonu elementu w programie Visual Studio wymaga, aby je.

Przy użyciu szablonu

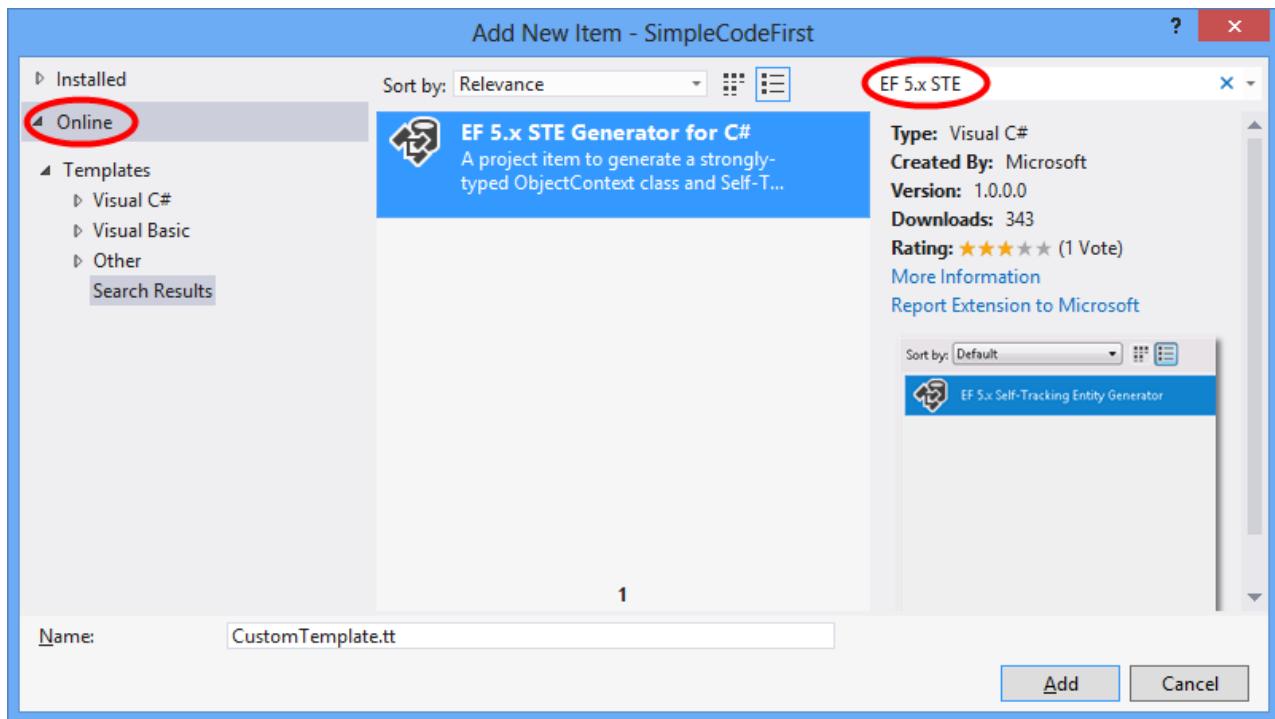
Aby rozpocząć korzystanie z szablonów generowania kodu, kliknij prawym przyciskiem myszy pusty miejscu na powierzchni projektowej, w Projektancie platformy EF i wybierz **Dodaj element generowanie kodu...**.



Jeśli masz już zainstalowaną szablonu chcesz użyć (lub został on uwzględniony w programie Visual Studio), a następnie będzie on dostępny w obszarze **kodu** lub **danych** sekcji w menu po lewej stronie.



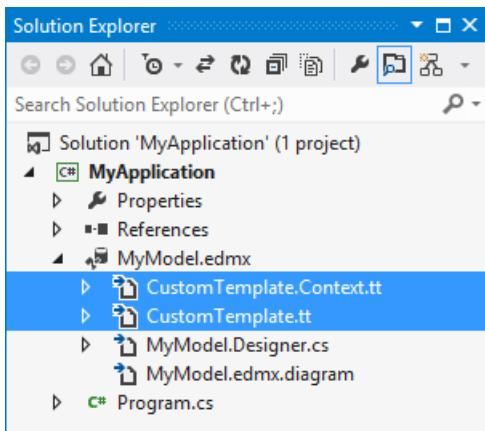
Jeśli nie masz jeszcze szablonu zainstalowany, wybierz opcję **Online** z menu po lewej stronie i wyszukaj odpowiedni szablon.



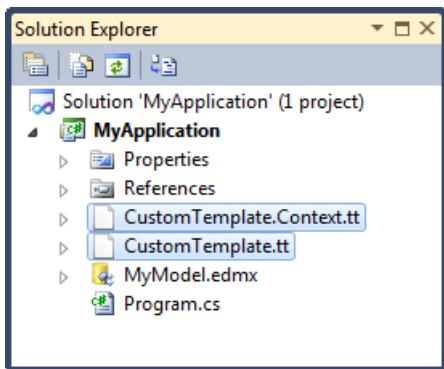
Jeśli używasz programu Visual Studio 2012, nowe pliki .tt będzie można zagnieździć w file.* edmx

NOTE

W przypadku modeli utworzonych w programie Visual Studio 2012, musisz usunąć szablonów używanych dla domyślnego generowania kodu w przeciwnym razie otrzymasz klasy zduplikowane oraz wygenerowany kontekst. Domyślne pliki są **<nazwę modelu>.tt** i **<nazwę modelu>.context.tt**.



Jeśli używasz programu Visual Studio 2010 pliki tt są dodawane bezpośrednio do swojego projektu.



Powracanie do obiektu ObjectContext w programie Entity Framework Designer

13.09.2018 • 2 minutes to read • [Edit Online](#)

Za pomocą poprzedniej wersji programu Entity Framework modelu utworzone za pomocą projektanta EF wygeneruje kontekstu, który pochodzi od obiektu ObjectContext i klas jednostek, które pochodzą EntityObject.

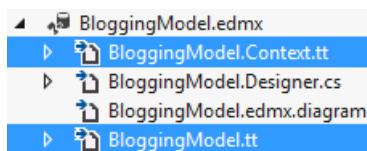
Począwszy od EF4.1 zalecane zamianą na szablon generowania kodu, który generuje wyprowadzanie z klas jednostek DbContext i POCO kontekstu.

W programie Visual Studio 2012 uzyskuje się kod DbContext generowane domyślnie dla wszystkich nowych modeli utworzone za pomocą projektanta EF. Istniejące modele będą w dalszym ciągu generowanie kodu na podstawie obiektu ObjectContext, chyba że zdecydujesz o przechodź do generatora kodu na podstawie typu DbContext.

Powracanie do generowania kodu obiektu ObjectContext

1. Wyłącz generowanie kodu typu DbContext

Generowanie klas pochodnych typu DbContext i POCO odbywa się przez dwa .TT — pliki w projekcie, po rozwinięciu pola pliku edmx w Eksploratorze rozwiązań zobaczą te pliki. Usuń oba te pliki z projektu.



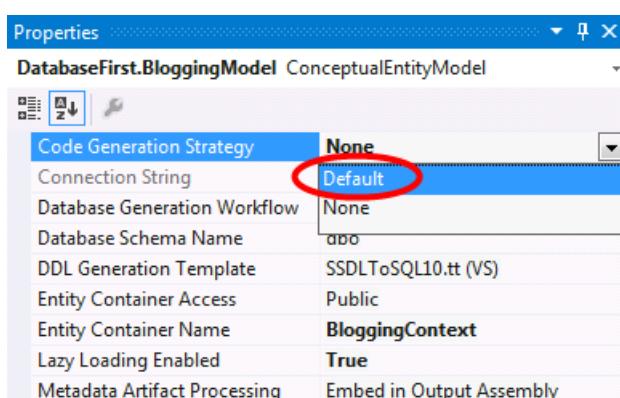
Jeśli używasz VB.NET będzie konieczne wybranie **Pokaż wszystkie pliki** przycisk, aby wyświetlić zagnieżdżone pliki.



2. Ponowne włączenie generowania kodu obiektu ObjectContext

Otwieranie modelu w Projektancie platformy EF, kliknij prawym przyciskiem myszy pustą sekcję projekt powierzchni i wybierz **właściwości**.

W oknie Zmień właściwości **strategia generowania kodu** z **Brak** do **domyślne**.



Specyfikacja CSDL

02.10.2018 • 109 minutes to read • [Edit Online](#)

Język definicji schematu koncepcyjnego (CSDL) to oparty na standardzie XML język, który opisuje jednostki, relacje i funkcje, które tworzą model koncepcyjny aplikacji opartych na danych. Ten model koncepcyjny może służyć przez Entity Framework lub usługi danych WCF. Metadane opisujące z CSDL jest używany przez program Entity Framework do mapowania jednostek i relacji, które są zdefiniowane w modelu koncepcyjnym ze źródłem danych. Aby uzyskać więcej informacji, zobacz [Specyfikacja SSDL](#) i [Specyfikacja MSL](#).

CSDL to implementacja programu Entity Framework modelu Entity Data Model.

W aplikacji Entity Framework metadanych modelu koncepcyjnego jest ładowany z pliku .csdl (napisanego w CSDL) do wystąpienia System.Data.Metadata.Edm.EdmItemCollection i są dostępne za pomocą metody Klasa System.Data.Metadata.Edm.MetadataWorkspace. Entity Framework używa modelu koncepcyjnego metadanych do translacji zapytania względem modelu koncepcyjnego poleceń specyficznymi dla źródła danych.

W Projektancie platformy EF przechowuje informacje o modelu koncepcyjnym w pliku edmx w czasie projektowania. W czasie komplikacji projektancie platformy EF używa informacji w pliku edmx można utworzyć pliku .csdl, które są wymagane przez Entity Framework w czasie wykonywania.

Wersje CSDL są zróżnicowane według przestrzeni nazw XML.

WERSJA CSDL	NAMESPACE XML
CSDL v1	http://schemas.microsoft.com/ado/2006/04/edm
CSDL v2	http://schemas.microsoft.com/ado/2008/09/edm
CSDL v3	http://schemas.microsoft.com/ado/2009/11/edm

Elementu Association (CSDL)

Skojarzenia element definiuje relację między dwoma typami encji. Skojarzenia należy określić typy jednostek, które są zaangażowane w relacji i możliwa liczba typów jednostek na każdym końcu relacji, który jest znany jako liczebności. Liczebność elementu end skojarzenia mogą mieć wartość równą jeden (1), zero lub jeden (od 0 do 1) lub wielu (*). Informacja ta jest określona w dwa elementy End podrzędnych.

Wystąpienia typu jednostki na jednym końcu asocjacji jest możliwy za pośrednictwem właściwości nawigacji lub kluczy obcych, jeśli są one widoczne na typ jednostki.

W aplikacji wystąpienia skojarzenia reprezentuje określone skojarzenia między wystąpieniami typów jednostek. Skojarzenie wystąpienia są logicznie pogrupowane w zestawie skojarzenia.

Skojarzenia element może mieć następujących elementów podrzędnych (w podanej kolejności):

- Dokumentacja (zero lub jeden element)
- Końcowy (elementy dokładnie 2)
- ReferentialConstraint (zero lub jeden element)
- Elementów adnotacji (zero lub więcej elementów)

Odpowiednie atrybuty

W poniższej tabeli opisano atrybuty, które mogą być stosowane do **skojarzenia** elementu.

NAZWA ATRYBUTU	JEST WYMAGANY	WARTOŚĆ
Nazwa	Tak	Nazwa skojarzenia.

NOTE

Dowolna liczba atrybutów adnotacji (niestandardowe atrybuty XML) można stosować do **skojarzenia** elementu. Jednak atrybutów niestandardowych, które nie mogą należeć do przestrzeni nazw XML, który jest zarezerwowany dla CSDL. W pełni kwalifikowanej nazwy dowolne dwa atrybuty niestandardowe nie może być taka sama.

Przykład

W poniższym przykładzie przedstawiono **skojarzenia** element, który definiuje **CustomerOrders** skojarzenie po klucze obce nie być narażone na **klienta** i **Kolejność** typów jednostek. **Liczebność** wartości dla każdej **zakończenia** skojarzenia wskazują tę liczbę **zamówienia** może być skojarzony z **klienta**, ale tylko jeden **klienta** może być skojarzony z **kolejności**. Ponadto **OnDelete** elementu wskazuje, że wszystkie **zamówienia** powiązanych z określonego **klienta** i zostały załadowane do obiektu ObjectContext zostaną usunięte. Jeśli **klienta** zostanie usunięty.

```
<Association Name="CustomerOrders">
    <End Type="ExampleModel.Customer" Role="Customer" Multiplicity="1" >
        <OnDelete Action="Cascade" />
    </End>
    <End Type="ExampleModel.Order" Role="Order" Multiplicity="*" />
</Association>
```

W poniższym przykładzie przedstawiono **skojarzenia** element, który definiuje **CustomerOrders** skojarzenie po klucze obce być narażone na **klienta** i **Kolejność** typów jednostek. Za pomocą kluczy obcych widoczne, relacji między jednostkami odbywa się za pomocą **ReferentialConstraint** elementu. Odpowiedni element **AssociationSetMapping** nie jest konieczne do mapowania tego skojarzenia ze źródłem danych.

```
<Association Name="CustomerOrders">
    <End Type="ExampleModel.Customer" Role="Customer" Multiplicity="1" >
        <OnDelete Action="Cascade" />
    </End>
    <End Type="ExampleModel.Order" Role="Order" Multiplicity="*" />
    <ReferentialConstraint>
        <Principal Role="Customer">
            <PropertyRef Name="Id" />
        </Principal>
        <Dependent Role="Order">
            <PropertyRef Name="CustomerId" />
        </Dependent>
    </ReferentialConstraint>
</Association>
```

Obiekt AssociationSet — Element (CSDL)

AssociationSet element języka definicji schematu koncepcyjnego (CSDL) to kontener logiczny dla wystąpień skojarzeń tego samego typu. Zestaw skojarzeń zawiera definicję dla grupowania wystąpień skojarzeń, dzięki czemu mogą być mapowane do źródła danych.

AssociationSet element może mieć następujących elementów podrzędnych (w podanej kolejności):

- Dokumentacja (zero lub jeden elementy dozwolone)
- Końcowy (dokładnie dwa elementy wymagane)

- Elementów adnotacji (zero lub więcej elementów dozwolone)

Skajarzenia atrybut określa typ powiązania, który zawiera zestaw skojarzeń. Zestawy jednostek, składających się kończy się zestawu skojarzeń są określane za pomocą podrzędej dokładnie dwóch **zakończenia** elementów.

Odpowiednie atrybuty

W poniższej tabeli opisano atrybuty, które mogą być stosowane do **AssociationSet** elementu.

NAZWA ATRYBUTU	JEST WYMAGANY	WARTOŚĆ
Nazwa	Tak	Nazwa zestawu jednostek. Wartość nazwa atrybut nie może być taka sama jak wartość skajarzenia atrybutu.
Skojarzenie	Tak	W pełni kwalifikowana nazwa skojarzenia, które zestawu skojarzeń zawiera wystąpienia. Skojarzenie musi być w tej samej przestrzeni nazw jako zestaw skojarzeń.

NOTE

Dowolna liczba atrybutów adnotacji (niestandardowe atrybuty XML) można stosować do **AssociationSet** elementu. Jednak atrybutów niestandardowych, które nie mogą należeć do przestrzeni nazw XML, który jest zarezerwowany dla CSDL. W pełni kwalifikowanej nazwy dowolne dwa atrybuty niestandardowe nie może być taka sama.

Przykład

W poniższym przykładzie przedstawiono **EntityContainer** elementu przy użyciu dwóch **AssociationSet** elementy:

```
<EntityContainer Name="BooksContainer" >
  <EntityType Name="Books" EntityType="BooksModel.Book" />
  <EntityType Name="Publishers" EntityType="BooksModel.Publisher" />
  <EntityType Name="Authors" EntityType="BooksModel.Author" />
  <AssociationSet Name="PublishedBy" Association="BooksModel.PublishedBy">
    <End Role="Book" EntitySet="Books" />
    <End Role="Publisher" EntitySet="Publishers" />
  </AssociationSet>
  <AssociationSet Name="WrittenBy" Association="BooksModel.WrittenBy">
    <End Role="Book" EntitySet="Books" />
    <End Role="Author" EntitySet="Authors" />
  </AssociationSet>
</EntityContainer>
```

Element CollectionType (CSDL)

CollectionType element język definicji schematu koncepcyjnego (CSDL) określa, że parametr funkcji lub funkcji zwracany typ jest kolekcją. **CollectionType** element może być elementem podrzędnym elementu parametru lub element ReturnType (funkcja). Typ kolekcji można określić za pomocą **typu** atrybutu lub jedną z następujących elementów podrzędnych:

- **Typ CollectionType**
- Element referenceType
- RowType
- TypeRef

NOTE

Model nie zostanie przeprowadzona Weryfikacja Jeśli typ kolekcji jest określony zarówno **typu** atrybut i nie zawiera elementu podrzędnego.

Odpowiednie atrybuty

W poniższej tabeli opisano atrybuty, które mogą być stosowane do **CollectionType** elementu. Należy pamiętać, że **DefaultValue**, **MaxLength**, **FixedLength**, **dokładności**, **skalowania**, **Unicode**, i **sortowania** atrybuty dotyczą tylko kolekcje **EDMSimpleTypes**.

NAZWA ATRYBUTU	JEST WYMAGANY	WARTOŚĆ
Typ	Nie	Typ kolekcji.
Dopuszcza wartości null	Nie	Wartość true , (wartość domyślna) lub False w zależności od tego, czy właściwość może mieć wartości null. [!NOTE]
> W CSDL v1 musi mieć właściwość typu złożonego <code>Nullable="False"</code> .		
defaultValue	Nie	Wartość domyślna właściwości.
Element maxLength	Nie	Maksymalna długość wartości właściwości.
Wartości	Nie	Wartość true , lub False w zależności od tego, czy przechowywana wartość właściwości jako ciąg znaków o stałej długości.
Precyzja	Nie	Dokładność wartości właściwości.
Skala	Nie	Skala wartości właściwości.
SRID	Nie	Identyfikator odwołania przestrzennego systemu. Prawidłowy tylko w przypadku właściwości typów przestrzennych. Aby uzyskać więcej informacji, zobacz SRID i SRID (SQL Server)
Unicode	Nie	Wartość true , lub False w zależności od tego, czy przechowywana wartość właściwości jako ciąg Unicode.
Sortowanie	Nie	Ciąg, który określa kolejność sortowania, które ma być używany w źródle danych.

NOTE

Dowolna liczba atrybutów adnotacji (niestandardowe atrybuty XML) można stosować do **CollectionType** elementu. Jednak atrybutów niestandardowych, które nie mogą należeć do przestrzeni nazw XML, który jest zarezerwowany dla CSDL. W pełni kwalifikowanej nazwy dowolne dwa atrybuty niestandardowe nie może być taka sama.

Przykład

W poniższym przykładzie pokazano funkcję definiowanych przez model, który używa **CollectionType** elementu, aby określić, że funkcja zwraca kolekcję **osoby** typów jednostek (jak określono za pomocą **ElementType** atrybutu).

```
<Function Name="LastNamesAfter">
    <Parameter Name="someString" Type="Edm.String"/>
    <ReturnType>
        <CollectionType ElementType="SchoolModel.Person"/>
    </ReturnType>
    <DefiningExpression>
        SELECT VALUE p
        FROM SchoolEntities.People AS p
        WHERE p.LastName >= someString
    </DefiningExpression>
</Function>
```

W poniższym przykładzie pokazano funkcji definiowanych przez model, który używa **CollectionType** elementu, aby określić, że funkcja zwraca Kolekcja wierszy (jak określono w **RowType** elementu).

```
<Function Name="LastNamesAfter">
    <Parameter Name="someString" Type="Edm.String" />
    <ReturnType>
        <CollectionType>
            <RowType>
                <Property Name="FirstName" Type="Edm.String" Nullable="false" />
                <Property Name="LastName" Type="Edm.String" Nullable="false" />
            </RowType>
        </CollectionType>
    </ReturnType>
    <DefiningExpression>
        SELECT VALUE ROW(p.FirstName, p.LastName)
        FROM SchoolEntities.People AS p
        WHERE p.LastName >= somestring
    </DefiningExpression>
</Function>
```

W poniższym przykładzie pokazano funkcji definiowanych przez model, który używa **CollectionType** elementu, aby określić, że funkcja przyjmuje jako parametr zbiór **działu** typów jednostek.

```
<Function Name="GetAvgBudget">
    <Parameter Name="Departments">
        <CollectionType>
            <TypeRef Type="SchoolModel.Department"/>
        </CollectionType>
    </Parameter>
    <ReturnType Type="Collection(Edm.Decimal)"/>
    <DefiningExpression>
        SELECT VALUE AVG(d.Budget) FROM Departments AS d
    </DefiningExpression>
</Function>
```

Element ComplexType (CSDL)

A **ComplexType** element definiuje strukturę danych składającą się z **EdmSimpleType** właściwości lub inne typy złożone. Typ złożony mogą być właściwość typu jednostki lub innego typu złożonego. Typ złożony jest podobny dla typu jednostki, w tym, że typ złożony definiuje dane. Jednakże istnieją niektóre podstawowe różnice między typy złożone i typy jednostek:

- Typy złożone nie mają tożsamości (lub kluczy) i dlatego nie mogą istnieć niezależnie. Typy złożone może istnieć tylko jako właściwości typów jednostek lub inne typy złożone.
- Typy złożone nie mogą uczestniczyć w skojarzeniach. Aby end elementu association może być to typ złożony, i w związku z tym nie można zdefiniować właściwości nawigacji dla typów złożonych.
- Właściwość typu złożonego nie może mieć wartość null, jeśli właściwości skalarne typu złożonego każdego można ustawić na wartość null.

A **ComplexType** element może mieć następujących elementów podrzędnych (w podanej kolejności):

- Dokumentacja (zero lub jeden element)
- Właściwości (zero lub więcej elementów)
- Elementów adnotacji (zero lub więcej elementów)

W poniższej tabeli opisano atrybuty, które mogą być stosowane do **ComplexType** elementu.

NAZWA ATRYBUTU	JEST WYMAGANY	WARTOŚĆ
Nazwa	Tak	Nazwa typu złożonego. Nazwa typu złożonego nie może być taka sama jak nazwa innego typu złożonego, typ jednostki lub skojarzenia, który znajduje się w zakresie modelu.
BaseType	Nie	Nazwa innego typu złożonego, który jest typ bazowy typ złożony, który jest definiowany. [!NOTE]
> Ten atrybut nie ma zastosowania w wersji 1 CSDL. Dziedziczenia dla typów złożonych nie jest obsługiwane w tej wersji.		
Abstrakcyjny	Nie	Wartość true , lub False (wartość domyślna) w zależności od tego, czy typ złożony jest typem abstrakcyjnym. [!NOTE]
> Ten atrybut nie ma zastosowania w wersji 1 CSDL. Typy złożone w tej wersji nie może być typów abstrakcyjnych.		

NOTE

Dowolna liczba atrybutów adnotacji (niestandardowe atrybuty XML) można stosować do **ComplexType** elementu. Jednak atrybutów niestandardowych, które nie mogą należeć do przestrzeni nazw XML, który jest zarezerwowany dla CSDL. W pełni kwalifikowanej nazwy dowolne dwa atrybuty niestandardowe nie może być taka sama.

Przykład

Poniższy przykład przedstawia typ złożony, **adres**, za pomocą **EdmSimpleType** właściwości **adres**, **Miasto**, **StanLubProwincja**, **kraju**, i **KodPocztowy**.

```
<ComplexType Name="Address" >
  <Property Type="String" Name="StreetAddress" Nullable="false" />
  <Property Type="String" Name="City" Nullable="false" />
  <Property Type="String" Name="StateOrProvince" Nullable="false" />
  <Property Type="String" Name="Country" Nullable="false" />
  <Property Type="String" Name="PostalCode" Nullable="false" />
</ComplexType>
```

Aby zdefiniować typ złożony **adres** (p. wyżej) jako właściwość typu jednostki, należy zadeklarować typ właściwości w definicji typu jednostki. W poniższym przykładzie przedstawiono **adres** właściwość typu złożonego typu encji (**wydawcy**):

```
<EntityType Name="Publisher">
  <Key>
    <PropertyRef Name="Id" />
  </Key>
  <Property Type="Int32" Name="Id" Nullable="false" />
  <Property Type="String" Name="Name" Nullable="false" />
  <Property Type="BooksModel.Address" Name="Address" Nullable="false" />
  <NavigationProperty Name="Books" Relationship="BooksModel.PublishedBy"
    FromRole="Publisher" ToRole="Book" />
</EntityType>
```

Element DefiningExpression (CSDL)

DefiningExpression element język definicji schematu koncepcyjnego (CSDL) zawiera wyrażenie SQL jednostki, które definiuje funkcję w modelu koncepcyjnym.

NOTE

Do celów sprawdzania poprawności **DefiningExpression** element może zawierać dowolną zawartość. Jednak Entity Framework spowoduje zgłoszenie wyjątku w czasie wykonywania przypadku **DefiningExpression** element nie zawiera nieprawidłowej SQL jednostki.

Odpowiednie atrybuty

Dowolna liczba atrybutów adnotacji (niestandardowe atrybuty XML) można stosować do **DefiningExpression** elementu. Jednak atrybutów niestandardowych, które nie mogą należeć do przestrzeni nazw XML, który jest zarezerwowany dla CSDL. W pełni kwalifikowanej nazwy dowolne dwa atrybuty niestandardowe nie może być taka sama.

Przykład

W poniższym przykładzie użyto **DefiningExpression** elementu, aby zdefiniować funkcję, która zwraca liczbę lat, ponieważ została opublikowana książki. Zawartość **DefiningExpression** elementu są zapisywane w języku SQL jednostki.

```
<Function Name="GetYearsInPrint" ReturnType="Edm.Int32" >
  <Parameter Name="book" Type="BooksModel.Book" />
  <DefiningExpression>
    Year(CurrentDateTime()) - Year(cast(book.PublishedDate as DateTime))
  </DefiningExpression>
</Function>
```

Element zależne (CSDL)

Zależne element język definicji schematu koncepcyjnego (CSDL) jest elementem podrzędnym elementu **ReferentialConstraint** i definiuje zależne koniec ograniczenia referencyjnego. A **ReferentialConstraint** element definiuje funkcjonalność, która jest podobna do ograniczenia integralności referencyjnej w relacyjnej bazie danych. W ten sam sposób, w kolumnie (lub kolumny) z tabeli bazy danych odwołać się do klucza podstawowego z innej tabeli Właściwość (lub właściwości), typu jednostki odwoływać się do klucza jednostki innego typu jednostki. Typ jednostki, do którego istnieje odwołanie jest wywoływana *jednostki zakończenia* ograniczenia. Typ jednostki, który odwołuje się do zakończenia podmiotu zabezpieczeń jest wywoływana *zależne zakończenia* ograniczenia. **PropertyRef** elementy są używane do określenia, których kluczy odwoływać się do zakończenia podmiotu zabezpieczeń.

Zależne element może mieć następujących elementów podrzędnych (w podanej kolejności):

- **PropertyRef** (jeden lub więcej elementów)
- Elementów adnotacji (zero lub więcej elementów)

Odpowiednie atrybuty

W poniższej tabeli opisano atrybuty, które mogą być stosowane do **zależne** elementu.

NAZWA ATRYBUTU	JEST WYMAGANY	WARTOŚĆ
Rola	Tak	Nazwa typu jednostki, na końcu zależne skojarzenia.

NOTE

Dowolna liczba atrybutów adnotacji (niestandardowe atrybuty XML) można stosować do **zależne** elementu. Jednak atrybutów niestandardowych, które nie mogą należeć do przestrzeni nazw XML, który jest zarezerwowany dla CSDL. W pełni kwalifikowanej nazwy dowolne dwa atrybuty niestandardowe nie może być taka sama.

Przykład

W poniższym przykładzie przedstawiono **ReferentialConstraint** używany jako część definicji elementu **PublishedBy** skojarzenia. **PublisherId** właściwość **książki** typu jednostki tworzy zależne koniec ograniczenia referencyjnego.

```
<Association Name="PublishedBy">
  <End Type="BooksModel.Book" Role="Book" Multiplicity="*" >
  </End>
  <End Type="BooksModel.Publisher" Role="Publisher" Multiplicity="1" />
  <ReferentialConstraint>
    <Principal Role="Publisher">
      <PropertyRef Name="Id" />
    </Principal>
    <Dependent Role="Book">
      <PropertyRef Name="PublisherId" />
    </Dependent>
  </ReferentialConstraint>
</Association>
```

Element documentation (CSDL)

Dokumentacji element język definicji schematu koncepcyjnego (CSDL) może służyć do Podaj informacje dotyczące obiektu, który jest zdefiniowany w elemencie rodzica. W pliku edmx gdy **dokumentacji** element jest elementem podrzędnym elementu, który pojawia się jako obiekt na powierzchni projektowej w Projektancie

platformy EF (np. jednostek, skojarzeń lub właściwości), zawartość **dokumentacji** element będzie wyświetlane jako w programie Visual Studio **właściwości** okna dla obiektu.

Dokumentacji element może mieć następujących elementów podrzędnych (w podanej kolejności):

- **Podsumowanie:** Krótki opis elementu nadrzędnego. (element zero lub jeden)
- **LongDescription:** rozbudowany opis elementu nadrzędnego. (element zero lub jeden)
- Elementy adnotacji. (elementy zero lub więcej)

Odpowiednie atrybuty

Dowolna liczba atrybutów adnotacji (niestandardowe atrybuty XML) można stosować do **dokumentacji** elementu. Jednak atrybutów niestandardowych, które nie mogą należeć do przestrzeni nazw XML, który jest zarezerwowany dla CSDL. W pełni kwalifikowanej nazwy dowolne dwa atrybuty niestandardowe nie może być taka sama.

Przykład

W poniższym przykładzie przedstawiono **dokumentacji** element jako element podrzędny elementu EntityType. Gdyby poniższy fragment kodu w CSDL edmx zawartości pliku, zawartość **Podsumowanie** i **LongDescription** elementy pojawią się w programie Visual Studio **właściwości** okna po kliknięciu `Customer` typu jednostki.

```
<EntityType Name="Customer">
  <Documentation>
    <Summary>Summary here.</Summary>
    <LongDescription>Long description here.</LongDescription>
  </Documentation>
  <Key>
    <PropertyRef Name="CustomerId" />
  </Key>
  <Property Type="Int32" Name="CustomerId" Nullable="false" />
  <Property Type="String" Name="Name" Nullable="false" />
</EntityType>
```

Element end (CSDL)

Zakończenia element język definicji schematu koncepcyjnego (CSDL) może być elementem podrzędnym elementu Association lub elementu AssociationSet. W każdym przypadku rolę **zakończenia** różni się elementu i atrybuty stosowane są różne.

Końcowy Element jako element podrzędny elementu Association

Zakończenia — element (jako element podrzędny elementu **skojarzenia** elementu) identyfikuje typ jednostki na jednym końcu asocjacji i liczbę wystąpień typu jednostki, które może znajdować się na końcu tego skojarzenia. Skojarzenia są zdefiniowane jako część skojarzenia; Skojarzenie musi mieć dokładnie dwa punkty końcowe skojarzenia. Wystąpienia typu jednostki na jednym końcu asocjacji jest możliwy za pośrednictwem właściwości nawigacji lub kluczy obcych, jeśli są one widoczne na typ jednostki.

Zakończenia element może mieć następujących elementów podrzędnych (w podanej kolejności):

- Dokumentacja (zero lub jeden element)
- OnDelete (zero lub jeden element)
- Elementów adnotacji (zero lub więcej elementów)

Odpowiednie atrybuty

W poniższej tabeli opisano atrybuty, które mogą być stosowane do **zakończenia** elementu, gdy jest elementem podrzędnym elementu **skojarzenia** elementu.

NAZWA ATRYBUTU	JEST WYMAGANY	WARTOŚĆ
Typ	Tak	Nazwa typu jednostki, na jednym końcu skojarzenia.
Rola	Nie	Nazwa dla elementu end skojarzenia. Jeśli nie podano żadnej nazwy, nazwa typu jednostki na punkt końcowy będzie używany.
Liczliwość	Tak	<p>1, od 0 do 1, lub * w zależności od liczby wystąpień typów jednostek, które mogą być na końcu skojarzenia. 1 oznacza tego wystąpienia typu dokładnie jedną jednostkę istnieje na końcu skojarzenia. od 0 do 1 oznacza, że na końcu skojarzenia zera lub jednego wystąpienia typu jednostki. * oznacza, że wartość zero, jeden lub więcej wystąpień typu jednostki na końcu skojarzenia.</p>

NOTE

Dowolna liczba atrybutów adnotacji (niestandardowe atrybuty XML) można stosować do **zakończenia** elementu. Jednak atrybutów niestandardowych, które nie mogą należeć do przestrzeni nazw XML, który jest zarezerwowany dla CSDL. W pełni kwalifikowanej nazwy dowolne dwa atrybuty niestandardowe nie może być taka sama.

Przykład

W poniższym przykładzie przedstawiono **skojarzenia** element, który definiuje **CustomerOrders** skojarzenia. **Liczliwość** wartości dla każdej **zakończenia** skojarzenia wskazują tę liczbę **zamówienia** może być skojarzony z **klienta**, ale tylko jeden **klienta** może być skojarzony z **kolejności**. Ponadto **OnDelete** elementu wskazuje, że wszystkie **zamówienia** powiązanych z określonego **klienta** i które zostały załadowane do obiektu ObjectContext będzie. Jeśli usunięto **klienta** zostanie usunięty.

```
<Association Name="CustomerOrders">
    <End Type="ExampleModel.Customer" Role="Customer" Multiplicity="1" />
    <End Type="ExampleModel.Order" Role="Order" Multiplicity="*" />
        <OnDelete Action="Cascade" />
    </End>
</Association>
```

Końcowy Element jako element podrzędny elementu AssociationSet

Zakończenia element określa jeden z punktów końcowych zestaw skojarzeń. **AssociationSet** musi zawierać dwa **zakończenia** elementów. Informacje zawarte w **zakończenia** element jest używany w mapowaniu skojarzenia Ustaw ze źródłem danych.

Zakończenia element może mieć następujących elementów podrzędnych (w podanej kolejności):

- Dokumentacja (zero lub jeden element)
- Elementów adnotacji (zero lub więcej elementów)

NOTE

Adnotacja elementów musi występować po wszystkich innych elementów podrzędnych. Elementów adnotacji są dozwolone tylko w wersji 2 CSDL i nowszych wersjach.

Odpowiednie atrybuty

W poniższej tabeli opisano atrybuty, które mogą być stosowane do **zakończenia** elementu, gdy jest elementem podrzędnym elementu **AssociationSet** elementu.

NAZWA ATRYBUTU	JEST WYMAGANY	WARTOŚĆ
Obiekt EntitySet	Tak	Nazwa EntitySet element, który definiuje jeden z punktów końcowych nadzawanego AssociationSet elementu. EntitySet element musi być zdefiniowany w tym samym kontenerze jednostki jako element nadzawny AssociationSet elementu.
Rola	Nie	Nazwa skojarzenia zestawu zakończenia. Jeśli rola atrybut nie jest używany, nazwę punktu końcowego zestawu skojarzenia będzie nazwa zestawu jednostek.

NOTE

Dowolna liczba atrybutów adnotacji (niestandardowe atrybuty XML) można stosować do **zakończenia** elementu. Jednak atrybutów niestandardowych, które nie mogą należeć do przestrzeni nazw XML, który jest zarezerwowany dla CSDL. W pełni kwalifikowanej nazwy dowolne dwa atrybuty niestandardowe nie może być taka sama.

Przykład

W poniższym przykładzie przedstawiono **EntityContainer** elementu przy użyciu dwóch **AssociationSet** przy użyciu dwóch elementów **zakończenia** elementy:

```
<EntityContainer Name="BooksContainer" >
  <EntitySet Name="Books" EntityType="BooksModel.Book" />
  <EntitySet Name="Publishers" EntityType="BooksModel.Publisher" />
  <EntitySet Name="Authors" EntityType="BooksModel.Author" />
  <AssociationSet Name="PublishedBy" Association="BooksModel.PublishedBy">
    <End Role="Book" EntitySet="Books" />
    <End Role="Publisher" EntitySet="Publishers" />
  </AssociationSet>
  <AssociationSet Name="WrittenBy" Association="BooksModel.WrittenBy">
    <End Role="Book" EntitySet="Books" />
    <End Role="Author" EntitySet="Authors" />
  </AssociationSet>
</EntityContainer>
```

Element EntityContainer (CSL)

EntityContainer element języka definicji schematu koncepcyjnego (CSL) jest kontenerem logicznym, zestawów encji, zestawów skojarzeń i Importy — funkcja. Kontener jednostek modelu koncepcyjnego mapuje do kontenera magazynu modelu jednostki za pośrednictwem elementu w elemencie EntityContainerMapping. Kontener jednostek modelu magazynu w tym artykule opisano struktury bazy danych: zestawy jednostek opisują tabel, zestawów skojarzeń opisano ograniczenia klucza obcego i Importy funkcji opisano procedury składowane w bazie

danych.

EntityContainer element może mieć zero lub jeden elementów dokumentacji. Jeśli **dokumentacji** element jest obecny, musi poprzedzać wszystkie **EntitySet**, **AssociationSet**, i **FunctionImport** elementów.

EntityContainer element może mieć zero lub więcej z następujących elementów podrzędnych (w podanej kolejności):

- Obiekt EntitySet
- Obiekt AssociationSet
- Element FunctionImport
- Elementów adnotacji

Możesz rozszerzyć **EntityContainer** element, aby uwzględnić zawartość innego **EntityContainer** który mieści się w tej samej przestrzeni nazw. Aby uwzględnić zawartość innego **EntityContainer**, w odwołujących się **obiektu EntityContainer** elementu, ustaw wartość **rozszerza** atrybutu Nazwa **Obiekt EntityContainer** element, który chcesz dołączyć. Wszystkie elementy podrzędne elementu dołączonej **EntityContainer** elementu będą traktowane jako elementy podrzędne odwołujących się **EntityContainer** elementu.

Odpowiednie atrybuty

W poniższej tabeli opisano atrybuty, które mogą być stosowane do **Using** elementu.

NAZWA ATRYBUTU	JEST WYMAGANY	WARTOŚĆ
Nazwa	Tak	Nazwa kontenera jednostek.
Rozszerza	Nie	Nazwa innego kontenera jednostek w ramach tej samej przestrzeni nazw.

NOTE

Dowolna liczba atrybutów adnotacji (niestandardowe atrybuty XML) można stosować do **EntityContainer** elementu. Jednak atrybutów niestandardowych, które nie mogą należeć do przestrzeni nazw XML, który jest zarezerwowany dla CSDL. W pełni kwalifikowanej nazwy dowolne dwa atrybuty niestandardowe nie może być taka sama.

Przykład

W poniższym przykładzie przedstawiono **EntityContainer** element, który definiuje trzy zestawy encji i zestawy skojarzeń dwa.

```
<EntityContainer Name="BooksContainer" >
  <EntitySet Name="Books" EntityType="BooksModel.Book" />
  <EntitySet Name="Publishers" EntityType="BooksModel.Publisher" />
  <EntitySet Name="Authors" EntityType="BooksModel.Author" />
  <AssociationSet Name="PublishedBy" Association="BooksModel.PublishedBy">
    <End Role="Book" EntitySet="Books" />
    <End Role="Publisher" EntitySet="Publishers" />
  </AssociationSet>
  <AssociationSet Name="WrittenBy" Association="BooksModel.WrittenBy">
    <End Role="Book" EntitySet="Books" />
    <End Role="Author" EntitySet="Authors" />
  </AssociationSet>
</EntityContainer>
```

Element EntitySet (CSDL)

EntitySet element język definicji schematu koncepcyjnego to kontener logiczny dla wystąpień tego typu jednostki

i wystąpień dowolnego typu, który jest tworzony na podstawie tego typu jednostki. Relacja między typem encji i zestawem jednostek jest analogiczne do relacji wiersz tabeli w relacyjnej bazie danych. Np. wiersz typu jednostki definiuje zestaw powiązanych danych i jak tabela, zestaw jednostek zawiera wystąpienia tej definicji. Zestaw jednostek zapewnia konstrukcję grupowanie wystąpień typów jednostek, dzięki czemu mogą być mapowane do pokrewne struktury danych w źródle danych.

Może być określona więcej niż jeden zestaw jednostek dla typu określonego obiektu.

NOTE

W Projektancie platformy EF nie obsługuje modelami koncepcyjnymi, które zawierają wiele zestawów jednostek według typu.

EntitySet element może mieć następujących elementów podrzędnych (w podanej kolejności):

- Element documentation (zero lub jeden elementy dozwolone)
- Elementów adnotacji (zero lub więcej elementów dozwolone)

Odpowiednie atrybuty

W poniższej tabeli opisano atrybuty, które mogą być stosowane do **EntitySet** elementu.

NAZWA ATRYBUTU	JEST WYMAGANY	WARTOŚĆ
Nazwa	Tak	Nazwa zestawu jednostek.
Typ entityType	Tak	W pełni kwalifikowana nazwa typu jednostki, dla której zestaw jednostek zawiera wystąpienia.

NOTE

Dowolna liczba atrybutów adnotacji (niestandardowe atrybuty XML) można stosować do **EntitySet** elementu. Jednak atrybutów niestandardowych, które nie mogą należeć do przestrzeni nazw XML, który jest zarezerwowany dla CSDL. W pełni kwalifikowanej nazwy dowolne dwa atrybuty niestandardowe nie może być taka sama.

Przykład

W poniższym przykładzie przedstawiono **EntityContainer** element z trzech **EntitySet** elementy:

```
<EntityContainer Name="BooksContainer" >
  <EntitySet Name="Books" EntityType="BooksModel.Book" />
  <EntitySet Name="Publishers" EntityType="BooksModel.Publisher" />
  <EntitySet Name="Authors" EntityType="BooksModel.Author" />
  <AssociationSet Name="PublishedBy" Association="BooksModel.PublishedBy">
    <End Role="Book" EntitySet="Books" />
    <End Role="Publisher" EntitySet="Publishers" />
  </AssociationSet>
  <AssociationSet Name="WrittenBy" Association="BooksModel.WrittenBy">
    <End Role="Book" EntitySet="Books" />
    <End Role="Author" EntitySet="Authors" />
  </AssociationSet>
</EntityContainer>
```

Istnieje możliwość definiowania wielu zestawów jednostek według typu (MEST). W poniższym przykładzie zdefiniowano kontener jednostek z dwoma zestawami jednostki dla **książki** typu jednostki:

```

<EntityContainer Name="BooksContainer" >
  <EntityType Name="Books" EntityType="BooksModel.Book" />
  <EntityType Name="FictionBooks" EntityType="BooksModel.Book" />
  <EntityType Name="Publishers" EntityType="BooksModel.Publisher" />
  <EntityType Name="Authors" EntityType="BooksModel.Author" />
  <AssociationSet Name="PublishedBy" Association="BooksModel.PublishedBy">
    <End Role="Book" EntitySet="Books" />
    <End Role="Publisher" EntitySet="Publishers" />
  </AssociationSet>
  <AssociationSet Name="BookAuthor" Association="BooksModel.BookAuthor">
    <End Role="Book" EntitySet="Books" />
    <End Role="Author" EntitySet="Authors" />
  </AssociationSet>
</EntityContainer>

```

Element EntityType (CSDL)

EntityType element reprezentuje strukturę koncepcji najwyższego poziomu, takich jak klient lub kolejność, w modelu koncepcyjnym. Typ jednostki jest szablonem dla wystąpień typów jednostek w aplikacji. Każdy szablon zawiera następujące informacje:

- Unikatowa nazwa. (Wymagane).
- Klucz jednostki, który jest definiowany przez jedną lub więcej właściwości. (Wymagane).
- Właściwości zawierające dane. (Opcjonalnie).
- Właściwości nawigacji, które umożliwiają nawigację z jednym końcem asocjacji na drugiej stronie. (Opcjonalnie).

W aplikacji wystąpienia typu jednostki reprezentują określonego obiektu (na przykład konkretnego klienta lub zamówienia). Każde wystąpienie typu jednostki musi mieć unikatowy klucz w ramach zestawu jednostek.

Dwa wystąpienia typu jednostki są traktowane jako równe tylko wtedy, gdy są one tego samego typu i wartości kluczy podmiotu są takie same.

EntityType element może mieć następujących elementów podrzędnych (w podanej kolejności):

- Dokumentacja (zero lub jeden element)
- Klucz (zero lub jeden element)
- Właściwości (zero lub więcej elementów)
- Element NavigationProperty (zero lub więcej elementów)
- Elementów adnotacji (zero lub więcej elementów)

Odpowiednie atrybuty

W poniższej tabeli opisano atrybuty, które mogą być stosowane do **EntityType** elementu.

NAZWA ATRYBUTU	JEST WYMAGANY	WARTOŚĆ
Nazwa	Tak	Nazwa typu jednostki.
BaseType	Nie	Nazwa innego typu jednostki, który jest typem podstawowym typu jednostki, który jest definiowany.
Abstrakcyjny	Nie	Wartość true , lub False , w zależności od tego, czy typ obiektu jest typem abstrakcyjnym.

NAZWA ATRYBUTU	JEST WYMAGANY	WARTOŚĆ
OpenType	Nie	Wartość true, lub False w zależności od tego, czy typ jednostki jest typem jednostki open. [!NOTE]
> OpenType atrybut ma zastosowanie tylko do typów jednostek, które są zdefiniowane w modelach koncepcyjnych, które są używane z architektury ADO.NET Data Services.		

NOTE

Dowolna liczba atrybutów adnotacji (niestandardowe atrybuty XML) można stosować do **EntityType** elementu. Jednak atrybutów niestandardowych, które nie mogą należeć do przestrzeni nazw XML, który jest zarezerwowany dla CSDL. W pełni kwalifikowanej nazwy dowolne dwa atrybuty niestandardowe nie może być taka sama.

Przykład

W poniższym przykładzie przedstawiono **EntityType** element z trzech **właściwość** elementów i dwóch **element NavigationProperty** elementy:

```
<EntityType Name="Book">
  <Key>
    <PropertyRef Name="ISBN" />
  </Key>
  <Property Type="String" Name="ISBN" Nullable="false" />
  <Property Type="String" Name="Title" Nullable="false" />
  <Property Type="Decimal" Name="Revision" Nullable="false" Precision="29" Scale="29" />
  <NavigationProperty Name="Publisher" Relationship="BooksModel.PublishedBy"
    FromRole="Book" ToRole="Publisher" />
  <NavigationProperty Name="Authors" Relationship="BooksModel.WrittenBy"
    FromRole="Book" ToRole="Author" />
</EntityType>
```

Element EnumType (CSDL)

EnumType element reprezentuje Typ wyliczany.

EnumType element może mieć następujących elementów podrzędnych (w podanej kolejności):

- Dokumentacja (zero lub jeden element)
- Element członkowski (zero lub więcej elementów)
- Elementów adnotacji (zero lub więcej elementów)

Odpowiednie atrybuty

W poniższej tabeli opisano atrybuty, które mogą być stosowane do **EnumType** elementu.

NAZWA ATRYBUTU	JEST WYMAGANY	WARTOŚĆ
Nazwa	Tak	Nazwa typu jednostki.

NAZWA ATRYBUTU	JEST WYMAGANY	WARTOŚĆ
IsFlags	Nie	Wartość true , lub False , w zależności od tego, czy typ wyliczeniowy może służyć jako zestaw flag. Wartość domyślna to wartość False..
UnderlyingType	Nie	Edm.Byte , Edm.Int16 , typem Edm.Int32 , Edm.Int64 lub Edm.SByte ogranicza zakres wartości tego typu. Domyślny typ podstawowy wyliczenia elementów to typem Edm.Int32..

NOTE

Dowolna liczba atrybutów adnotacji (niestandardowe atrybuty XML) można stosować do **EnumType** elementu. Jednak atrybutów niestandardowych, które nie mogą należeć do przestrzeni nazw XML, który jest zarezerwowany dla CSDL. W pełni kwalifikowanej nazwy dowolne dwa atrybuty niestandardowe nie może być taka sama.

Przykład

W poniższym przykładzie przedstawiono **EnumType** element z trzech **elementu członkowskiego** elementy:

```
<EnumType Name="Color" IsFlags="false" UnderlyingTyp="Edm.Byte">
  <Member Name="Red" />
  <Member Name="Green" />
  <Member Name="Blue" />
</EntityType>
```

Function — Element (CSDL)

Funkcja element język definicji schematu koncepcyjnego (CSDL) jest używany do definiowania lub deklarowania funkcji w modelu koncepcyjnym. Funkcja jest zdefiniowana za pomocą elementu **DefiningExpression**.

A **funkcja** element może mieć następujących elementów podrzędnych (w podanej kolejności):

- Dokumentacja (zero lub jeden element)
- Parametr (zero lub więcej elementów)
- DefiningExpression (zero lub jeden element)
- ReturnType (funkcja) (zero lub jeden element)
- Elementów adnotacji (zero lub więcej elementów)

Zwracana typ dla funkcji, należy określić albo **ReturnType** — element (funkcja) lub **ReturnType** atrybutu (patrz poniżej), ale nie oba. Możliwe zwarcane typy są wszelkie **EdmSimpleType**, jednostki typu, typu złożonego, typ wiersza lub typu referencyjnego (lub zbiór jednego z następujących typów).

Odpowiednie atrybuty

W poniższej tabeli opisano atrybuty, które mogą być stosowane do **funkcja** elementu.

NAZWA ATRYBUTU	JEST WYMAGANY	WARTOŚĆ
Nazwa	Tak	Nazwa funkcji.
ReturnType	Nie	Typ zwracany przez funkcję.

NOTE

Dowolna liczba atrybutów adnotacji (niestandardowe atrybuty XML) można stosować do **funkcja** elementu. Jednak atrybutów niestandardowych, które nie mogą należeć do przestrzeni nazw XML, który jest zarezerwowany dla CSDL. W pełni kwalifikowanej nazwy dowolne dwa atrybuty niestandardowe nie może być taka sama.

Przykład

W poniższym przykładzie użyto **funkcja** elementu, aby zdefiniować funkcję, która zwraca liczbę lat, ponieważ została zatrudniony pod kierunkiem instruktora.

```
<Function Name="YearsSince" ReturnType="Edm.Int32">
    <Parameter Name="date" Type="Edm.DateTime" />
    <DefiningExpression>
        Year(CurrentDateTime()) - Year(date)
    </DefiningExpression>
</Function>
```

Element FunctionImport (CSDL)

FunctionImport element język definicji schematu koncepcyjnego (CSDL) reprezentuje funkcję, która jest zdefiniowana w źródle danych, ale dostępne obiekty za pośrednictwem modelu koncepcyjnego. Na przykład elementem funkcji w modelu magazynu może służyć do reprezentowania procedur składowaną w bazie danych.

A **FunctionImport** elementu w modelu koncepcyjnym reprezentuje odpowiednich funkcji w aplikacji Entity Framework i jest mapowany do funkcji modelu magazynu przy użyciu elementu **FunctionImportMapping**. Gdy funkcja jest wywoływana w aplikacji, odpowiednie procedury składowanej jest wykonywany w bazie danych.

FunctionImport element może mieć następujących elementów podrzędnych (w podanej kolejności):

- Dokumentacja (zero lub jeden elementy dozwolone)
- Parametr (zero lub więcej elementów dozwolone)
- Elementów adnotacji (zero lub więcej elementów dozwolone)
- ReturnType (FunctionImport) (zero lub więcej elementów dozwolone)

Jeden **parametru** element powinien być zdefiniowany dla każdego parametru, który akceptuje funkcji.

Zwracana typ dla funkcji, należy określić albo **ReturnType** elementu (element **FunctionImport**) lub **ReturnType** atrybutu (patrz poniżej), ale nie oba. Wartość zwracany typ musi być kolekcją **EdmSimpleType**, **EntityType** lub **ComplexType**.

Odpowiednie atrybuty

W poniższej tabeli opisano atrybuty, które mogą być stosowane do **FunctionImport** elementu.

NAZWA ATRYBUTU	JEST WYMAGANY	WARTOŚĆ
Nazwa	Tak	Nazwa funkcji zimportowane.
ReturnType	Nie	Typ, który zwraca funkcja. Nie należy używać tego atrybutu, jeśli funkcja nie zwraca wartości. W przeciwnym razie wartość musi być kolekcją ComplexType , EntityType lub EDMSimpleType .

NAZWA ATRYBUTU	JEST WYMAGANY	WARTOŚĆ
Obiekt EntitySet	Nie	Jeśli funkcja zwraca kolekcję jednostek typy wartości EntitySet musi być zestaw jednostek do której należy kolekcji. W przeciwnym razie EntitySet atrybutu nie może być używany.
IsComposable	Nie	Jeśli wartość jest równa true, funkcja jest konfigurowalna (funkcji zwracającej tabelę) i mogą być używane w zapytaniu LINQ. Wartość domyślna to false .

NOTE

Dowolna liczba atrybutów adnotacji (niestandardowe atrybuty XML) można stosować do **FunctionImport** elementu. Jednak atrybutów niestandardowych, które nie mogą należeć do przestrzeni nazw XML, który jest zarezerwowany dla CSDL. W pełni kwalifikowanej nazwy dowolne dwa atrybuty niestandardowe nie może być taka sama.

Przykład

W poniższym przykładzie przedstawiono **FunctionImport** element, który przyjmuje jeden parametr i zwraca kolekcję typów jednostek:

```
<FunctionImport Name="GetStudentGrades"
    EntitySet="StudentGrade"
    ReturnType="Collection(SchoolModel.StudentGrade)">
    <Parameter Name="StudentID" Mode="In" Type="Int32" />
</FunctionImport>
```

Kluczowym elementem (CSDL)

Klucz element jest elementem podrzędnym elementu **EntityType** i definiuje *klucz jednostki* (właściwość lub ustaw właściwości typu jednostki, które określają tożsamość). Właściwości, które tworzą klucz jednostki są wybierane w czasie projektowania. Wartości właściwości klucza jednostki musi jednoznacznie identyfikować wystąpienie jednostki typu w obrębie jednostki ustawienie w czasie wykonywania. Należy wybrać właściwości, które tworzą klucz jednostki w celu zagwarantowania unikalności wystąpień w zestawie jednostek. **Klucz** element definiuje klucz jednostki, odwołując się do przynajmniej jednej w właściwości typu jednostki.

Klucz element może mieć następujące elementy podrzędne:

- **PropertyRef** (jeden lub więcej elementów)
- Elementów adnotacji (zero lub więcej elementów)

Odpowiednie atrybuty

Dowolna liczba atrybutów adnotacji (niestandardowe atrybuty XML) można stosować do **klucz** elementu. Jednak atrybutów niestandardowych, które nie mogą należeć do przestrzeni nazw XML, który jest zarezerwowany dla CSDL. W pełni kwalifikowanej nazwy dowolne dwa atrybuty niestandardowe nie może być taka sama.

Przykład

Poniższy przykład definiuje typ jednostki o nazwie **książki**. Klucz jednostki jest zdefiniowany, odwołując się do **ISBN** właściwość typu jednostki.

```

<EntityType Name="Book">
  <Key>
    <PropertyRef Name="ISBN" />
  </Key>
  <Property Type="String" Name="ISBN" Nullable="false" />
  <Property Type="String" Name="Title" Nullable="false" />
  <Property Type="Decimal" Name="Revision" Nullable="false" Precision="29" Scale="29" />
  <NavigationProperty Name="Publisher" Relationship="BooksModel.PublishedBy"
    FromRole="Book" ToRole="Publisher" />
  <NavigationProperty Name="Authors" Relationship="BooksModel.WrittenBy"
    FromRole="Book" ToRole="Author" />
</EntityType>

```

ISBN właściwość jest dobrym wyborem dla klucza jednostki, ponieważ International Standard książki numer (ISBN) unikalowo identyfikuje książki.

Poniższy przykład przedstawia typ jednostki (**Autor**) zawierający klucz jednostki, która składa się z dwóch właściwości **nazwa i adres**.

```

<EntityType Name="Author">
  <Key>
    <PropertyRef Name="Name" />
    <PropertyRef Name="Address" />
  </Key>
  <Property Type="String" Name="Name" Nullable="false" />
  <Property Type="String" Name="Address" Nullable="false" />
  <NavigationProperty Name="Books" Relationship="BooksModel.WrittenBy"
    FromRole="Author" ToRole="Book" />
</EntityType>

```

Za pomocą **nazwa i adres** jednostki klucz jest wyboru rozsądny, ponieważ dwóch autorów o takiej samej nazwie jest mało prawdopodobne na żywo na ten sam adres. Jednak ten wybór dla klucza jednostki absolutnie nie gwarantuje klucze unikatowe jednostki w zestawie jednostek. Dodawanie właściwości, takie jak **wartość IDAutora**, który może być używany do jednoznacznego identyfikowania Autora może w tym przypadku zalecane.

Element członkowski (CSDL)

Elementu członkowskiego element jest elementem podrzędnym elementu EnumType i definiuje składową typu wyliczeniowego.

Odpowiednie atrybuty

W poniższej tabeli opisano atrybuty, które mogą być stosowane do **FunctionImport** elementu.

NAZWA ATRYBUTU	JEST WYMAGANY	WARTOŚĆ
Nazwa	Tak	Nazwa elementu członkowskiego.
Wartość	Nie	Wartość elementu członkowskiego. Domyślnie pierwszy element członkowski ma wartość 0, a wartość kolejnych moduł wyliczający jest zwiększana o 1. Może istnieć wiele składowych o tej samej wartości.

NOTE

Dowolna liczba atrybutów adnotacji (niestandardowe atrybuty XML) można stosować do **FunctionImport** elementu. Jednak atrybutów niestandardowych, które nie mogą należeć do przestrzeni nazw XML, który jest zarezerwowany dla CSDL. W pełni kwalifikowanej nazwy dowolne dwa atrybuty niestandardowe nie może być taka sama.

Przykład

W poniższym przykładzie przedstawiono **EnumType** element z trzech **elementu członkowskiego** elementy:

```
<EnumType Name="Color">
  <Member Name="Red" Value="1"/>
  <Member Name="Green" Value="3" />
  <Member Name="Blue" Value="5"/>
</EntityType>
```

Element NavigationProperty — Element (CSDL)

A **element NavigationProperty** element definiuje właściwość nawigacji, który zawiera dokumentację na drugim końcu skojarzenia. W odróżnieniu od właściwości zdefiniowane przy użyciu elementu właściwości właściwości nawigacji definiuje kształt i właściwości danych. Zapewniają one sposób przechodzenia skojarzenie między dwoma typami encji.

Należy pamiętać, że właściwości nawigacji są opcjonalne na obu typów jednostek na końcach asocjacji. Jeśli zdefiniujesz właściwości nawigacji na jednym typie jednostek na końcu asocjacji, ma Zdefiniuj właściwość nawigacji typu jednostki na drugim końcu skojarzenia.

Typ danych zwracanych przez właściwość nawigacji jest ustalana liczebność jej punkt końcowy skojarzenia zdalnego. Na przykład, założmy, że właściwość nawigacji, **OrdersNavProp**, istnieje na **klienta** typu jednostki i przechodzi skojarzenia typu jeden do wielu między **klienta** i **Kolejność**. Ponieważ punkt końcowy skojarzenia zdalnego dla właściwości nawigacji ma liczebność wiele (*), jego typ danych to kolekcja (z **kolejności**). Podobnie, jeśli właściwość nawigacji, **CustomerNavProp**, istnieje na **kolejności** typu jednostki, będzie jego typu danych **klienta** ponieważ liczebności zakończenia zdalnego jeden (1).

A **element NavigationProperty** element może mieć następujących elementów podrzędnych (w podanej kolejności):

- Dokumentacja (zero lub jeden element)
- Elementów adnotacji (zero lub więcej elementów)

Odpowiednie atrybuty

W poniższej tabeli opisano atrybuty, które mogą być stosowane do **element NavigationProperty** elementu.

NAZWA ATRYBUTU	JEST WYMAGANY	WARTOŚĆ
Nazwa	Tak	Nazwa właściwości nawigacji.
Relacja	Tak	Nazwa skojarzenia, które znajduje się w zakresie modelu.

NAZWA ATRYBUTU	JEST WYMAGANY	WARTOŚĆ
ToRole	Tak	Koniec asocjacji, na której kończy się nawigacji. Wartość ToRole atrybut musi być taka sama jak wartość jednego z rolí atrybuty zdefiniowane w jednym z punktów końcowych skojarzenia (zdefiniowanej w elemencie punkt końcowy skojarzenia).
Wartości elementów FromRole	Tak	Koniec skojarzenia, w którym rozpoczyna się nawigacji. Wartość FromRole atrybut musi być taka sama jak wartość jednego z rolí atrybuty zdefiniowane w jednym z punktów końcowych skojarzenia (zdefiniowanej w elemencie punkt końcowy skojarzenia).

NOTE

Dowolna liczba atrybutów adnotacji (niestandardowe atrybuty XML) można stosować do **element NavigationProperty** elementu. Jednak atrybutów niestandardowych, które nie mogą należeć do przestrzeni nazw XML, który jest zarezerwowany dla CSDL. W pełni kwalifikowanej nazwy dowolne dwa atrybuty niestandardowe nie może być taka sama.

Przykład

W poniższym przykładzie zdefiniowano typ jednostki (**książki**) z dwóch właściwości nawigacji (**PublishedBy** i **WrittenBy**):

```
<EntityType Name="Book">
  <Key>
    <PropertyRef Name="ISBN" />
  </Key>
  <Property Type="String" Name="ISBN" Nullable="false" />
  <Property Type="String" Name="Title" Nullable="false" />
  <Property Type="Decimal" Name="Revision" Nullable="false" Precision="29" Scale="29" />
  <NavigationProperty Name="Publisher" Relationship="BooksModel.PublishedBy"
    FromRole="Book" ToRole="Publisher" />
  <NavigationProperty Name="Authors" Relationship="BooksModel.WrittenBy"
    FromRole="Book" ToRole="Author" />
</EntityType>
```

Element OnDelete (CSDL)

OnDelete element języka definicji schematu koncepcyjnego (CSDL) definiuje zachowanie, która jest połączona z skojarzeniem. Jeśli **akcji** ma ustawioną wartość atrybutu **Cascade** na jednym końcu asocjacji powiązanych typów jednostek na drugim końcu skojarzenia zostaną usunięte po usunięciu typu jednostki, które na pierwszy punkt końcowy. Jeśli skojarzenie między dwoma typami encji jest relacją klucza podstawowego klucza do podstawowej, a następnie załadować obiektu zależnego zostanie usunięty po usunięciu obiektu podmiotu zabezpieczeń na drugim końcu skojarzenia niezależnie od wartości **OnDelete** Specyfikacja.

NOTE

OnDelete element ma wpływ tylko na zachowanie środowiska uruchomieniowego aplikacji; nie ma wpływu na zachowanie w źródle danych. Zachowanie zdefiniowane w źródle danych powinno być taka sama jak zachowanie zdefiniowane w aplikacji.

OnDelete element może mieć następujących elementów podrzędnych (w podanej kolejności):

- Dokumentacja (zero lub jeden element)
- Elementów adnotacji (zero lub więcej elementów)

Odpowiednie atrybuty

W poniższej tabeli opisano atrybuty, które mogą być stosowane do **OnDelete** elementu.

NAZWA ATRYBUTU	JEST WYMAGANY	WARTOŚĆ
Akcja	Tak	Kaskadowe lub Brak . Jeśli Cascade , typy jednostek zależnych zostaną usunięte po usunięciu typu podmiotu zabezpieczeń jednostki. Jeśli Brak , typy jednostek zależnych nie zostaną usunięte po usunięciu typu podmiotu zabezpieczeń jednostki.

NOTE

Dowolna liczba atrybutów adnotacji (niestandardowe atrybuty XML) można stosować do **skojarzenia** elementu. Jednak atrybutów niestandardowych, które nie mogą należeć do przestrzeni nazw XML, który jest zarezerwowany dla CSDL. W pełni kwalifikowanej nazwy dowolne dwa atrybuty niestandardowe nie może być taka sama.

Przykład

W poniższym przykładzie przedstawiono **skojarzenia** element, który definiuje **CustomerOrders** skojarzenia.

OnDelete elementu wskazuje, że wszystkie **zamówienia** powiązanych z określonego **klienta** i zostały załadowane do obiektu ObjectContext zostaną usunięte po **Klient** zostanie usunięty.

```
<Association Name="CustomerOrders">
  <End Type="ExampleModel.Customer" Role="Customer" Multiplicity="1">
    <OnDelete Action="Cascade" />
  </End>
  <End Type="ExampleModel.Order" Role="Order" Multiplicity="*" />
</Association>
```

Parameter — Element (CSDL)

Parametru element język definicji schematu koncepcyjnego (CSDL) może być elementem podrzędnym elementu **FunctionImport** lub elemencie **Function**.

FunctionImport Element aplikacji

A **parametru** — element (jako element podrzędny elementu **FunctionImport** elementu) służy do definiowania parametry wejściowe i wyjściowe w przypadku importowania funkcji, które są zadeklarowane w CSDL.

Parametru element może mieć następujących elementów podrzędnych (w podanej kolejności):

- Dokumentacja (zero lub jeden elementy dozwolone)
- Elementów adnotacji (zero lub więcej elementów dozwolone)

Odpowiednie atrybuty

W poniższej tabeli opisano atrybuty, które mogą być stosowane do **parametru** elementu.

NAZWA ATRYBUTU	JEST WYMAGANY	WARTOŚĆ
Nazwa	Tak	Nazwa parametru.
Typ	Tak	Typ parametru. Wartość musi być EDMSimpleType lub typ złożony, który znajduje się w zakresie modelu.
Tryb	Nie	W, się , lub InOut w zależności od tego, czy parametr jest danych wejściowych, danych wyjściowych lub parametr input/output.
Element maxLength	Nie	Maksymalna dozwolona liczba znaków parametru.
Precyzja	Nie	Dokładność parametru.
Skala	Nie	Skala parametru.
SRID	Nie	Identyfikator odwołania przestrzennego systemu. Prawidłowy tylko dla parametrów typów przestrzennych. Aby uzyskać więcej informacji, zobacz SRID i SRID (SQL Server) .

NOTE

Dowolna liczba atrybutów adnotacji (niestandardowe atrybuty XML) można stosować do **parametru** elementu. Jednak atrybutów niestandardowych, które nie mogą należeć do przestrzeni nazw XML, który jest zarezerwowany dla CSDL. W pełni kwalifikowanej nazwy dowolne dwa atrybuty niestandardowe nie może być taka sama.

Przykład

W poniższym przykładzie przedstawiono **FunctionImport** elementu za pomocą jednego **parametru** elementu podrzędnego. Funkcja przyjmuje jeden parametr wejściowy i zwraca kolekcję typów jednostek.

```
<FunctionImport Name="GetStudentGrades"
    EntitySet="StudentGrade"
    ReturnType="Collection(SchoolModel.StudentGrade)">
    <Parameter Name="StudentID" Mode="In" Type="Int32" />
</FunctionImport>
```

Funkcja elementu aplikacji

A **parametru** — element (jako element podrzędny elementu **funkcja** elementu) definiuje parametry dla funkcji, które zdefiniowana lub zadeklarowana w modelu koncepcyjnym.

Parametru element może mieć następujących elementów podrzędnych (w podanej kolejności):

- Dokumentacja (zero lub jeden elementy)
- Typ CollectionType (zero lub jeden elementy)
- Element ReferenceType (zero lub jeden elementy)
- RowType (zero lub jeden elementy)

NOTE

Tylko jeden z **CollectionType**, **ReferenceType**, lub **RowType** elementy mogą być elementem podrzędnym właściwość elementu.

- Elementów adnotacji (zero lub więcej elementów dozwolone)

NOTE

Adnotacja elementów musi występować po wszystkich innych elementów podrzędnych. Elementów adnotacji są dozwolone tylko w wersji 2 CSDL i nowszych wersjach.

Odpowiednie atrybuty

W poniższej tabeli opisano atrybuty, które mogą być stosowane do **parametru** elementu.

NAZWA ATRYBUTU	JEST WYMAGANY	WARTOŚĆ
Nazwa	Tak	Nazwa parametru.
Typ	Nie	Typ parametru. Parametr może być dowolny z następujących typów (lub kolekcje z tych typów): EdmSimpleType Typ jednostki Typ złożony Typ wiersza typ odwołania
Dopuszcza wartości null	Nie	Wartość true , (wartość domyślna) lub False w zależności od tego, czy właściwość może mieć null wartość.
defaultValue	Nie	Wartość domyślna właściwości.
Element maxLength	Nie	Maksymalna długość wartości właściwości.
Wartości	Nie	Wartość true , lub False w zależności od tego, czy przechowywana wartość właściwości jako ciąg znaków o stałej długości.
Precyzja	Nie	Dokładność wartości właściwości.
Skala	Nie	Skala wartości właściwości.
SRID	Nie	Identyfikator odwołania przestrzennego systemu. Prawidłowy tylko w przypadku właściwości typów przestrzennych. Aby uzyskać więcej informacji, zobacz SRID i SRID (SQL Server) .
Unicode	Nie	Wartość true , lub False w zależności od tego, czy przechowywana wartość właściwości jako ciąg Unicode.

NAZWA ATRYBUTU	JEST WYMAGANY	WARTOŚĆ
Sortowanie	Nie	Ciąg, który określa kolejność sortowania, które ma być używany w źródle danych.

NOTE

Dowolna liczba atrybutów adnotacji (niestandardowe atrybuty XML) można stosować do **parametru** elementu. Jednak atrybutów niestandardowych, które nie mogą należeć do przestrzeni nazw XML, który jest zarezerwowany dla CSDL. W pełni kwalifikowanej nazwy dowolne dwa atrybuty niestandardowe nie może być taka sama.

Przykład

W poniższym przykładzie przedstawiono **funkcja** element, który korzysta z jednego **parametru** elementu podzielnego, aby zdefiniować parametr funkcji.

```
<Function Name="GetYearsEmployed" ReturnType="Edm.Int32">
<Parameter Name="Instructor" Type="SchoolModel.Person" />
<DefiningExpression>
Year(CurrentDateTime()) - Year(cast(Instructor.HireDate as DateTime))
</DefiningExpression>
</Function>
```

Element jednostki (CSDL)

Jednostki element język definicji schematu koncepcyjnego (CSDL) jest elementem podzielnym do elementu **ReferentialConstraint**, który definiuje główny koniec ograniczenia referencyjnego. A **ReferentialConstraint** element definiuje funkcjonalność, która jest podobna do ograniczenia integralności referencyjnej w relacyjnej bazie danych. W ten sam sposób, w kolumnie (lub kolumny) z tabeli bazy danych odwołać się do klucza podstawowego z innej tabeli. Właściwość (lub właściwości), typu jednostki odwoływać się do klucza jednostki innego typu jednostki. Typ jednostki, do którego istnieje odwołanie jest wywoływana *jednostki zakończenia ograniczenia*. Typ jednostki, który odwołuje się do zakończenia podmiotu zabezpieczeń jest wywoływana *zależne zakończenia ograniczenia*.

PropertyRef elementy są używane do określania, klucze, które są przywoływane przez zależne zakończenia.

Jednostki element może mieć następujących elementów podzielnnych (w podanej kolejności):

- **PropertyRef** (jeden lub więcej elementów)
- Elementów adnotacji (zero lub więcej elementów)

Odpowiednie atrybuty

W poniższej tabeli opisano atrybuty, które mogą być stosowane do **jednostki** elementu.

NAZWA ATRYBUTU	JEST WYMAGANY	WARTOŚĆ
Rola	Tak	Nazwa typu jednostki, na końcu jednostki skojarzenia.

NOTE

Dowolna liczba atrybutów adnotacji (niestandardowe atrybuty XML) można stosować do **jednostki** elementu. Jednak atrybutów niestandardowych, które nie mogą należeć do przestrzeni nazw XML, który jest zarezerwowany dla CSDL. W pełni kwalifikowanej nazwy dowolne dwa atrybuty niestandardowe nie może być taka sama.

Przykład

W poniższym przykładzie przedstawiono **ReferentialConstraint** element, który jest częścią definicji **PublishedBy** skojarzenia. **Identyfikator** właściwość **wydawcy** typu jednostki stanowi główny koniec ograniczenia referencyjnego.

```
<Association Name="PublishedBy">
  <End Type="BooksModel.Book" Role="Book" Multiplicity="*" >
  </End>
  <End Type="BooksModel.Publisher" Role="Publisher" Multiplicity="1" />
  <ReferentialConstraint>
    <Principal Role="Publisher">
      <PropertyRef Name="Id" />
    </Principal>
    <Dependent Role="Book">
      <PropertyRef Name="PublisherId" />
    </Dependent>
  </ReferentialConstraint>
</Association>
```

Property — Element (CSDL)

Właściwość element język definicji schematu koncepcyjnego (CSDL) może być elementem podrzędnym elementu EntityType, ComplexType element lub RowType element.

Obiekt EntityType i ComplexType elementu aplikacji

Właściwość elementów (jako elementy podrzędne **EntityType** lub **ComplexType** elementy) zdefiniuj kształt i charakterystyki danych, która będzie zawierać wystąpienia typu jednostki lub typ złożony. Właściwości w modelu koncepcyjnym są analogiczne do właściwości, które są zdefiniowane w klasie. W ten sam sposób, że właściwości w klasie określenia kształtu elementu klasy i zawierają informacje o obiektach właściwości w modelu koncepcyjnym określenia kształtu typu jednostki i zawierają informacje na temat wystąpień typu jednostki.

Właściwość element może mieć następujących elementów podrzędnych (w podanej kolejności):

- Element documentation (zero lub jeden elementy dozwolone)
- Elementów adnotacji (zero lub więcej elementów dozwolone)

Następujące aspekty mogą być stosowane do **właściwość** element: **Nullable**, **DefaultValue**, **MaxLength**, **Wartości, dokładności, skalowania, Unicode, sortowania, Właściwość ConcurrencyMode**. Zestawy reguł są atrybuty XML, które zawierają informacje dotyczące sposobu wartości właściwości są przechowywane w magazynie danych.

NOTE

Aspektami może być stosowany tylko do właściwości typu **EDMSimpleType**.

Odpowiednie atrybuty

W poniższej tabeli opisano atrybuty, które mogą być stosowane do **właściwość** elementu.

NAZWA ATRYBUTU	JEST WYMAGANY	WARTOŚĆ
Nazwa	Tak	Nazwa właściwości.

NAZWA ATRYBUTU	JEST WYMAGANY	WARTOŚĆ
Typ	Tak	Typ wartości właściwości. Typ wartości właściwości musi być EDMSimpleType lub typ złożony (wskazywanym przez w pełni kwalifikowaną nazwą), który znajduje się w zakresie modelu.
Dopuszcza wartości null	Nie	Wartość true , (wartość domyślna) lub False w zależności od tego, czy właściwość może mieć wartości null. [!NOTE]
> W wersji 1 CSDL musi mieć właściwość typu złożonego <code>Nullable="False"</code> .		
defaultValue	Nie	Wartość domyślna właściwości.
Element maxLength	Nie	Maksymalna długość wartości właściwości.
Wartości	Nie	Wartość true , lub False w zależności od tego, czy przechowywana wartość właściwości jako ciąg znaków o stałej długości.
Precyzja	Nie	Dokładność wartości właściwości.
Skala	Nie	Skala wartości właściwości.
SRID	Nie	Identyfikator odwołania przestrzennego systemu. Prawidłowy tylko w przypadku właściwości typów przestrzennych. Aby uzyskać więcej informacji, zobacz SRID i SRID (SQL Server) .
Unicode	Nie	Wartość true , lub False w zależności od tego, czy przechowywana wartość właściwości jako ciąg Unicode.
Sortowanie	Nie	Ciąg, który określa kolejność sortowania, które ma być używany w źródle danych.
Właściwość ConcurrencyMode	Nie	Brak (wartość domyślna) lub stałe . Jeśli wartość jest równa stałe , zostanie użyta wartość właściwości w kontroli optymistycznej współbieżności.

NOTE

Dowolna liczba atrybutów adnotacji (niestandardowe atrybuty XML) można stosować do **właściwość** elementu. Jednak atrybutów niestandardowych, które nie mogą należeć do przestrzeni nazw XML, który jest zarezerwowany dla CSDL. W pełni kwalifikowanej nazwy dowolne dwa atrybuty niestandardowe nie może być taka sama.

Przykład

W poniższym przykładzie przedstawiono **EntityType** element z trzech **właściwość** elementy:

```
<EntityType Name="Book">
  <Key>
    <PropertyRef Name="ISBN" />
  </Key>
  <Property Type="String" Name="ISBN" Nullable="false" />
  <Property Type="String" Name="Title" Nullable="false" />
  <Property Type="Decimal" Name="Revision" Nullable="false" Precision="29" Scale="29" />
  <NavigationProperty Name="Publisher" Relationship="BooksModel.PublishedBy"
    FromRole="Book" ToRole="Publisher" />
  <NavigationProperty Name="Authors" Relationship="BooksModel.WrittenBy"
    FromRole="Book" ToRole="Author" />
</EntityType>
```

W poniższym przykładzie przedstawiono **ComplexType** element z pięciu **właściwość** elementy:

```
<ComplexType Name="Address" >
  <Property Type="String" Name="StreetAddress" Nullable="false" />
  <Property Type="String" Name="City" Nullable="false" />
  <Property Type="String" Name="StateOrProvince" Nullable="false" />
  <Property Type="String" Name="Country" Nullable="false" />
  <Property Type="String" Name="PostalCode" Nullable="false" />
</ComplexType>
```

RowType Element aplikacji

Właściwość elementów (jako element podrzędny elementu **RowType** elementu) zdefiniuj kształt i charakterystyki dane, które mogą być przekazywane do lub zwracany z funkcji definiowanych przez model.

Właściwość element może mieć dokładnie jeden z następujących elementów podrzędnych:

- Typ CollectionType
- Element referenceType
- RowType

Właściwość element może mieć żadnych liczba elementów podrzędnych w adnotacji.

NOTE

Elementów adnotacji są dozwolone tylko w wersji 2 CSDL i nowszych wersjach.

Odpowiednie atrybuty

W poniższej tabeli opisano atrybuty, które mogą być stosowane do **właściwość** elementu.

NAZWA ATRYBUTU	JEST WYMAGANY	WARTOŚĆ
Nazwa	Tak	Nazwa właściwości.
Typ	Tak	Typ wartości właściwości.
Dopuszcza wartości null	Nie	Wartość true , (wartość domyślana) lub False w zależności od tego, czy właściwość może mieć wartości null. [!NOTE]

NAZWA ATRYBUTU	JEST WYMAGANY	WARTOŚĆ
> W wersji 1 CSDL musi mieć właściwość typu złożonego <code>Nullable="False"</code> .		
defaultValue	Nie	Wartość domyślna właściwości.
Element maxLength	Nie	Maksymalna długość wartości właściwości.
Wartości	Nie	Wartość true , lub False w zależności od tego, czy przechowywana wartość właściwości jako ciąg znaków o stałej długości.
Precyzja	Nie	Dokładność wartości właściwości.
Skala	Nie	Skala wartości właściwości.
SRID	Nie	Identyfikator odwołania przestrzennego systemu. Prawidłowy tylko w przypadku właściwości typów przestrzennych. Aby uzyskać więcej informacji, zobacz SRID (SQL Server) .
Unicode	Nie	Wartość true , lub False w zależności od tego, czy przechowywana wartość właściwości jako ciąg Unicode.
Sortowanie	Nie	Ciąg, który określa kolejność sortowania, który ma być używany w źródle danych.

NOTE

Dowolna liczba atrybutów adnotacji (niestandardowe atrybuty XML) można stosować do **właściwość** elementu. Jednak atrybutów niestandardowych, które nie mogą należeć do przestrzeni nazw XML, który jest zarezerwowany dla CSDL. W pełni kwalifikowanej nazwy dowolne dwa atrybuty niestandardowe nie może być taka sama.

Przykład

W poniższym przykładzie przedstawiono **właściwość** elementy służące do określenia kształtu elementu zwracany typ funkcji definiowanych przez model.

```

<Function Name="LastNamesAfter">
  <Parameter Name="someString" Type="Edm.String" />
  <ReturnType>
    <CollectionType>
      <RowType>
        <Property Name="FirstName" Type="Edm.String" Nullable="false" />
        <Property Name="LastName" Type="Edm.String" Nullable="false" />
      </RowType>
    </CollectionType>
  </ReturnType>
  <DefiningExpression>
    SELECT VALUE ROW(p.FirstName, p.LastName)
    FROM SchoolEntities.People AS p
    WHERE p.LastName &gt;= somestring
  </DefiningExpression>
</Function>

```

Element PropertyRef (CSDL)

PropertyRef element języka definicji schematu koncepcyjnego (CSDL) odwołuje się do właściwości typu jednostki, aby wskazać, że właściwość wykona jedną z następujących ról:

- Część klucza jednostki (właściwość lub ustaw właściwości typu jednostki, które określają tożsamość). Co najmniej jeden **PropertyRef** elementy mogą być używane do definiowania klucza jednostki.
- Koniec zależnych lub jednostki ograniczenia referencyjnego.

PropertyRef element może mieć tylko elementów adnotacji (zero lub więcej) jako elementy podzielone.

NOTE

Elementów adnotacji są dozwolone tylko w wersji 2 CSDL i nowszych wersjach.

Odpowiednie atrybuty

W poniższej tabeli opisano atrybuty, które mogą być stosowane do **PropertyRef** elementu.

NAZWA ATRYBUTU	JEST WYMAGANY	WARTOŚĆ
Nazwa	Tak	Nazwa właściwości, której dotyczy odwołanie.

NOTE

Dowolna liczba atrybutów adnotacji (niestandardowe atrybuty XML) można stosować do **PropertyRef** elementu. Jednak atrybutów niestandardowych, które nie mogą należeć do przestrzeni nazw XML, który jest zarezerwowany dla CSDL. W pełni kwalifikowanej nazwy dowolne dwa atrybuty niestandardowe nie może być taka sama.

Przykład

Poniższy przykład definiuje typ jednostki (**książki**). Klucz jednostki jest zdefiniowany, odwołując się do **ISBN** właściwość typu jednostki.

```

<EntityType Name="Book">
  <Key>
    <PropertyRef Name="ISBN" />
  </Key>
  <Property Type="String" Name="ISBN" Nullable="false" />
  <Property Type="String" Name="Title" Nullable="false" />
  <Property Type="Decimal" Name="Revision" Nullable="false" Precision="29" Scale="29" />
  <NavigationProperty Name="Publisher" Relationship="BooksModel.PublishedBy"
    FromRole="Book" ToRole="Publisher" />
  <NavigationProperty Name="Authors" Relationship="BooksModel.WrittenBy"
    FromRole="Book" ToRole="Author" />
</EntityType>

```

W następnym przykładzie dwóch **PropertyRef** elementy są używane w celu wskazania, że dwie właściwości (**Identyfikator** i **PublisherId**) są głównym i zależnym końców typu odwołanie ograniczenie.

```

<Association Name="PublishedBy">
  <End Type="BooksModel.Book" Role="Book" Multiplicity="*" />
  </End>
  <End Type="BooksModel.Publisher" Role="Publisher" Multiplicity="1" />
  <ReferentialConstraint>
    <Principal Role="Publisher">
      <PropertyRef Name="Id" />
    </Principal>
    <Dependent Role="Book">
      <PropertyRef Name="PublisherId" />
    </Dependent>
  </ReferentialConstraint>
</Association>

```

Element ReferenceType (CSDL)

ReferenceType element język definicji schematu koncepcyjnego (CSDL) Określa odwołanie do typu jednostki.

ReferenceType element może być elementem podrzędnym następujące elementy:

- ReturnType (funkcja)
- Parametr
- Typ CollectionType

ReferenceType element jest używany podczas definiowania parametr lub zwracany typ funkcji.

A **ReferenceType** element może mieć następujących elementów podrzędnych (w podanej kolejności):

- Dokumentacja (zero lub jeden element)
- Elementów adnotacji (zero lub więcej elementów)

Odpowiednie atrybuty

W poniższej tabeli opisano atrybuty, które mogą być stosowane do **ReferenceType** elementu.

NAZWA ATRYBUTU	JEST WYMAGANY	WARTOŚĆ
Typ	Tak	Nazwa typu jednostki, do którego nastąpiło odwołanie.

NOTE

Dowolna liczba atrybutów adnotacji (niestandardowe atrybuty XML) można stosować do **ReferenceType** elementu. Jednak atrybutów niestandardowych, które nie mogą należeć do przestrzeni nazw XML, który jest zarezerwowany dla CSDL. W pełni kwalifikowanej nazwy dowolne dwa atrybuty niestandardowe nie może być taka sama.

Przykład

W poniższym przykładzie przedstawiono **ReferenceType** element używany jako element podrzędny elementu **parametru** elementu w funkcji definiowanych przez model, który akceptuje odwołanie do **osoby** jednostki Typ:

```
<Function Name="GetYearsEmployed" ReturnType="Edm.Int32">
    <Parameter Name="instructor">
        <ReferenceType Type="SchoolModel.Person" />
    </Parameter>
    <DefiningExpression>
        Year(CurrentDateTime()) - Year(cast(instructor.HireDate as DateTime))
    </DefiningExpression>
</Function>
```

W poniższym przykładzie przedstawiono **ReferenceType** element używany jako element podrzędny elementu **ReturnType** — element (funkcja) w funkcji definiowanych przez model, który zwraca odwołanie do **osobytyp** jednostki:

```
<Function Name="GetPersonReference">
    <Parameter Name="p" Type="SchoolModel.Person" />
    <ReturnType>
        <ReferenceType Type="SchoolModel.Person" />
    </ReturnType>
    <DefiningExpression>
        REF(p)
    </DefiningExpression>
</Function>
```

Element ReferentialConstraint (CSDL)

A **ReferentialConstraint** element w język definicji schematu koncepcyjnego (CSDL) definiuje funkcjonalność, która jest podobna do ograniczenia integralności referencyjnej w relacyjnej bazie danych. W ten sam sposób, w kolumnie (lub kolumny) z tabeli bazy danych odwołać się do klucza podstawowego z innej tabeli Właściwość (lub właściwości), typu jednostki odwoływać się do klucza jednostki innego typu jednostki. Typ jednostki, do którego istnieje odwołanie jest wywoływana *jednostki zakończenia ograniczenia*. Typ jednostki, który odwołuje się do zakończenia podmiotu zabezpieczeń jest wywoywana *zależne zakończenia ograniczenia*.

Jeśli klucz obcy, która jest widoczna w jednej jednostce typu odwołuje się do właściwości na inny typ jednostki, **ReferentialConstraint** element definiuje skojarzenie między tymi typami dwie jednostki. Ponieważ **ReferentialConstraint** element udostępnia dowiedzieć się, jak dwa typy jednostek są powiązane z nie odpowiadającym elementu **AssociationSetMapping** niezbędne jest w języku specyfikacji mapowanie (MSL). Skojarzenie między dwoma typami jednostek, które nie mają klucze obce udostępniane musi mieć odpowiedni **AssociationSetMapping** element, aby zamapować informacji o skojarzeniu ze źródłem danych.

Jeśli klucz obcy nie jest uwidaczniana typu encji **ReferentialConstraint** elementu można zdefiniować tylko klucz do podstawowego ograniczenia klucza podstawowego między typem jednostki i innego typu jednostki.

A **ReferentialConstraint** element może mieć następujących elementów podrzędnych (w podanej kolejności):

- Dokumentacja (zero lub jeden element)

- Podmiot zabezpieczeń (dokładnie jeden element)
- Zależnych od ustawień lokalnych (dokładnie jeden element)
- Elementów adnotacji (zero lub więcej elementów)

Odpowiednie atrybuty

ReferentialConstraint element może mieć dowolną liczbę adnotacji atrybutów (niestandardowe atrybuty XML). Jednak atrybutów niestandardowych, które nie mogą należeć do przestrzeni nazw XML, który jest zarezerwowany dla CSDL. W pełni kwalifikowanej nazwy dowolne dwa atrybuty niestandardowe nie może być taka sama.

Przykład

W poniższym przykładzie przedstawiono **ReferentialConstraint** używany jako część definicji elementu **PublishedBy** skojarzenia.

```
<Association Name="PublishedBy">
  <End Type="BooksModel.Book" Role="Book" Multiplicity="*" />
  </End>
  <End Type="BooksModel.Publisher" Role="Publisher" Multiplicity="1" />
  <ReferentialConstraint>
    <Principal Role="Publisher">
      <PropertyRef Name="Id" />
    </Principal>
    <Dependent Role="Book">
      <PropertyRef Name="PublisherId" />
    </Dependent>
  </ReferentialConstraint>
</Association>
```

Element ReturnType (funkcja) (CSDL)

ReturnType (funkcja) element w języku definicji schematu koncepcyjnego (CSDL) określa typ zwracany dla funkcji, która jest zdefiniowana w elemencie Function. Można również określić typ zwracany z funkcji **ReturnType** atrybutu.

Zwraca typy mogą być **EdmSimpleType**, typ jednostki, typu złożonego, typ wiersza, typu referencyjnego lub zbiór jednego z tych typów.

Zwracany typ funkcji można określić za pomocą albo **typu** atrybutu **ReturnType** elementu (funkcja), czy jeden z następujących elementów podrzędnych:

- Typ CollectionType
- Element referenceType
- RowType

NOTE

Model nie zostanie zweryfikowany, jeśli określisz, że funkcja zwracany typ zarówno **typu** atrybutu **ReturnType** elementu (funkcja) i jeden z elementów podrzędnych.

Odpowiednie atrybuty

W poniższej tabeli opisano atrybuty, które mogą być stosowane do **ReturnType** — element (funkcja).

NAZWA ATRYBUTU	JEST WYMAGANY	WARTOŚĆ
ReturnType	Nie	Typ zwracany przez funkcję.

NOTE

Dowolna liczba atrybutów adnotacji (niestandardowe atrybuty XML) można stosować do **ReturnType** — element (funkcja). Jednak atrybutów niestandardowych, które nie mogą należeć do przestrzeni nazw XML, który jest zarezerwowany dla CSDL. W pełni kwalifikowanej nazwy dowolne dwa atrybuty niestandardowe nie może być taka sama.

Przykład

W poniższym przykładzie użyto **funkcja** elementu, aby zdefiniować funkcji, która zwraca liczbę lat książki znajdował się w drukowania. Należy zauważyć, że typ zwracany jest określony przez **typu** atrybutu **ReturnType** — element (funkcja).

```
<Function Name="GetYearsInPrint">
  <ReturnType Type=="Edm.Int32">
    <Parameter Name="book" Type="BooksModel.Book" />
    <DefiningExpression>
      Year(CurrentDateTime()) - Year(cast(book.PublishedDate as DateTime))
    </DefiningExpression>
  </ReturnType>
</Function>
```

Element ReturnType (FunctionImport) (CSDL)

ReturnType elementu (element FunctionImport) w język definicji schematu koncepcyjnego (CSDL) określa typ zwracany dla funkcji, która jest zdefiniowana w elemencie FunctionImport. Można również określić typ zwracany z funkcji **ReturnType** atrybutu.

Zwraca typy mogą być dowolnej kolekcji typu jednostki, typu złożonego lub **EdmSimpleType**,

Zwracany typ funkcji jest określony za pomocą **typu** atrybutu **ReturnType** elementu (element FunctionImport).

Odpowiednie atrybuty

W poniższej tabeli opisano atrybuty, które mogą być stosowane do **ReturnType** elementu (element FunctionImport).

NAZWA ATRYBUTU	JEST WYMAGANY	WARTOŚĆ
Typ	Nie	Typ, który zwraca funkcja. Wartość musi być kolekcją ComplexType, EntityType lub EDMSimpleType.
Obiekt EntitySet	Nie	Jeśli funkcja zwraca kolekcję jednostek typy wartości EntitySet musi być zestaw jednostek do której należy kolekcji. W przeciwnym razie EntitySet atrybutu nie może być używany.

NOTE

Dowolna liczba atrybutów adnotacji (niestandardowe atrybuty XML) można stosować do **ReturnType** elementu (element FunctionImport). Jednak atrybutów niestandardowych, które nie mogą należeć do przestrzeni nazw XML, który jest zarezerwowany dla CSDL. W pełni kwalifikowanej nazwy dowolne dwa atrybuty niestandardowe nie może być taka sama.

Przykład

W poniższym przykładzie użyto **FunctionImport** zwracającego książki i wydawców. Należy pamiętać, że funkcja zwraca dwa zestawy wyników i w związku z tym dwa **ReturnType** są określone elementy (FunctionImport).

```

<FunctionImport Name="GetBooksAndPublishers">
  <ReturnType Type=="Collection(BooksModel.Book )" EntitySet="Books">
  <ReturnType Type=="Collection(BooksModel.Publisher)" EntitySet="Publishers">
</FunctionImport>

```

Element RowType (CSDL)

A **RowType** element w języku definicji schematu koncepcyjnego (CSDL) definiuje strukturę nienazwane jako parametr lub zwracany typ funkcji zdefiniowanych w modelu koncepcyjnym.

A **RowType** element może być elementem podrzędnym elementu następujące elementy:

- Typ CollectionType
- Parametr
- ReturnType (funkcja)

A **RowType** element może mieć następujących elementów podrzędnych (w podanej kolejności):

- Właściwości (jeden lub więcej)
- Elementów adnotacji (zero lub więcej)

Odpowiednie atrybuty

Dowolna liczba atrybutów adnotacji (niestandardowe atrybuty XML) można stosować do **RowType** elementu. Jednak atrybutów niestandardowych, które nie mogą należeć do przestrzeni nazw XML, który jest zarezerwowany dla CSDL. W pełni kwalifikowanej nazwy dowolne dwa atrybuty niestandardowe nie może być taka sama.

Przykład

W poniższym przykładzie pokazano funkcji definiowanych przez model, który używa **CollectionType** elementu, aby określić, że funkcja zwraca Kolekcja wierszy (jak określono w **RowType** elementu).

```

<Function Name="LastNamesAfter">
  <Parameter Name="someString" Type="Edm.String" />
  <ReturnType>
    <CollectionType>
      <RowType>
        <Property Name="FirstName" Type="Edm.String" Nullable="false" />
        <Property Name="LastName" Type="Edm.String" Nullable="false" />
      </RowType>
    </CollectionType>
  </ReturnType>
  <DefiningExpression>
    SELECT VALUE ROW(p.FirstName, p.LastName)
    FROM SchoolEntities.People AS p
    WHERE p.LastName &gt;= somestring
  </DefiningExpression>
</Function>

```

Element schematu (CSDL)

Schematu element jest elementem głównym modelu koncepcyjnego definicji. Zawiera definicje dla obiektów, funkcji i kontenerów, które tworzą modelu koncepcyjnego.

Schematu element może zawierać zero lub więcej z następujących elementów podrzędnych:

- za pomocą
- Obiekt EntityContainer
- Typ entityType

- EnumType
- Skojarzenie
- ComplexType
- Funkcja

A **schematu** element może zawierać zero lub jeden elementów adnotacji.

NOTE

Funkcja element i adnotacji elementy są dozwolone tylko w wersji 2 CSDL i nowszych wersjach.

Schematu element używa **Namespace** atrybut do definiowania przestrzeni nazw dla typu jednostki, typu złożonego i skojarzenia obiekty w modelu koncepcyjnym. W przestrzeni nazw nie dwa obiekty mogą mieć takiej samej nazwy. Przestrzenie nazw może obejmować wiele **schematu** elementów i wiele plików .csdl.

Model koncepcyjny przestrzeni nazw różni się od przestrzeni nazw XML **schematu** elementu. Model koncepcyjny przestrzeni nazw (zgodnie z definicją **Namespace** atrybutu) to logiczny kontener przeznaczony dla typów jednostek, typy złożone i typy stwarzyszenie. Przestrzeń nazw XML (wskazywanym przez **xmlns** atrybutu) z **schematu** element jest domyślny obszar nazw dla elementów podległych i atrybutów **schematu** elementu. Obszary nazw XML w postaci <http://schemas.microsoft.com/ado/YYYY/MM/edm> (gdzie RRRR i MM stanowi rok i miesiąc odpowiednio) są zarezerwowane dla CSDL. Niestandardowe elementy i atrybuty nie może być w przestrzeni nazw, które mają postać.

Odpowiednie atrybuty

W poniższej tabeli opisano atrybuty mogą być stosowane do **schematu** elementu.

NAZWA ATRYBUTU	JEST WYMAGANY	WARTOŚĆ
Namespace	Tak	Przestrzeń nazw modelu koncepcyjnego. Wartość Namespace atrybut jest używany w celu utworzenia w pełni kwalifikowana nazwa typu. Na przykład jeśli EntityType o nazwie <i>klienta</i> znajduje się w przestrzeni nazw Simple.Example.Model, a następnie w pełni kwalifikowana nazwa EntityType jest SimpleExampleModel.Customer. Nie można użyć następujących ciągów jako wartość pozycji Namespace atrybut: systemu, przejściowy, lub Edm . Wartość Namespace atrybut nie może być taka sama jak wartość Namespace atrybutu w elemencie schematu SSDL.
Alias	Nie	Identyfikator używany zamiast nazwy przestrzeni nazw. Na przykład jeśli EntityType o nazwie <i>klienta</i> znajduje się w przestrzeni nazw Simple.Example.Model i wartość Alias atrybut jest <i>modelu</i> , wówczas można użyć Model.Customer jako w pełni kwalifikowana nazwa typu EntityType .

NOTE

Dowolna liczba atrybutów adnotacji (niestandardowe atrybuty XML) można stosować do **schematu** elementu. Jednak atrybutów niestandardowych, które nie mogą należeć do przestrzeni nazw XML, który jest zarezerwowany dla CSDL. W pełni kwalifikowanej nazwy dowolne dwa atrybuty niestandardowe nie może być taka sama.

Przykład

W poniższym przykładzie przedstawiono **schematu** element, który zawiera **EntityContainer** element, dwa **EntityType** elementów, a drugi **skojarzenia** elementu.

```

<Schema xmlns="http://schemas.microsoft.com/ado/2009/11/edm"
    xmlns:cg="http://schemas.microsoft.com/ado/2009/11/codegeneration"
    xmlns:store="http://schemas.microsoft.com/ado/2009/11/edm/EntityStoreSchemaGenerator"
    Namespace="ExampleModel" Alias="Self">
    <EntityTypeContainer Name="ExampleModelContainer">
        <EntityTypeSet Name="Customers"
            EntityType="ExampleModel.Customer" />
        <EntityTypeSet Name="Orders" EntityType="ExampleModel.Order" />
        <AssociationSet
            Name="CustomerOrder"
            Association="ExampleModel.CustomerOrders">
            <End Role="Customer" EntitySet="Customers" />
            <End Role="Order" EntitySet="Orders" />
        </AssociationSet>
    </EntityTypeContainer>
    <EntityType Name="Customer">
        <Key>
            <PropertyRef Name="CustomerId" />
        </Key>
        <Property Type="Int32" Name="CustomerId" Nullable="false" />
        <Property Type="String" Name="Name" Nullable="false" />
        <NavigationProperty
            Name="Orders"
            Relationship="ExampleModel.CustomerOrders"
            FromRole="Customer" ToRole="Order" />
    </EntityType>
    <EntityType Name="Order">
        <Key>
            <PropertyRef Name="OrderId" />
        </Key>
        <Property Type="Int32" Name="OrderId" Nullable="false" />
        <Property Type="Int32" Name="ProductId" Nullable="false" />
        <Property Type="Int32" Name="Quantity" Nullable="false" />
        <NavigationProperty
            Name="Customer"
            Relationship="ExampleModel.CustomerOrders"
            FromRole="Order" ToRole="Customer" />
        <Property Type="Int32" Name="CustomerId" Nullable="false" />
    </EntityType>
    <Association Name="CustomerOrders">
        <End Type="ExampleModel.Customer"
            Role="Customer" Multiplicity="1" />
        <End Type="ExampleModel.Order"
            Role="Order" Multiplicity="*" />
        <ReferentialConstraint>
            <Principal Role="Customer">
                <PropertyRef Name="CustomerId" />
            </Principal>
            <Dependent Role="Order">
                <PropertyRef Name="CustomerId" />
            </Dependent>
        </ReferentialConstraint>
    </Association>
</Schema>

```

Element TypeRef (CSDL)

TypeRef element język definicji schematu koncepcyjnego (CSDL) zawiera odwołanie do istniejącego typu nazwanego. **TypeRef** element może być elementem podrzędnym elementu `CollectionType`, który jest używany do określenia, czy funkcja jest kolekcją jako parametr lub zwracany typ.

A **TypeRef** element może mieć następujących elementów podrzędnych (w podanej kolejności):

- Dokumentacja (zero lub jeden element)

- Elementów adnotacji (zero lub więcej elementów)

Odpowiednie atrybuty

W poniższej tabeli opisano atrybuty, które mogą być stosowane do **TypeRef** elementu. Należy pamiętać, że **DefaultValue**, **MaxLength**, **FixedLength**, **dokładności**, **skalowania**, **Unicode**, i **sortowania** atrybuty dotyczą tylko **EDMSimpleTypes**.

NAZWA ATRYBUTU	JEST WYMAGANY	WARTOŚĆ
Typ	Nie	Nazwa typu, do którego nastąpiło odwołanie.
Dopuszcza wartości null	Nie	Wartość true , (wartość domyślna) lub False w zależności od tego, czy właściwość może mieć wartości null. [!NOTE]
> W wersji 1 CSDL musi mieć właściwość typu złożonego <code>Nullable="False"</code> .		
defaultValue	Nie	Wartość domyślna właściwości.
Element maxLength	Nie	Maksymalna długość wartości właściwości.
Wartości	Nie	Wartość true , lub False w zależności od tego, czy przechowywana wartość właściwości jako ciąg znaków o stałej długości.
Precyzja	Nie	Dokładność wartości właściwości.
Skala	Nie	Skala wartości właściwości.
SRID	Nie	Identyfikator odwołania przestrzennego systemu. Prawidłowy tylko w przypadku właściwości typów przestrzennych. Aby uzyskać więcej informacji, zobacz SRID i SRID (SQL Server) .
Unicode	Nie	Wartość true , lub False w zależności od tego, czy przechowywana wartość właściwości jako ciąg Unicode.
Sortowanie	Nie	Ciąg, który określa kolejność sortowania, które ma być używany w źródle danych.

NOTE

Dowolna liczba atrybutów adnotacji (niestandardowe atrybuty XML) można stosować do **CollectionType** elementu. Jednak atrybutów niestandardowych, które nie mogą należeć do przestrzeni nazw XML, który jest zarezerwowany dla CSDL. W pełni kwalifikowanej nazwy dowolne dwa atrybuty niestandardowe nie może być taka sama.

Przykład

W poniższym przykładzie pokazano funkcji definiowanych przez model, który używa **TypeRef** — element (jako element podrzędny elementu **CollectionType** elementu) do określenia, że funkcja ta akceptuje zbiór **Dział** typów jednostek.

```
<Function Name="GetAvgBudget">
  <Parameter Name="Departments">
    <CollectionType>
      <TypeRef Type="SchoolModel.Department"/>
    </CollectionType>
  </Parameter>
  <ReturnType Type="Collection(Edm.Decimal)" />
  <DefiningExpression>
    SELECT VALUE AVG(d.Budget) FROM Departments AS d
  </DefiningExpression>
</Function>
```

Za pomocą elementu (CSDL)

Using element język definicji schematu koncepcyjnego (CSDL) importuje zawartość modelu koncepcyjnego, który znajduje się w innej przestrzeni nazw. Ustawiając wartość **Namespace** atrybut, można się odwoływać do typów jednostek, typy złożone i typy skojarzenia, które są zdefiniowane w modelu koncepcyjnym innego. Więcej niż jeden **Using** element może być elementem podrzędnym **schematu** elementu.

NOTE

Using element CSDL nie działa dokładnie tak jak **przy użyciu** instrukcji w języku programowania. Importując przestrzeń nazw z **przy użyciu** instrukcji w języku programowania, możesz nie wpływać na obiekty w oryginalnym przestrzeni nazw. W CSDL importowanych przestrzeni nazw może zawierać typ jednostki, który pochodzi od typu jednostki w oryginalnym przestrzeni nazw. Może to wpływać na zestawy jednostek, zadeklarowany w oryginalnym przestrzeni nazw.

Using element może mieć następujące elementy podrzędne:

- Dokumentacja (zero lub jeden elementy dozwolone)
- Elementów adnotacji (zero lub więcej elementów dozwolone)

Odpowiednie atrybuty

W poniższej tabeli opisano atrybuty mogą być stosowane do **Using** elementu.

NAZWA ATRYBUTU	JEST WYMAGANY	WARTOŚĆ
Namespace	Tak	Nazwa importowanych przestrzeni nazw.
Alias	Tak	Identyfikator używany zamiast nazwy przestrzeni nazw. Mimo że ten atrybut jest wymagany, nie jest wymagane, aby z niego korzystać zamiast nazwy przestrzeni nazw kwalifikowania nazwy obiektów.

NOTE

Dowolna liczba atrybutów adnotacji (niestandardowe atrybuty XML) można stosować do **Using** elementu. Jednak atrybutów niestandardowych, które nie mogą należeć do przestrzeni nazw XML, który jest zarezerwowany dla CSDL. W pełni kwalifikowanej nazwy dowolne dwa atrybuty niestandardowe nie może być taka sama.

Przykład

W poniższym przykładzie pokazano **Using** elementu używane do importowania przestrzeni nazw, która jest definiowane w innym miejscu. Należy pamiętać, że przestrzeń nazw dla **schematu** elementu wyświetlna jest **BooksModel** . **Address** Właściwość **Publisher** **EntityType** to typ złożony, który jest zdefiniowany w **ExtendedBooksModel** przestrzeni nazw (zimportowane wraz z **Using** elementu).

```
<Schema xmlns="http://schemas.microsoft.com/ado/2009/11/edm"
    xmlns:cg="http://schemas.microsoft.com/ado/2009/11/codegeneration"
    xmlns:store="http://schemas.microsoft.com/ado/2009/11/edm/EntityStoreSchemaGenerator"
    Namespace="BooksModel" Alias="Self">

    <Using Namespace="BooksModel.Extended" Alias="BMEExt" />

    <EntityContainer Name="BooksContainer" >
        <EntitySet Name="Publishers" EntityType="BooksModel.Publisher" />
    </EntityContainer>

    <EntityType Name="Publisher">
        <Key>
            <PropertyRef Name="Id" />
        </Key>
        <Property Type="Int32" Name="Id" Nullable="false" />
        <Property Type="String" Name="Name" Nullable="false" />
        <Property Type="BMEExt.Address" Name="Address" Nullable="false" />
    </EntityType>

</Schema>
```

Atrybuty adnotacji (CSDL)

Atrybuty adnotacji w język definicji schematu koncepcyjnego (CSDL) są niestandardowe atrybuty XML w modelu koncepcyjnym. Oprócz prawidłowej struktury XML, wymaga spełnienia następujących warunków adnotacji atrybutów:

- Atrybuty adnotacji nie może być w przestrzeni nazw XML, który jest zarezerwowany dla CSDL.
- Więcej niż jeden atrybut adnotacji można stosować do danego elementu CSDL.
- W pełni kwalifikowanej nazwy wszelkie atrybuty dwóch adnotacji nie może być taka sama.

Atrybuty adnotacji może służyć do zapewnienia dodatkowe metadane na temat elementów w modelu koncepcyjnym. W czasie wykonywania przy użyciu klas w przestrzeni nazw System.Data.Metadata.Edm możliwy jest metadanych elementów adnotacji.

Przykład

W poniższym przykładzie przedstawiono **EntityType** element z atrybutem adnotacji (**CustomAttribute —**). W przykładzie pokazano również element adnotacji stosowane do elementu typu jednostki.

```

<Schema Namespace="SchoolModel" Alias="Self"
    xmlns:annotation="http://schemas.microsoft.com/ado/2009/02/edm/annotation"
    xmlns="http://schemas.microsoft.com/ado/2009/11/edm">
    <EntityContainer Name="SchoolEntities" annotation:LazyLoadingEnabled="true">
        <EntityType Name="Person" EntityType="SchoolModel.Person" />
    </EntityContainer>
    <EntityType Name="Person" xmlns:p="http://CustomNamespace.com"
        p:CustomAttribute="Data here.">
        <Key>
            <PropertyRef Name="PersonID" />
        </Key>
        <Property Name="PersonID" Type="Int32" Nullable="false"
            annotation:StoreGeneratedPattern="Identity" />
        <Property Name="LastName" Type="String" Nullable="false"
            MaxLength="50" Unicode="true" FixedLength="false" />
        <Property Name="FirstName" Type="String" Nullable="false"
            MaxLength="50" Unicode="true" FixedLength="false" />
        <Property Name="HireDate" Type="DateTime" />
        <Property Name="EnrollmentDate" Type="DateTime" />
        <p:CustomElement>
            Custom metadata.
        </p:CustomElement>
    </EntityType>
</Schema>

```

Poniższy kod służy do pobierania metadanych w atrybucie adnotacji i zapisuje go do konsoli:

```

EdmItemCollection collection = new EdmItemCollection("School.csdl");
MetadataWorkspace workspace = new MetadataWorkspace();
workspace.RegisterItemCollection(collection);
EdmType contentType;
workspace.TryGetType("Person", "SchoolModel", DataSpace.CSpace, out contentType);
if (contentType.MetadataProperties.Contains("http://CustomNamespace.com:CustomAttribute"))
{
    MetadataProperty annotationProperty =
        contentType.MetadataProperties["http://CustomNamespace.com:CustomAttribute"];
    object annotationValue = annotationProperty.Value;
    Console.WriteLine(annotationValue.ToString());
}

```

Powyższy kod zakłada, że `School.csdl` plik znajduje się w katalogu wyjściowego projektu i dodano następujące `Imports` i `Using` instrukcje do projektu:

```
using System.Data.Metadata.Edm;
```

Elementów adnotacji (CSDL)

Elementy adnotacji w język definicji schematu koncepcyjnego (CSDL) są niestandardowe elementy XML w modelu koncepcyjnym. Oprócz prawidłowej struktury XML, wymaga spełnienia następujących warunków elementów adnotacji:

- Elementów adnotacji nie może być w przestrzeni nazw XML, który jest zarezerwowany dla CSDL.
- Więcej niż jeden element adnotacji może być elementem podrzędnym danego elementu CSDL.
- W pełni kwalifikowanej nazwy dowolne elementy dwóch adnotacji nie może być taka sama.
- Adnotacja elementów musi występować po wszystkich innych elementów podrzędnych danego elementu CSDL.

Elementów adnotacji może służyć do zapewnienia dodatkowe metadane na temat elementów w modelu

koncepcyjnym. Począwszy od programu .NET Framework w wersji 4, metadanych elementów adnotacji są dostępne w czasie wykonywania przy użyciu klas w przestrzeni nazw System.Data.Metadata.Edm.

Przykład

W poniższym przykładzie przedstawiono **EntityType** element z element adnotacji (**CustomElement**). Przykład pokazują również adnotacji zastosowany do elementu typu jednostki.

```
<Schema Namespace="SchoolModel" Alias="Self"
    xmlns:annotation="http://schemas.microsoft.com/ado/2009/02/edm/annotation"
    xmlns="http://schemas.microsoft.com/ado/2009/11/edm">
    <EntityContainer Name="SchoolEntities" annotation:LazyLoadingEnabled="true">
        <EntitySet Name="People" EntityType="SchoolModel.Person" />
    </EntityContainer>
    <EntityType Name="Person" xmlns:p="http://CustomNamespace.com"
        p:CustomAttribute="Data here.">
        <Key>
            <PropertyRef Name="PersonID" />
        </Key>
        <Property Name="PersonID" Type="Int32" Nullable="false"
            annotation:StoreGeneratedPattern="Identity" />
        <Property Name="LastName" Type="String" Nullable="false"
            MaxLength="50" Unicode="true" FixedLength="false" />
        <Property Name="FirstName" Type="String" Nullable="false"
            MaxLength="50" Unicode="true" FixedLength="false" />
        <Property Name="HireDate" Type="DateTime" />
        <Property Name="EnrollmentDate" Type="DateTime" />
        <p:CustomElement>
            Custom metadata.
        </p:CustomElement>
    </EntityType>
</Schema>
```

Poniższy kod służy do pobierania metadanych w elemencie adnotacji i zapisuje go do konsoli:

```
EdmItemCollection collection = new EdmItemCollection("School.csdl");
MetadataWorkspace workspace = new MetadataWorkspace();
workspace.RegisterItemCollection(collection);
EdmType contentType;
workspace.TryGetType("Person", "SchoolModel", DataSpace.CSpace, out contentType);
if (contentType.MetadataProperties.Contains("http://CustomNamespace.com:CustomElement"))
{
    MetadataProperty annotationProperty =
        contentType.MetadataProperties["http://CustomNamespace.com:CustomElement"];
    object annotationValue = annotationProperty.Value;
    Console.WriteLine(annotationValue.ToString());
}
```

Powyższy kod założono, że plik School.csdl znajduje się w katalogu wyjściowego projektu, i dodano następujące

`Imports` i `Using` instrukcje do projektu:

```
using System.Data.Metadata.Edm;
```

Model koncepcyjny typów (CSDL)

Język definicji schematu koncepcyjnego (CSDL) obsługuje zestaw abstrakcyjne pierwotne typy danych, nazywane **EDMSimpleTypes**, który definiują właściwości w modelu koncepcyjnym. **EDMSimpleTypes** serwerów proxy dla typów danych pierwotnych, które są obsługiwane w środowisku hostingu lub magazynu.

Poniższa tabela zawiera listę typów danych pierwotnych, które są obsługiwane przez CSDL. W tabeli

przedstawiono listę zestawów reguł, które mogą być stosowane do każdego **EDMSimpleType**.

EDMSIMPLETYPE	OPIS	ZASTOSOWANIE ZESTAWÓW REGUŁ
Edm.Binary	Zawiera dane binarne.	Element MaxLength, wartości null, domyślne
Typem Edm.Boolean	Zawiera wartość true lub false .	Wartość null, domyślne
Edm.Byte	Zawiera wartość Liczba całkowita bez znaku 8-bitowych.	Precyzja dopuszczającego wartość null, domyślny
Edm.DateTime	Reprezentuje datę i godzinę.	Precyzja dopuszczającego wartość null, domyślny
Edm.DateTimeOffset	Zawiera datę i godzinę w ciągu kilku minut od GMT przesunięcia.	Precyzja dopuszczającego wartość null, domyślny
Edm.Decimal	Zawiera wartość liczbową ze stałym dokładnością i skali.	Precyzja dopuszczającego wartość null, domyślny
Edm.Double	Zawiera zmiennoprzecinkowa numer z dokładnością do 15 cyfr	Precyzja dopuszczającego wartość null, domyślny
Edm.Float	Zawiera zmiennoprzecinkowej liczba z 7-cyfrowy dokładnością.	Precyzja dopuszczającego wartość null, domyślny
Edm.Guid	Zawiera unikatowy identyfikator 16-bajtowy.	Precyzja dopuszczającego wartość null, domyślny
Edm.Int16	Zawiera wartość liczby całkowitej ze znakiem 16-bitowych.	Precyzja dopuszczającego wartość null, domyślny
Typem Edm.Int32	Zawiera wartość całkowita 32-bitowa.	Precyzja dopuszczającego wartość null, domyślny
Edm.Int64	Zawiera wartość całkowita 64-bitowa.	Precyzja dopuszczającego wartość null, domyślny
Edm.SByte	Zawiera wartość całkowita 8-bitowa.	Precyzja dopuszczającego wartość null, domyślny
Edm.String	Zawiera dane znaków.	Unicode dla wpisu, MaxLength, sortowanie, dokładności, dopuszczającego wartość null, domyślne
Edm.Time	Zawiera porze dnia.	Precyzja dopuszczającego wartość null, domyślny
Edm.Geography		Wartość null, domyślnie, SRID
Edm.GeographyPoint		Wartość null, domyślnie, SRID
Edm.GeographyLineString		Wartość null, domyślnie, SRID

EDMSIMPLETYPE	OPIS	ZASTOSOWANIE ZESTAWÓW REGUŁ
Edm.GeographyPolygon		Wartość null, domyślnie, SRID
Edm.GeographyMultiPoint		Wartość null, domyślnie, SRID
Edm.GeographyMultiLineString		Wartość null, domyślnie, SRID
Edm.GeographyMultiPolygon		Wartość null, domyślnie, SRID
Edm.GeographyCollection		Wartość null, domyślnie, SRID
Edm.Geometry		Wartość null, domyślnie, SRID
Edm.GeometryPoint		Wartość null, domyślnie, SRID
Edm.GeometryLineString		Wartość null, domyślnie, SRID
Edm.GeometryPolygon		Wartość null, domyślnie, SRID
Edm.GeometryMultiPoint		Wartość null, domyślnie, SRID
Edm.GeometryMultiLineString		Wartość null, domyślnie, SRID
Edm.GeometryMultiPolygon		Wartość null, domyślnie, SRID
Edm.GeometryCollection		Wartość null, domyślnie, SRID

Zestawy reguł (CSDL)

Aspekty w języku definicji schematu koncepcyjnego (CSDL) reprezentują ograniczenia dotyczące właściwości typów jednostek i typów złożonych. Zestawy reguł są wyświetlane jako atrybuty XML w następujących elementów CSDL:

- Właściwość
- TypeRef
- Parametr

W poniższej tabeli opisano aspekty, które są obsługiwane przez CSDL. Wszystkie zestawy reguł są opcjonalne. Niektóre aspekty wymienione poniżej są używane przez program Entity Framework, podczas generowania bazy danych na podstawie modelu koncepcyjnego.

NOTE

Informacje o typach danych w modelu koncepcyjnym na ten temat można znaleźć w koncepcyjny modelu typy (CSDL).

ZESTAW REGUŁ	OPIS	INFORMACJE ZAWARTE W TYM ARTYKULE DOTYCZĄ	UŻYTO DO GENEROWANIA BAZY DANYCH	UŻYWANE PRZEZ ŚRODOWISKO URUCHOMIENIOWE
--------------	------	---	--	---

ZESTAW REGUŁ	OPIS	INFORMACJE ZAWARTE W TYM ARTYKULE DOTYCZĄ	UŻYTO DO GENEROWANIA BAZY DANYCH	UŻYWANE PRZEZ ŚRODOWISKO URUCHOMIENIOWE
Sortowanie	Okręsła kolejność sortowania (lub sekwencji sortowania) do użycia podczas przeprowadzania porównania i kolejność operacji na wartościach właściwości.	Edm.String	Tak	Nie
Właściwość ConcurrencyMode	Wskazuje, że wartość właściwości powinien być używany w celu pomyślnych kontroli współbieżności.	Wszystkie EDMSimpleType właściwości	Nie	Tak
Default	Okręsła domyślną wartość właściwości, jeśli nie dostarczono żadnej wartości dla wystąpienia.	Wszystkie EDMSimpleType właściwości	Tak	Tak
Wartości	Okręsła, czy długość wartości właściwości mogą się różnić.	Edm.Binary , Edm.String	Tak	Nie
Element maxLength	Okręsła maksymalną długość wartości właściwości.	Edm.Binary , Edm.String	Tak	Nie
Dopuszcza wartości null	Okręsła, czy właściwość może mieć null wartość.	Wszystkie EDMSimpleType właściwości	Tak	Tak
Precyzja	Dla właściwości typu dziesiętna , określa liczbę cyfr, może mieć wartości właściwości. Dla właściwości typu czasu , daty/godziny , i DateTimeOffset , określa liczbę cyfr ułamkowych części sekundy w wartości właściwości.	Edm.DateTime , Edm.DateTimeOffset , Edm.Decimal , Edm.Time	Tak	Nie
Skala	Okręsła liczbę cyfr po prawej stronie przecinka dziesiętnego dla wartości właściwości.	Edm.Decimal	Tak	Nie

ZESTAW REGUŁ	OPIS	INFORMACJE ZAWARTE W TYM ARTYKULE DOTYCZĄ	UŻYTO DO GENEROWANIA BAZY DANYCH	UŻYWANE PRZEZ ŚRODOWISKO URUCHOMIENIOWE
SRID	Okręsła identyfikator System przestrzenne odwołanie do systemu. Aby uzyskać więcej informacji, zobacz SRID i SRID (SQL Server) .	Edm.Geography Edm.GeographyPoint, Edm.GeographyLineString, Edm.GeographyPolygon, Edm.GeographyMultiPoint, Edm.GeographyMultiLineString, Edm.GeographyMultiPolygon, Edm.GeographyCollection, Edm.Geometry, Edm.GeometryPoint ,	Nie	Tak
Unicode	Wskazuje, czy wartość właściwości jest przechowywana jako Unicode.	Edm.String	Tak	Tak

NOTE

Podczas generowania bazę danych z modelu koncepcyjnego, Kreator bazy danych Generowanie rozpoznaje wartość parametru **StoreGeneratedPattern** atrybutu na właściwość elementu, jeśli znajduje się w następujących przestrzeni nazw: <http://schemas.microsoft.com/ado/2009/02/edm/annotation>. Obsługiwane wartości dla atrybutu to **tożsamości** i **obliczane**. Wartość **tożsamości** dadzą kolumny bazy danych przy użyciu wartości tożsamości, który jest generowany w bazie danych. Wartość **obliczane** dadzą kolumna z wartością, która jest kolumną obliczaną w bazie danych.

Przykład

Poniższy przykład przedstawia aspektami stosowany do właściwości typu jednostki:

```
<EntityType Name="Product">
  <Key>
    <PropertyRef Name="ProductId" />
  </Key>
  <Property Type="Int32"
    Name="ProductId" Nullable="false"
    a:StoreGeneratedPattern="Identity"
    xmlns:a="http://schemas.microsoft.com/ado/2009/02/edm/annotation" />
  <Property Type="String"
    Name="ProductName"
    Nullable="false"
    MaxLength="50" />
  <Property Type="String"
    Name="Location"
    Nullable="true"
    MaxLength="25" />
</EntityType>
```

Specyfikacja MSL

02.10.2018 • 68 minutes to read • [Edit Online](#)

Mapowanie specyfikacji języka (MSL) to język oparty na formacie XML, który opisuje mapowanie między modelem koncepcyjnym i modelem magazynu aplikacji platformy Entity Framework.

W aplikacji Entity Framework mapowania metadanych jest ładowany z pliku MSL albo identyfikatorem (zapisany w pliku MSL) w czasie komplikacji. Entity Framework używa mapowania metadanych w czasie wykonywania do przekształcania zapytania względem modelu koncepcyjnego polecenia specyficzne dla magazynu.

Entity Framework Designer (Projektant EF) przechowuje informacje dotyczące mapowania w pliku edmx w czasie projektowania. W czasie komplikacji Projektant jednostki używa informacji w pliku edmx można utworzyć pliku MSL albo identyfikatorem, które są wymagane przez Entity Framework w czasie wykonywania

Nazwy wszystkich koncepcji lub typów modelu magazynu, które są określone w pliku MSL musi być kwalifikowana przy użyciu ich nazw odpowiednich przestrzeni nazw. Aby uzyskać informacji na temat modelu koncepcyjnego nazwa przestrzeni nazw, zobacz [Specyfikacja CSDL](#). Aby dowiedzieć się, nazwa przestrzeni nazw w modelu magazynu, zobacz [Specyfikacja SSDL](#).

Wersje MSL są zróżnicowane według przestrzeni nazw XML.

WERSJA PLIKU MSL	NAMESPACE XML
MSL v1	Nazwa urn: schemas-microsoft-com:windows:storage:mapping:CS
MSL v2	http://schemas.microsoft.com/ado/2008/09/mapping/cs
MSL v3	http://schemas.microsoft.com/ado/2009/11/mapping/cs

W elemencie aliasu (MSL)

Alias element w mapowaniu language specification (MSL) jest elementem podrzędnym elementu mapowania, który jest używany do definiowania aliasy koncepcyjny model i magazynu modeli w przypadku przestrzeni nazw. Nazwy wszystkich koncepcji lub typów modelu magazynu, które są określone w pliku MSL musi być kwalifikowana przy użyciu ich nazw odpowiednich przestrzeni nazw. Dla informacji o modelu koncepcyjnego nazwa przestrzeni nazw Zobacz Element schematu (CSDL). Informacje o nazwie przestrzeni nazw w modelu magazynu na ten temat można znaleźć w elemencie schematu (SSDL).

Alias elementu nie może mieć elementów podrzędnych.

Odpowiednie atrybuty

W poniższej tabeli opisano atrybuty, które mogą być stosowane do **Alias** elementu.

NAZWA ATRYBUTU	JEST WYMAGANY	WARTOŚĆ
Key	Tak	Alias przestrzeni nazw, który jest określony przez wartość atrybutu.
Wartość	Tak	Przestrzeń nazw, dla których wartość klucz element jest aliasem.

Przykład

W poniższym przykładzie przedstawiono **Alias** element, który definiuje alias, `c`, dla których typy zostały zdefiniowane w modelu koncepcyjnym.

```
<Mapping Space="C-S"
    xmlns="http://schemas.microsoft.com/ado/2009/11/mapping/cs">
    <Alias Key="c" Value="SchoolModel"/>
    <EntityContainerMapping StorageEntityContainer="SchoolModelStoreContainer"
        CdmEntityContainer="SchoolModelEntities">
        <EntityTypeMapping TypeName="c.Course">
            <MappingFragment StoreEntitySet="Course">
                <ScalarProperty Name="CourseID" ColumnName="CourseID" />
                <ScalarProperty Name="Title" ColumnName="Title" />
                <ScalarProperty Name="Credits" ColumnName="Credits" />
                <ScalarProperty Name="DepartmentID" ColumnName="DepartmentID" />
            </MappingFragment>
        </EntityTypeMapping>
    </EntitySetMapping>
    <EntitySetMapping Name="Departments">
        <EntityTypeMapping TypeName="c.Department">
            <MappingFragment StoreEntitySet="Department">
                <ScalarProperty Name="DepartmentID" ColumnName="DepartmentID" />
                <ScalarProperty Name="Name" ColumnName="Name" />
                <ScalarProperty Name="Budget" ColumnName="Budget" />
                <ScalarProperty Name="StartDate" ColumnName="StartDate" />
                <ScalarProperty Name="Administrator" ColumnName="Administrator" />
            </MappingFragment>
        </EntityTypeMapping>
    </EntitySetMapping>
    </EntityContainerMapping>
</Mapping>
```

Punkt końcowy skojarzenia elementu (MSL)

Punkt końcowy skojarzenia element w mapowaniu language specification (MSL) jest używany, gdy funkcji modyfikacji typu jednostki w modelu koncepcyjnym są mapowane do procedur składowanych w bazie danych. Jeśli modyfikacja procedury składowanej przyjmuje parametr, którego wartość jest przechowywana w właściwość skojarzenia **punkt końcowy skojarzenia** element mapuje wartości właściwości do parametru. Aby uzyskać więcej informacji zobacz przykład poniżej.

Aby uzyskać więcej informacji na temat mapowania funkcji modyfikacji typów jednostek do procedur składowanych, zobacz Element ModificationFunctionMapping (MSL) i wskazówki: mapowanie jednostki do procedur składowanych.

Punkt końcowy skojarzenia element może mieć następujące elementy podrzędne:

- ScalarProperty

Odpowiednie atrybuty

W poniższej tabeli opisano atrybuty, które mają zastosowanie do **punkt końcowy skojarzenia** elementu.

NAZWA ATRYBUTU	JEST WYMAGANY	WARTOŚĆ
Obiekt AssociationSet	Tak	Nazwa skojarzenia, który jest mapowany.

NAZWA ATRYBUTU	JEST WYMAGANY	WARTOŚĆ
From	Tak	Wartość FromRole atrybut właściwość nawigacji, która odnosi się do skojarzenia mapowanego. Aby uzyskać więcej informacji zobacz element NavigationProperty — Element (CSDL) .
To	Tak	Wartość ToRole atrybut właściwość nawigacji, która odnosi się do skojarzenia mapowanego. Aby uzyskać więcej informacji zobacz element NavigationProperty — Element (CSDL) .

Przykład

Należy wziąć pod uwagę następujące typy modelu koncepcyjnego entity:

```
<EntityType Name="Course">
  <Key>
    <PropertyRef Name="CourseID" />
  </Key>
  <Property Type="Int32" Name="CourseID" Nullable="false" />
  <Property Type="String" Name="Title" Nullable="false" MaxLength="100"
            FixedLength="false" Unicode="true" />
  <Property Type="Int32" Name="Credits" Nullable="false" />
  <NavigationProperty Name="Department"
    Relationship="SchoolModel.FK_Course_Department"
    FromRole="Course" ToRole="Department" />
</EntityType>
```

Należy również rozważyć następującą procedurę składowaną:

```
CREATE PROCEDURE [dbo].[UpdateCourse]
    @CourseID int,
    @Title nvarchar(50),
    @Credits int,
    @DepartmentID int
    AS
        UPDATE Course SET Title=@Title,
                          Credits=@Credits,
                          DepartmentID=@DepartmentID
        WHERE CourseID=@CourseID;
```

Aby zamapować funkcję aktualizacji `Course` jednostki do tej procedury składowanej, należy podać wartość **DepartmentID** parametru. Wartość `DepartmentID` nie odpowiada na właściwość typu jednostki; znajduje się w skojarzeniu niezależnie od którego mapowanie jest następująca:

```
<AssociationSetMapping Name="FK_Course_Department"
    TypeName="SchoolModel.FK_Course_Department"
    StoreEntitySet="Course">
  <EndProperty Name="Course">
    <ScalarProperty Name="CourseID" ColumnName="CourseID" />
  </EndProperty>
  <EndProperty Name="Department">
    <ScalarProperty Name="DepartmentID" ColumnName="DepartmentID" />
  </EndProperty>
</AssociationSetMapping>
```

Poniższy kod przedstawia **punkt końcowy skojarzenia** element używany do mapowania **DepartmentID**

właściwość **klucza Obcego_kurs_działu** skojarzenie, aby **UpdateCourse** procedury składowanej (do którego funkcja aktualizacji **kurs** typ jednostki jest mapowany):

```
<EntitySetMapping Name="Courses">
    <EntityTypeMapping TypeName="SchoolModel.Course">
        <MappingFragment StoreEntitySet="Course">
            <ScalarProperty Name="Credits" ColumnName="Credits" />
            <ScalarProperty Name="Title" ColumnName="Title" />
            <ScalarProperty Name="CourseID" ColumnName="CourseID" />
        </MappingFragment>
    </EntityTypeMapping>
    <EntityTypeMapping TypeName="SchoolModel.Course">
        <ModificationFunctionMapping>
            <UpdateFunction FunctionName="SchoolModel.Store.UpdateCourse">
                <AssociationEnd AssociationSet="FK_Course_Department"
                    From="Course" To="Department">
                    <ScalarProperty Name="DepartmentID"
                        ParameterName="DepartmentID"
                        Version="Current" />
                </AssociationEnd>
                <ScalarProperty Name="Credits" ParameterName="Credits"
                    Version="Current" />
                <ScalarProperty Name="Title" ParameterName="Title"
                    Version="Current" />
                <ScalarProperty Name="CourseID" ParameterName="CourseID"
                    Version="Current" />
            </UpdateFunction>
        </ModificationFunctionMapping>
    </EntityTypeMapping>
</EntitySetMapping>
```

Element AssociationSetMapping (MSL)

AssociationSetMapping elementu w mapowaniu language specification (MSL) definiuje mapowanie między skojarzenia koncepcyjny kolumn w modelu i tabeli w bazie danych.

Skojarzeń w modelu koncepcyjnym są typy, których właściwości reprezentują podstawowe i obce kolumny klucza w bazie danych. **AssociationSetMapping** element używa dwóch elementów właściwości EndProperty, aby zdefiniować mapowania między właściwościami typu skojarzenia i kolumn w bazie danych. Warunki można umieścić na te mapowania z elementem warunku. Mapowanie insert, update i delete funkcji w celu skojarzenia do procedur składowanych w bazie danych za pomocą elementu ModificationFunctionMapping. Za pomocą ciągu jednostki SQL w elemencie QueryView, należy zdefiniować tylko do odczytu mapowania między skojarzenia i kolumn w tabeli.

NOTE

Jeśli nie zdefiniowano ograniczenie referencyjne dla skojarzenia w modelu koncepcyjnym, skojarzenie nie muszą być mapowane z **AssociationSetMapping** elementu. Jeśli **AssociationSetMapping** element jest obecny dla skojarzenia, która ma ograniczenie referencyjne, mapowania zdefiniowane w **AssociationSetMapping** element zostaną zignorowane. Aby uzyskać więcej informacji zobacz Element ReferentialConstraint (CSDL).

AssociationSetMapping element może mieć następujące elementy podrzędne

- Obiekt QueryView (zero lub jeden)
- Właściwości EndProperty (zero lub dwóch)
- Warunek (zero lub więcej)
- ModificationFunctionMapping (zero lub jeden)

Odpowiednie atrybuty

W poniższej tabeli opisano atrybuty, które mogą być stosowane do **AssociationSetMapping** elementu.

NAZWA ATRYBUTU	JEST WYMAGANY	WARTOŚĆ
Nazwa	Tak	Nazwa zestawu skojarzeń modelu koncepcyjnego, który jest mapowany.
Nazwa typu	Nie	Przestrzeń nazw kwalifikowana nazwa typu skojarzenia modelu koncepcyjnego, który jest mapowany.
StoreEntitySet	Nie	Nazwa tabeli, która jest mapowana.

Przykład

W poniższym przykładzie przedstawiono **AssociationSetMapping** elementu, w którym **klucza Obcego_kurs_działu** skojarzeń w modelu koncepcyjnym jest mapowany na **Kurs** tabeli w bazie danych. Mapowania między właściwościami typu skojarzenia i kolumn w tabeli są określone w podzajęcej **właściwości EndProperty** elementów.

```
<AssociationSetMapping Name="FK_Course_Department"
    TypeName="SchoolModel.FK_Course_Department"
    StoreEntitySet="Course">
    <EndProperty Name="Department">
        <ScalarProperty Name="DepartmentID" ColumnName="DepartmentID" />
    </EndProperty>
    <EndProperty Name="Course">
        <ScalarProperty Name="CourseID" ColumnName="CourseID" />
    </EndProperty>
</AssociationSetMapping>
```

Element ComplexProperty (MSL)

A **ComplexProperty** element w mapowaniu language specification (MSL) definiuje mapowanie między właściwość typu złożonego w modelu koncepcyjnym entity kolumny Typ i tabeli w bazie danych. Mapowania kolumn właściwości są określone w elementy ScalarProperty podrzędne.

ComplexType property — element może mieć następujące elementy podrzędne:

- ScalarProperty (zero lub więcej)
- **ComplexProperty** (zero lub więcej)
- ComplexTypeMapping (zero lub więcej)
- Warunek (zero lub więcej)

Odpowiednie atrybuty

W poniższej tabeli opisano atrybuty, które mają zastosowanie do **ComplexProperty** elementu:

NAZWA ATRYBUTU	JEST WYMAGANY	WARTOŚĆ
Nazwa	Tak	Nazwa właściwości złożonej typu jednostki w modelu koncepcyjnym, który jest mapowany.
Nazwa typu	Nie	Przestrzeń nazw kwalifikowana nazwa typu właściwości modelu koncepcyjnego.

Przykład

Poniższy przykład jest oparty na modelu szkoły. Następujący typ złożony został dodany do modelu koncepcyjnego:

```
<ComplexType Name="FullName">
    <Property Type="String" Name="LastName"
        Nullable="false" MaxLength="50"
        FixedLength="false" Unicode="true" />
    <Property Type="String" Name="FirstName"
        Nullable="false" MaxLength="50"
        FixedLength="false" Unicode="true" />
</ComplexType>
```

LastName i **FirstName** właściwości **osoby** typu jednostki zostały zamienione na jedną właściwość złożoną, **nazwa**:

```
<EntityType Name="Person">
    <Key>
        <PropertyRef Name="PersonID" />
    </Key>
    <Property Name="PersonID" Type="Int32" Nullable="false"
        annotation:StoreGeneratedPattern="Identity" />
    <Property Name="HireDate" Type="DateTime" />
    <Property Name="EnrollmentDate" Type="DateTime" />
    <Property Name="Name" Type="SchoolModel.FullName" Nullable="false" />
</EntityType>
```

Pokazano w poniższym pliku MSL **ComplexProperty** element używany do mapowania **nazwa** właściwości do kolumn w bazie danych:

```
<EntitySetMapping Name="People">
    <EntityTypeMapping TypeName="SchoolModel.Person">
        <MappingFragment StoreEntitySet="Person">
            <ScalarProperty Name="PersonID" ColumnName="PersonID" />
            <ScalarProperty Name="HireDate" ColumnName="HireDate" />
            <ScalarProperty Name="EnrollmentDate" ColumnName="EnrollmentDate" />
            <ComplexProperty Name="Name" TypeName="SchoolModel.FullName">
                <ScalarProperty Name="FirstName" ColumnName="FirstName" />
                <ScalarProperty Name="LastName" ColumnName="LastName" />
            </ComplexProperty>
        </MappingFragment>
    </EntityTypeMapping>
</EntitySetMapping>
```

Element ComplexTypeMapping (MSL)

ComplexTypeMapping elementu w mapowaniu language specification (MSL) jest elementem podrzędnym elementu ResultMapping i definiuje mapowanie między importowanie funkcji, w modelu koncepcyjnym i procedury składowanej w źródłowym Baza danych, gdy spełnione są następujące:

- Importowanie funkcji zwraca koncepcyjny typu złożonego.
- Nazwy kolumn zwrócone przez procedurę składowaną nie odpowiadają dokładnie nazwy właściwości na typ złożony.

Domyślnie mapowania między kolumnami zwrócone przez procedurę składowaną i typ złożony opiera się na nazwy kolumn i właściwości. Nazwy kolumn nie odpowiadają dokładnie nazw właściwości, należy użyć

ComplexTypeMapping elementu, aby zdefiniować mapowanie. Aby uzyskać przykład domyślnego mapowania Zobacz Element FunctionImportMapping (MSL).

ComplexTypeMapping element może mieć następujące elementy podrzędne:

- ScalarProperty (zero lub więcej)

Odpowiednie atrybuty

W poniższej tabeli opisano atrybuty, które mają zastosowanie do **ComplexTypeMapping** elementu.

NAZWA ATRYBUTU	JEST WYMAGANY	WARTOŚĆ
Nazwa typu	Tak	Nazwa kwalifikowana przez obszar nazw typ złożony, który jest mapowany.

Przykład

Należy wziąć pod uwagę następujące procedury składowanej:

```
CREATE PROCEDURE [dbo].[GetGrades]
    @student_Id int
    AS
        SELECT      EnrollmentID as enroll_id,
                    Grade as grade,
                    CourseID as course_id,
                    StudentID as student_id
        FROM dbo.StudentGrade
        WHERE StudentID = @student_Id
```

Należy również rozważyć następujący typ złożony modelu koncepcyjnego:

```
<ComplexType Name="GradeInfo">
    <Property Type="Int32" Name="EnrollmentID" Nullable="false" />
    <Property Type="Decimal" Name="Grade" Nullable="true"
              Precision="3" Scale="2" />
    <Property Type="Int32" Name="CourseID" Nullable="false" />
    <Property Type="Int32" Name="StudentID" Nullable="false" />
</ComplexType>
```

Aby można było utworzyć importu funkcji, który zwraca wystąpień poprzedniego typu złożonego, mapowanie między kolumnami zwrócone przez procedurę składowaną i typ jednostki musi być zdefiniowany w

ComplexTypeMapping elementu:

```
<FunctionImportMapping FunctionImportName="GetGrades"
                           FunctionName="SchoolModel.Store.GetGrades" >
    <ResultMapping>
        <ComplexTypeMapping TypeName="SchoolModel.GradeInfo">
            <ScalarProperty Name="EnrollmentID" ColumnName="enroll_id"/>
            <ScalarProperty Name="CourseID" ColumnName="course_id"/>
            <ScalarProperty Name="StudentID" ColumnName="student_id"/>
            <ScalarProperty Name="Grade" ColumnName="grade"/>
        </ComplexTypeMapping>
    </ResultMapping>
</FunctionImportMapping>
```

Warunek, Element (MSL)

Warunek elementu w mapowaniu language specification (MSL) powoduje warunków mapowania między modelem koncepcyjnym i podstawowej bazy danych. Mapowania, która jest zdefiniowana w węźle XML jest nieprawidłowy, jeśli wszystkie warunki, jako element podrzędny określonego w **warunek** elementów, są spełnione. W przeciwnym razie mapowanie jest nieprawidłowy. Na przykład, jeśli MappingFragment element zawiera jeden lub więcej

warunek elementy podrzędne, mapowanie zdefiniowane w ramach **MappingFragment** węzeł będzie obowiązywał tylko jeśli wszystkie warunki podrzędnych **Warunek** elementy są spełnione.

Każdy warunek można zastosować do jednej **nazwa** (nazwa właściwości jednostki modelu koncepcyjnego, określony przez **nazwa** atrybut), lub **ColumnName** (nazwa kolumny w Określona baza danych, przez **ColumnName** atrybutu). Gdy **nazwa** ma ustawioną wartość atrybutu, warunek jest sprawdzany na podstawie wartości właściwości jednostki. Gdy **ColumnName** ma ustawioną wartość atrybutu, warunek jest sprawdzana względem wartości kolumny. Tylko jeden z **nazwa** lub **ColumnName** atrybutu można określić w **warunek** elementu.

NOTE

Gdy **warunek** element jest używany w elemencie FunctionImportMapping tylko **nazwa** atrybut nie ma zastosowania.

Warunek element może być elementem podrzędnym następujące elementy:

- AssociationSetMapping
- ComplexProperty
- Obiekcie EntitySetMapping
- MappingFragment
- EntityTypeMapping

Warunek element może mieć żadnych elementów podrzędnych.

Odpowiednie atrybuty

W poniższej tabeli opisano atrybuty, które mają zastosowanie do **warunek** elementu:

NAZWA ATRYBUTU	JEST WYMAGANY	WARTOŚĆ
NazwaKolumny	Nie	Nazwa kolumny tabeli, którego wartość jest używana do warunku.
IsNull	Nie	Wartość true , lub False . Jeśli wartość jest True , a wartość kolumny jest wartości null , lub jeśli wartość jest False i wartość kolumny jest null , warunek jest prawdziwy . W przeciwnym razie warunek ma wartość false. IsNull i wartość atrybutów nie można użyć w tym samym czasie.
Wartość	Nie	Wartość, z którym wartość kolumny jest porównywany. Jeśli wartości są takie same, warunek jest prawdziwy. W przeciwnym razie warunek ma wartość false. IsNull i wartość atrybutów nie można użyć w tym samym czasie.
Nazwa	Nie	Nazwa właściwości jednostki modelu koncepcyjnego, którego wartość jest używana do warunku. Ten atrybut nie ma zastosowania. Jeśli warunek element jest używany w elemencie FunctionImportMapping.

Przykład

W poniższym przykładzie przedstawiono **warunek** elementy jako elementy podrzędne **MappingFragment** elementów. Gdy **DataZatrudnienia** nie ma wartości null i **EnrollmentDate** jest wartość null, dane zamapowane między **SchoolModel.Instructor** typu i **PersonID** i **DataZatrudnienia** kolumn **osoby** tabeli. Gdy **EnrollmentDate** nie ma wartości null i **DataZatrudnienia** jest wartość null, dane zamapowane między **SchoolModel.Student** typu i **PersonID** i **rejestracji** kolumn **osoby** tabeli.

```
<EntitySetMapping Name="People">
    <EntityTypeMapping TypeName="IsTypeOf(SchoolModel.Person)">
        <MappingFragment StoreEntitySet="Person">
            <ScalarProperty Name="PersonID" ColumnName="PersonID" />
            <ScalarProperty Name="FirstName" ColumnName="FirstName" />
            <ScalarProperty Name="LastName" ColumnName="LastName" />
        </MappingFragment>
    </EntityTypeMapping>
    <EntityTypeMapping TypeName="IsTypeOf(SchoolModel.Instructor)">
        <MappingFragment StoreEntitySet="Person">
            <ScalarProperty Name="PersonID" ColumnName="PersonID" />
            <ScalarProperty Name="HireDate" ColumnName="HireDate" />
            <Condition ColumnName="HireDate" IsNull="false" />
            <Condition ColumnName="EnrollmentDate" IsNull="true" />
        </MappingFragment>
    </EntityTypeMapping>
    <EntityTypeMapping TypeName="IsTypeOf(SchoolModel.Student)">
        <MappingFragment StoreEntitySet="Person">
            <ScalarProperty Name="PersonID" ColumnName="PersonID" />
            <ScalarProperty Name="EnrollmentDate"
                ColumnName="EnrollmentDate" />
            <Condition ColumnName="EnrollmentDate" IsNull="false" />
            <Condition ColumnName="HireDate" IsNull="true" />
        </MappingFragment>
    </EntityTypeMapping>
</EntitySetMapping>
```

Element DeleteFunction (MSL)

DeleteFunction elementu w specyfikacji języka (MSL) mapowania mapuje funkcji usuwania w typie jednostki lub skojarzeń w modelu koncepcyjnym procedurę składowaną w bazie danych. Musi być zadeklarowany procedur skadowanych do modyfikacji, które funkcje są mapowane w modelu magazynu. Aby uzyskać więcej informacji zobacz Element — funkcja (SSDL).

NOTE

Jeśli nie mapuję wszystkie trzy insert, update lub delete operacje typu jednostki, do procedur składowanych, niezamapowane operacji zakończy się niepowodzeniem, jeśli wykonywane w czasie wykonywania, i jest generowany, UpdateException.

Stosowane do EntityTypeMapping DeleteFunction

W przypadku zastosowania do elementu EntityTypeMapping **DeleteFunction** element mapy funkcji usuwania typu jednostki w modelu koncepcyjnym do procedury składowanej.

DeleteFunction element może mieć następujące elementy podrzędne, po zastosowaniu do **EntityTypeMapping** elementu:

- Punkt końcowy skojarzenia (zero lub więcej)
- ComplexProperty (zero lub więcej)
- ScarlarProperty (zero lub więcej)

Odpowiednie atrybuty

W poniższej tabeli opisano atrybuty, które mogą być stosowane do **DeleteFunction** elementu, jeśli jest stosowany

do **EntityTypeMapping** elementu.

NAZWA ATRYBUTU	JEST WYMAGANY	WARTOŚĆ
functionName	Tak	Przestrzeń nazw kwalifikowana nazwa procedury składowanej, na który jest mapowany funkcji usuwania. Procedura składowana musi być zadeklarowana w modelu magazynu.
RowsAffectedParameter	Nie	Nazwa parametru output, która zwraca liczbę uwzględnionych wierszy.

Przykład

Poniższy przykład jest oparty na modelu szkoły i pokazuje **DeleteFunction** element mapowania funkcji usuwania **osoby** typu jednostki **DeletePerson**. Procedura składowana **DeletePerson** procedury składowanej jest zadeklarowana w modelu magazynu.

```
<EntityTypeMapping Name="People">
    <EntityTypeMapping TypeName="SchoolModel.Person">
        <MappingFragment StoreEntitySet="Person">
            <ScalarProperty Name="PersonID" ColumnName="PersonID" />
            <ScalarProperty Name="LastName" ColumnName="LastName" />
            <ScalarProperty Name="FirstName" ColumnName="FirstName" />
            <ScalarProperty Name="HireDate" ColumnName="HireDate" />
            <ScalarProperty Name="EnrollmentDate"
                ColumnName="EnrollmentDate" />
        </MappingFragment>
    </EntityTypeMapping>
    <EntityTypeMapping TypeName="SchoolModel.Person">
        <ModificationFunctionMapping>
            <InsertFunction FunctionName="SchoolModel.Store.InsertPerson">
                <ScalarProperty Name="EnrollmentDate"
                    ParameterName="EnrollmentDate" />
                <ScalarProperty Name="HireDate" ParameterName="HireDate" />
                <ScalarProperty Name="FirstName" ParameterName="FirstName" />
                <ScalarProperty Name="LastName" ParameterName="LastName" />
                <ResultBinding Name="PersonID" ColumnName="NewPersonID" />
            </InsertFunction>
            <UpdateFunction FunctionName="SchoolModel.Store.UpdatePerson">
                <ScalarProperty Name="EnrollmentDate"
                    ParameterName="EnrollmentDate"
                    Version="Current" />
                <ScalarProperty Name="HireDate" ParameterName="HireDate"
                    Version="Current" />
                <ScalarProperty Name="FirstName" ParameterName="FirstName"
                    Version="Current" />
                <ScalarProperty Name="LastName" ParameterName="LastName"
                    Version="Current" />
                <ScalarProperty Name="PersonID" ParameterName="PersonID"
                    Version="Current" />
            </UpdateFunction>
            <DeleteFunction FunctionName="SchoolModel.Store.DeletePerson">
                <ScalarProperty Name="PersonID" ParameterName="PersonID" />
            </DeleteFunction>
        </ModificationFunctionMapping>
    </EntityTypeMapping>
</EntityTypeMapping>
```

Stosowane do AssociationSetMapping DeleteFunction

W przypadku zastosowania do elementu AssociationSetMapping **DeleteFunction** element mapy funkcji usuwania skojarzenia w modelu koncepcyjnym do procedury składowanej.

DeleteFunction element może mieć następujące elementy podrzędne, po zastosowaniu do **AssociationSetMapping** elementu:

- Właściwości EndProperty

Odpowiednie atrybuty

W poniższej tabeli opisano atrybuty, które mogą być stosowane do **DeleteFunction** elementu, jeśli jest stosowany do **AssociationSetMapping** elementu.

NAZWA ATRYBUTU	JEST WYMAGANY	WARTOŚĆ
functionName	Tak	Przestrzeń nazw kwalifikowana nazwa procedury składowanej, na który jest mapowany funkcji usuwania. Procedura składowana musi być zadeklarowana w modelu magazynu.
RowsAffectedParameter	Nie	Nazwa parametru output, która zwraca liczbę uwzględnionych wierszy.

Przykład

Poniższy przykład jest oparty na modelu szkoły i pokazuje **DeleteFunction** element używany do mapowania funkcji usuwania **CourseInstructor** skojarzenia **DeleteCourseInstructor** procedury składowanej. **DeleteCourseInstructor** procedury składowanej jest zadeklarowana w modelu magazynu.

```
<AssociationSetMapping Name="CourseInstructor"
    TypeName="SchoolModel.CourseInstructor"
    StoreEntitySet="CourseInstructor">
    <EndProperty Name="Person">
        <ScalarProperty Name="PersonID" ColumnName="PersonID" />
    </EndProperty>
    <EndProperty Name="Course">
        <ScalarProperty Name="CourseID" ColumnName="CourseID" />
    </EndProperty>
    <ModificationFunctionMapping>
        <InsertFunction FunctionName="SchoolModel.Store.InsertCourseInstructor" >
            <EndProperty Name="Course">
                <ScalarProperty Name="CourseID" ParameterName="courseId"/>
            </EndProperty>
            <EndProperty Name="Person">
                <ScalarProperty Name="PersonID" ParameterName="instructorId"/>
            </EndProperty>
        </InsertFunction>
        <DeleteFunction FunctionName="SchoolModel.Store.DeleteCourseInstructor">
            <EndProperty Name="Course">
                <ScalarProperty Name="CourseID" ParameterName="courseId"/>
            </EndProperty>
            <EndProperty Name="Person">
                <ScalarProperty Name="PersonID" ParameterName="instructorId"/>
            </EndProperty>
        </DeleteFunction>
    </ModificationFunctionMapping>
</AssociationSetMapping>
```

Element właściwości EndProperty (MSL)

Właściwości EndProperty element w mapowaniu language specification (MSL) definiuje mapowanie między punktu końcowego lub funkcją Modyfikowanie skojarzenia typu modelu koncepcyjnego i podstawowej bazy danych. Mapowanie kolumny właściwości jest określone w elemencie ScalarProperty podrzędnym.

Gdy **właściwości EndProperty** element jest używany do definiowania mapowania dla elementu end skojarzenia modelu koncepcyjnego, jest elementem podrzędnym elementu AssociationSetMapping. Gdy **właściwości EndProperty** element jest używany do definiowania mapowania dla funkcji modyfikacji skojarzenia modelu koncepcyjnego, jest elementem podrzędnym elementu InsertFunction lub DeleteFunction elementu.

Właściwości EndProperty element może mieć następujące elementy podrzędne:

- ScalarProperty (zero lub więcej)

Odpowiednie atrybuty

W poniższej tabeli opisano atrybuty, które mają zastosowanie do **właściwości EndProperty** elementu:

NAZWA ATRYBUTU	JEST WYMAGANY	WARTOŚĆ
Nazwa	Tak	Nazwa elementu end skojarzenia, który jest mapowany.

Przykład

W poniższym przykładzie przedstawiono **AssociationSetMapping** elementu, w którym **klucza Obcego_kurs_działu** skojarzeń w modelu koncepcyjnym jest mapowany na **Kurs** tabeli w bazie danych. Mapowania między właściwościami typu skojarzenia i kolumn w tabeli są określone w podrzędnej **właściwości EndProperty** elementów.

```
<AssociationSetMapping Name="FK_Course_Department"
    TypeName="SchoolModel.FK_Course_Department"
    StoreEntitySet="Course">
    <EndProperty Name="Department">
        <ScalarProperty Name="DepartmentID" ColumnName="DepartmentID" />
    </EndProperty>
    <EndProperty Name="Course">
        <ScalarProperty Name="CourseID" ColumnName="CourseID" />
    </EndProperty>
</AssociationSetMapping>
```

Przykład

W poniższym przykładzie przedstawiono **właściwości EndProperty** mapowania funkcji wstawiania i usuwania skojarzenia elementu (**CourseInstructor**) do procedur składowanych w bazie danych. Funkcje, które są mapowane na są deklarowane w modelu magazynu.

```

<AssociationSetMapping Name="CourseInstructor"
    TypeName="SchoolModel.CourseInstructor"
    StoreEntitySet="CourseInstructor">
    <EndProperty Name="Person">
        <ScalarProperty Name="PersonID" ColumnName="PersonID" />
    </EndProperty>
    <EndProperty Name="Course">
        <ScalarProperty Name="CourseID" ColumnName="CourseID" />
    </EndProperty>
    <ModificationFunctionMapping>
        <InsertFunction FunctionName="SchoolModel.Store.InsertCourseInstructor" >
            <EndProperty Name="Course">
                <ScalarProperty Name="CourseID" ParameterName="courseId"/>
            </EndProperty>
            <EndProperty Name="Person">
                <ScalarProperty Name="PersonID" ParameterName="instructorId"/>
            </EndProperty>
        </InsertFunction>
        <DeleteFunction FunctionName="SchoolModel.Store.DeleteCourseInstructor">
            <EndProperty Name="Course">
                <ScalarProperty Name="CourseID" ParameterName="courseId"/>
            </EndProperty>
            <EndProperty Name="Person">
                <ScalarProperty Name="PersonID" ParameterName="instructorId"/>
            </EndProperty>
        </DeleteFunction>
    </ModificationFunctionMapping>
</AssociationSetMapping>

```

Element elemencie EntityContainerMapping (MSL)

Elemencie EntityContainerMapping elementu w specyfikacji języka (MSL) mapowania mapuje kontener jednostek w modelu koncepcyjnym kontener jednostek w modelu magazynu. **Elemencie EntityContainerMapping** element jest elementem podrzędnym elementu mapowania.

Elemencie EntityContainerMapping element może mieć następujących elementów podrzędnych (w podanej kolejności):

- Obiekcie EntitySetMapping (zero lub więcej)
- AssociationSetMapping (zero lub więcej)
- FunctionImportMapping (zero lub więcej)

Odpowiednie atrybuty

W poniższej tabeli opisano atrybuty, które mogą być stosowane do **elemencie EntityContainerMapping** elementu.

NAZWA ATRYBUTU	JEST WYMAGANY	WARTOŚĆ
StorageModelContainer	Tak	Nazwa kontenera magazynu jednostki modelu, który jest mapowany.
CdmEntityContainer	Tak	Nazwa kontenera jednostek modelu koncepcyjnego, który jest mapowany.

NAZWA ATRYBUTU	JEST WYMAGANY	WARTOŚĆ
GenerateUpdateViews	Nie	Wartość true, lub False. Jeśli False , są generowane nie widoków aktualizacji. Ten atrybut powinien być ustawiony na False przypadku mapowania tylko do odczytu, która będzie nieprawidłowa, ponieważ dane mogą być nie obustronne pomyślnie. Wartość domyślna to True .

Przykład

W poniższym przykładzie przedstawiono **element EntityContainerMapping** element, który mapuje **SchoolModelEntities** container (kontener modelu koncepcyjnego entity) do **SchoolModelStoreContainer** container (kontener jednostek modelu magazynu):

```
<EntityContainerMapping StorageEntityContainer="SchoolModelStoreContainer"
                       CdmEntityContainer="SchoolModelEntities">
  <EntitySetMapping Name="Courses">
    <EntityTypeMapping TypeName="c.Course">
      <MappingFragment StoreEntitySet="Course">
        <ScalarProperty Name="CourseID" ColumnName="CourseID" />
        <ScalarProperty Name="Title" ColumnName="Title" />
        <ScalarProperty Name="Credits" ColumnName="Credits" />
        <ScalarProperty Name="DepartmentID" ColumnName="DepartmentID" />
      </MappingFragment>
    </EntityTypeMapping>
  </EntitySetMapping>
  <EntitySetMapping Name="Departments">
    <EntityTypeMapping TypeName="c.Department">
      <MappingFragment StoreEntitySet="Department">
        <ScalarProperty Name="DepartmentID" ColumnName="DepartmentID" />
        <ScalarProperty Name="Name" ColumnName="Name" />
        <ScalarProperty Name="Budget" ColumnName="Budget" />
        <ScalarProperty Name="StartDate" ColumnName="StartDate" />
        <ScalarProperty Name="Administrator" ColumnName="Administrator" />
      </MappingFragment>
    </EntityTypeMapping>
  </EntitySetMapping>
</EntityContainerMapping>
```

Element obiekcie EntitySetMapping (MSL)

Obiekcie EntitySetMapping elementu w specyfikacji języka (MSL) mapy wszystkie typy w jednostce modelu koncepcyjnego równa jednostki mapowania zestawów w modelu magazynu. Zestawu jednostek w modelu koncepcyjnym to kontener logiczny dla wystąpień jednostek z tego samego typu (i jego typów pochodnych). Zestawu jednostek w modelu magazynu reprezentuje tabelę lub widok w bazie danych. Zestaw jednostek modelu koncepcyjnego jest określony przez wartość **nazwa** atrybutu **obiekcie EntitySetMapping** elementu. Mapowany do tabeli lub widoku jest określona przez **StoreEntitySet** atrybutu w każdym elemencie MappingFragment podrzędnej lub w **obiekcie EntitySetMapping** samego elementu.

Obiekcie EntitySetMapping element może mieć następujące elementy podrzędne:

- EntityTypeMapping (zero lub więcej)
- Obiekt QueryView (zero lub jeden)
- MappingFragment (zero lub więcej)

Odpowiednie atrybuty

W poniższej tabeli opisano atrybuty, które mogą być stosowane do **obiekcie EntitySetMapping** elementu.

NAZWA ATRYBUTU	JEST WYMAGANY	WARTOŚĆ
Nazwa	Tak	Nazwa zestawu jednostek modelu koncepcyjnego, który jest mapowany.
Element TypeName 1	Nie	Nazwa typu jednostki modelu koncepcyjnego, który jest mapowany.
StoreEntitySet 1	Nie	Nazwa zestawu jednostek modelu magazynu, który jest mapowany na.
MakeColumnsDistinct	Nie	Wartość true , lub False w zależności od tego, czy tylko unikatowe wiersze są zwracane. Jeśli ten atrybut jest ustawiony na True , GenerateUpdateViews atrybut elementu w elemencie EntityContainerMapping musi być równa False .

1 TypeName i StoreEntitySet atrybuty można zamiast elementów podrzędnych EntityTypeMapping i MappingFragment mapowania typu pojedynczy element na pojedynczą tabelę.

Przykład

W poniższym przykładzie przedstawiono **obiekcie EntitySetMapping** element, który mapuje trzy typy (typ podstawowy i dwa typy pochodne) w **kursów** zestaw jednostek modelu koncepcyjny do trzech różnych tabel w podstawowej bazie danych. Tabele są określone przez **StoreEntitySet** atrybutu w każdym **MappingFragment** elementu.

```
<EntitySetMapping Name="Courses">
  <EntityTypeMapping TypeName="IsTypeOf(SchoolModel1.Course)">
    <MappingFragment StoreEntitySet="Course">
      <ScalarProperty Name="CourseID" ColumnName="CourseID" />
      <ScalarProperty Name="DepartmentID" ColumnName="DepartmentID" />
      <ScalarProperty Name="Credits" ColumnName="Credits" />
      <ScalarProperty Name="Title" ColumnName="Title" />
    </MappingFragment>
  </EntityTypeMapping>
  <EntityTypeMapping TypeName="IsTypeOf(SchoolModel1.OnlineCourse)">
    <MappingFragment StoreEntitySet="OnlineCourse">
      <ScalarProperty Name="CourseID" ColumnName="CourseID" />
      <ScalarProperty Name="URL" ColumnName="URL" />
    </MappingFragment>
  </EntityTypeMapping>
  <EntityTypeMapping TypeName="IsTypeOf(SchoolModel1.OnsiteCourse)">
    <MappingFragment StoreEntitySet="OnsiteCourse">
      <ScalarProperty Name="CourseID" ColumnName="CourseID" />
      <ScalarProperty Name="Time" ColumnName="Time" />
      <ScalarProperty Name="Days" ColumnName="Days" />
      <ScalarProperty Name="Location" ColumnName="Location" />
    </MappingFragment>
  </EntityTypeMapping>
</EntitySetMapping>
```

Element EntityTypeMapping (MSL)

EntityTypeMapping elementu w mapowaniu language specification (MSL) definiuje mapowanie między typem jednostki w modelu koncepcyjnym i tabel lub widoków w bazie danych. Dla informacji na temat typów jednostek

modelu koncepcyjnego i podstawowej bazy danych tabel lub widoków Zobacz Element EntityType (CSDL) i elementu obiektu EntitySet (SSDL). Typ jednostki modelu koncepcyjnego, który jest mapowany jest określony przez **TypeName** atrybutu **EntityTypeMapping** elementu. Tabela lub widok, który jest mapowany jest określona przez **StoreEntitySet** atrybutu elementu MappingFragment podrzędnego.

ModificationFunctionMapping, element podrzędny może być używane do mapowania Wstawianie, aktualizowanie lub usuwanie funkcji typów jednostek do procedur składowanych w bazie danych.

EntityTypeMapping element może mieć następujące elementy podrzędne:

- MappingFragment (zero lub więcej)
- ModificationFunctionMapping (zero lub jeden)
- ScalarProperty
- Warunek

NOTE

MappingFragment i **ModificationFunctionMapping** elementy nie mogą być elementami podrzędnymi **EntityTypeMapping** element w tym samym czasie.

NOTE

ScalarProperty i **warunek** elementów może zawierać tylko elementy podrzędne **EntityTypeMapping** elementu, gdy jest używany w elemencie FunctionImportMapping.

Odpowiednie atrybuty

W poniższej tabeli opisano atrybuty, które mogą być stosowane do **EntityTypeMapping** elementu.

NAZWA ATRYBUTU	JEST WYMAGANY	WARTOŚĆ
Nazwa typu	Tak	Przestrzeń nazw kwalifikowana nazwa typu jednostki modelu koncepcyjnego, który jest mapowany. Jeśli typ jest abstrakcyjny ani typem pochodnym, wartość musi być <code>IsOfType(Namespace-qualified_type_name)</code>

Przykład

W poniższym przykładzie pokazano element obiektu EntitySetMapping z dwóch podrzędnych **EntityTypeMapping** elementów. W pierwszym **EntityTypeMapping** elementu **SchoolModel.Person** typ jednostki jest mapowany na **osoby** tabeli. W drugim **EntityTypeMapping** element, funkcja aktualizacji z **SchoolModel.Person** typu jest mapowany do procedury składowanej **UpdatePerson**, w bazie danych .

```

<EntitySetMapping Name="People">
    <EntityTypeMapping TypeName="SchoolModel.Person">
        <MappingFragment StoreEntitySet="Person">
            <ScalarProperty Name="PersonID" ColumnName="PersonID" />
            <ScalarProperty Name="LastName" ColumnName="LastName" />
            <ScalarProperty Name="FirstName" ColumnName="FirstName" />
            <ScalarProperty Name="HireDate" ColumnName="HireDate" />
            <ScalarProperty Name="EnrollmentDate" ColumnName="EnrollmentDate" />
        </MappingFragment>
    </EntityTypeMapping>
    <EntityTypeMapping TypeName="SchoolModel.Person">
        <ModificationFunctionMapping>
            <UpdateFunction FunctionName="SchoolModel.Store.UpdatePerson">
                <ScalarProperty Name="EnrollmentDate" ParameterName="EnrollmentDate"
                               Version="Current" />
                <ScalarProperty Name="HireDate" ParameterName="HireDate"
                               Version="Current" />
                <ScalarProperty Name="FirstName" ParameterName="FirstName"
                               Version="Current" />
                <ScalarProperty Name="LastName" ParameterName="LastName"
                               Version="Current" />
                <ScalarProperty Name="PersonID" ParameterName="PersonID"
                               Version="Current" />
            </UpdateFunction>
        </ModificationFunctionMapping>
    </EntityTypeMapping>
</EntitySetMapping>

```

Przykład

W kolejnym przykładzie pokazano mapowanie hierarchii typów, w którym główny typ jest abstrakcyjny. Zwróć uwagę na użycie `IsOfType` składnia **TypeName** atrybutów.

```

<EntitySetMapping Name="People">
    <EntityTypeMapping TypeName="IsTypeOf(SchoolModel.Person)">
        <MappingFragment StoreEntitySet="Person">
            <ScalarProperty Name="PersonID" ColumnName="PersonID" />
            <ScalarProperty Name="FirstName" ColumnName="FirstName" />
            <ScalarProperty Name="LastName" ColumnName="LastName" />
        </MappingFragment>
    </EntityTypeMapping>
    <EntityTypeMapping TypeName="IsTypeOf(SchoolModel.Instructor)">
        <MappingFragment StoreEntitySet="Person">
            <ScalarProperty Name="PersonID" ColumnName="PersonID" />
            <ScalarProperty Name="HireDate" ColumnName="HireDate" />
            <Condition ColumnName="HireDate" IsNull="false" />
            <Condition ColumnName="EnrollmentDate" IsNull="true" />
        </MappingFragment>
    </EntityTypeMapping>
    <EntityTypeMapping TypeName="IsTypeOf(SchoolModel.Student)">
        <MappingFragment StoreEntitySet="Person">
            <ScalarProperty Name="PersonID" ColumnName="PersonID" />
            <ScalarProperty Name="EnrollmentDate"
                           ColumnName="EnrollmentDate" />
            <Condition ColumnName="EnrollmentDate" IsNull="false" />
            <Condition ColumnName="HireDate" IsNull="true" />
        </MappingFragment>
    </EntityTypeMapping>
</EntitySetMapping>

```

Element **FunctionImportMapping** (MSL)

FunctionImportMapping elementu w mapowaniu language specification (MSL) definiuje mapowanie między

importowanie funkcji, w modelu koncepcyjnego i procedury składowanej lub funkcji w bazie danych. Importy funkcja musi być zadeklarowana w modelu koncepcyjnym i procedur składowanych musi być zadeklarowana w modelu magazynu. Aby uzyskać więcej informacji zobacz Element FunctionImport (CSDL) i funkcji — Element (SSDL).

NOTE

Domyślnie jeśli import funkcja zwraca modelu koncepcyjnego typu jednostki lub typ złożony, następnie nazwy kolumn zwrócone przez procedurę składowaną podstawowej musi dokładnie odpowiadać nazwy właściwości w typie modelu koncepcyjnego. Jeśli nazwy kolumn nie odpowiadają dokładnie nazw właściwości, mapowanie musi być zdefiniowany w elemencie ResultMapping.

FunctionImportMapping element może mieć następujące elementy podrzędne:

- ResultMapping (zero lub więcej)

Odpowiednie atrybuty

W poniższej tabeli opisano atrybuty, które mają zastosowanie do **FunctionImportMapping** elementu:

NAZWA ATRYBUTU	JEST WYMAGANY	WARTOŚĆ
FunctionImportName	Tak	Nazwa funkcji importowania w modelu koncepcyjnym, który jest mapowany.
functionName	Tak	Nazwa kwalifikowana przez obszar nazw funkcji w modelu magazynu, który jest mapowany.

Przykład

Poniższy przykład jest oparty na modelu szkoły. Rozważmy następującą funkcję w modelu magazynu:

```
<Function Name="GetStudentGrades" Aggregate="false"
          BuiltIn="false" NiladicFunction="false"
          IsComposable="false" ParameterTypeSemantics="AllowImplicitConversion"
          Schema="dbo">
    <Parameter Name="StudentID" Type="int" Mode="In" />
</Function>
```

Należy również rozważyć importowanie tej funkcji w modelu koncepcyjnym:

```
<FunctionImport Name="GetStudentGrades" EntitySet="StudentGrades"
                  ReturnType="Collection(SchoolModel.StudentGrade)">
    <Parameter Name="StudentID" Mode="In" Type="Int32" />
</FunctionImport>
```

W poniższych pokazano przykład **FunctionImportMapping** element używany do mapowania funkcji i funkcji importu powyżej ze sobą:

```
<FunctionImportMapping FunctionImportName="GetStudentGrades"
                           FunctionName="SchoolModel.Store.GetStudentGrades" />
```

Element InsertFunction (MSL)

InsertFunction elementu w specyfikacji języka (MSL) mapowania mapuje funkcji wstawiania w typie jednostki lub

skojarzeń w modelu koncepcyjnym procedurę składowaną w bazie danych. Musi być zadeklarowany procedury składanych do modyfikacji, które funkcje są mapowane w modelu magazynu. Aby uzyskać więcej informacji zobacz Element — funkcja (SSDL).

NOTE

Jeśli nie mapują wszystkie trzy insert, update lub delete operacje typu jednostki, do procedur składanych, niezamapowane operacji zakończy się niepowodzeniem, jeśli wykonywane w czasie wykonywania, i jest generowany, UpdateException.

InsertFunction element może być elementem podrzędnym elementu **ModificationFunctionMapping** i zastosowane do elementu **EntityTypeMapping** lub **AssociationSetMapping** element.

Stosowane do **EntityTypeMapping** **InsertFunction**

W przypadku zastosowania do elementu **EntityTypeMapping** **InsertFunction** element mapy typu jednostki w modelu koncepcyjnym funkcji wstawiania do procedury składowej.

InsertFunction element może mieć następujące elementy podrzędne, po zastosowaniu do **EntityTypeMapping** elementu:

- Punkt końcowy skojarzenia (zero lub więcej)
- ComplexProperty (zero lub więcej)
- ResultBinding (zero lub jeden)
- ScalarProperty (zero lub więcej)

Odpowiednie atrybuty

W poniższej tabeli opisano atrybuty, które mogą być stosowane do **InsertFunction** elementu, gdy jest stosowany do **EntityTypeMapping** elementu.

NAZWA ATRYBUTU	JEST WYMAGANY	WARTOŚĆ
functionName	Tak	Przestrzeń nazw kwalifikowana nazwa procedury składowej, na który jest mapowany funkcji wstawiania. Procedura składowana musi być zadeklarowany w modelu magazynu.
RowsAffectedParameter	Nie	Nazwa parametru output, która zwraca liczbę wierszy dotyczy.

Przykład

Poniższy przykład jest oparty na modelu szkoły i pokazuje **InsertFunction** element używany do mapowania funkcji wstawiania typu jednostki osoby **InsertPerson** procedury składowej. **InsertPerson** procedury składowej jest zadeklarowana w modelu magazynu.

```

<EntityTypeMapping TypeName="SchoolModel.Person">
  <ModificationFunctionMapping>
    <InsertFunction FunctionName="SchoolModel.Store.InsertPerson">
      <ScalarProperty Name="EnrollmentDate"
                      ParameterName="EnrollmentDate" />
      <ScalarProperty Name="HireDate" ParameterName="HireDate" />
      <ScalarProperty Name="FirstName" ParameterName="FirstName" />
      <ScalarProperty Name="LastName" ParameterName="LastName" />
      <ResultBinding Name="PersonID" ColumnName="NewPersonID" />
    </InsertFunction>
    <UpdateFunction FunctionName="SchoolModel.Store.UpdatePerson">
      <ScalarProperty Name="EnrollmentDate"
                      ParameterName="EnrollmentDate"
                      Version="Current" />
      <ScalarProperty Name="HireDate" ParameterName="HireDate"
                      Version="Current" />
      <ScalarProperty Name="FirstName" ParameterName="FirstName"
                      Version="Current" />
      <ScalarProperty Name="LastName" ParameterName="LastName"
                      Version="Current" />
      <ScalarProperty Name="PersonID" ParameterName="PersonID"
                      Version="Current" />
    </UpdateFunction>
    <DeleteFunction FunctionName="SchoolModel.Store.DeletePerson">
      <ScalarProperty Name="PersonID" ParameterName="PersonID" />
    </DeleteFunction>
  </ModificationFunctionMapping>
</EntityTypeMapping>

```

Stosowane do AssociationSetMapping InsertFunction

W przypadku zastosowania do elementu AssociationSetMapping **InsertFunction** element mapy funkcji wstawiania skojarzenia w modelu koncepcyjnym do procedury składowanej.

InsertFunction element może mieć następujące elementy podrzędne, po zastosowaniu do **AssociationSetMapping** elementu:

- Właściwości EndProperty

Odpowiednie atrybuty

W poniższej tabeli opisano atrybuty, które mogą być stosowane do **InsertFunction** elementu, jeśli jest stosowany do **AssociationSetMapping** elementu.

NAZWA ATRYBUTU	JEST WYMAGANY	WARTOŚĆ
functionName	Tak	Przestrzeń nazw kwalifikowana nazwa procedury składowanej, na który jest mapowany funkcji wstawiania. Procedura składowana musi być zadeklarowany w modelu magazynu.
RowsAffectedParameter	Nie	Nazwa parametru output, która zwraca liczbę uwzględnionych wierszy.

Przykład

Poniższy przykład jest oparty na modelu szkoły i pokazuje **InsertFunction** element używany do mapowania funkcji wstawiania **CourseInstructor** skojarzenia **InsertCourseInstructor** procedury składowanej.

InsertCourseInstructor procedury składowanej jest zadeklarowana w modelu magazynu.

```

<AssociationSetMapping Name="CourseInstructor"
    TypeName="SchoolModel.CourseInstructor"
    StoreEntitySet="CourseInstructor">
    <EndProperty Name="Person">
        <ScalarProperty Name="PersonID" ColumnName="PersonID" />
    </EndProperty>
    <EndProperty Name="Course">
        <ScalarProperty Name="CourseID" ColumnName="CourseID" />
    </EndProperty>
    <ModificationFunctionMapping>
        <InsertFunction FunctionName="SchoolModel.Store.InsertCourseInstructor" >
            <EndProperty Name="Course">
                <ScalarProperty Name="CourseID" ParameterName="courseId"/>
            </EndProperty>
            <EndProperty Name="Person">
                <ScalarProperty Name="PersonID" ParameterName="instructorId"/>
            </EndProperty>
        </InsertFunction>
        <DeleteFunction FunctionName="SchoolModel.Store.DeleteCourseInstructor">
            <EndProperty Name="Course">
                <ScalarProperty Name="CourseID" ParameterName="courseId"/>
            </EndProperty>
            <EndProperty Name="Person">
                <ScalarProperty Name="PersonID" ParameterName="instructorId"/>
            </EndProperty>
        </DeleteFunction>
    </ModificationFunctionMapping>
</AssociationSetMapping>

```

Mapowanie elementu (MSL)

Mapowania element w mapowaniu language specification (MSL) zawiera informacje umożliwiające mapowanie obiektów, które są zdefiniowane w model koncepcyjny do bazy danych (zgodnie z opisem w modelu pamięci masowej). Aby uzyskać więcej informacji zobacz specyfikacja CSDL i specyfikacja SSDL.

Mapowania element jest elementem głównym specyfikacji mapowania. Przestrzeń nazw XML do mapowania specyfikacji jest <http://schemas.microsoft.com/ado/2009/11/mapping/cs>.

Element mapowania może używać następujących elementów podrzędnych (w podanej kolejności):

- Alias (zero lub więcej)
- Elementy EntityContainerMapping (dokładnie jeden)

Nazwy koncepcje i typów modelu magazynu, które są określone w pliku MSL musi być kwalifikowana przy użyciu ich nazw odpowiednich przestrzeni nazw. Dla informacji o modelu koncepcyjnego nazwa przestrzeni nazw Zobacz Element schematu (CSDL). Informacje o nazwie przestrzeni nazw w modelu magazynu na ten temat można znaleźć w elemencie schematu (SSDL). Aliasy dla przestrzeni nazw, które są używane w pliku MSL można zdefiniować w elemencie aliasu.

Odpowiednie atrybuty

W poniższej tabeli opisano atrybuty, które mogą być stosowane do **mapowanie** elementu.

NAZWA ATRYBUTU	JEST WYMAGANY	WARTOŚĆ
miejscie	Tak	C-S. To jest wartość stałą i nie można jej zmienić.

Przykład

W poniższym przykładzie przedstawiono **mapowanie** element, który opiera się na części modelu szkoły. Aby

uzyskać więcej informacji na temat modelu szkoły zobacz Przewodnik Szybki Start (Entity Framework):

```
<Mapping Space="C-S"
      xmlns="http://schemas.microsoft.com/ado/2009/11/mapping/cs">
  <Alias Key="c" Value="SchoolModel"/>
  <EntityContainerMapping StorageEntityContainer="SchoolModelStoreContainer"
    CdmEntityContainer="SchoolModelEntities">
    <EntitySetMapping Name="Courses">
      <EntityTypeMapping TypeName="c.Course">
        <MappingFragment StoreEntitySet="Course">
          <ScalarProperty Name="CourseID" ColumnName="CourseID" />
          <ScalarProperty Name="Title" ColumnName="Title" />
          <ScalarProperty Name="Credits" ColumnName="Credits" />
          <ScalarProperty Name="DepartmentID" ColumnName="DepartmentID" />
        </MappingFragment>
      </EntityTypeMapping>
    </EntitySetMapping>
    <EntitySetMapping Name="Departments">
      <EntityTypeMapping TypeName="c.Department">
        <MappingFragment StoreEntitySet="Department">
          <ScalarProperty Name="DepartmentID" ColumnName="DepartmentID" />
          <ScalarProperty Name="Name" ColumnName="Name" />
          <ScalarProperty Name="Budget" ColumnName="Budget" />
          <ScalarProperty Name="StartDate" ColumnName="StartDate" />
          <ScalarProperty Name="Administrator" ColumnName="Administrator" />
        </MappingFragment>
      </EntityTypeMapping>
    </EntitySetMapping>
  </EntityContainerMapping>
</Mapping>
```

Element MappingFragment (MSL)

MappingFragment element w mapowaniu language specification (MSL) definiuje mapowanie między właściwościami typu jednostki modelu koncepcyjnego i tabeli lub widoku w bazie danych. Dla informacji na temat typów jednostek modelu koncepcyjnego i podstawowej bazy danych tabel lub widoków Zobacz Element EntityType (CSDL) i elementu obiektu EntitySet (SSDL). **MappingFragment** może być elementem podrzędnym elementu EntityTypeMapping lub element w obiekcie EntitySetMapping.

MappingFragment element może mieć następujące elementy podrzędne:

- ComplexType (zero lub więcej)
- ScalarProperty (zero lub więcej)
- Warunek (zero lub więcej)

Odpowiednie atrybuty

W poniższej tabeli opisano atrybuty, które mogą być stosowane do **MappingFragment** elementu.

NAZWA ATRYBUTU	JEST WYMAGANY	WARTOŚĆ
StoreEntitySet	Tak	Nazwa tabeli lub widoku, który jest mapowany.

NAZWA ATRYBUTU	JEST WYMAGANY	WARTOŚĆ
MakeColumnsDistinct	Nie	<p>Wartość true, lub False w zależności od tego, czy tylko unikatowe wiersze są zwracane.</p> <p>Jeśli ten atrybut jest ustawiony na True, GenerateUpdateViews atrybut elementu w elemencie EntityContainerMapping musi być równa False.</p>

Przykład

W poniższym przykładzie przedstawiono **MappingFragment** element jako element podrzędny **EntityTypeMapping** elementu. W tym przykładzie właściwości **kurs** typu w modelu koncepcyjnym są mapowane na kolumny **kurs** tabeli w bazie danych.

```
<EntitySetMapping Name="Courses">
  <EntityTypeMapping TypeName="SchoolModel.Course">
    <MappingFragment StoreEntitySet="Course">
      <ScalarProperty Name="CourseID" ColumnName="CourseID" />
      <ScalarProperty Name="Title" ColumnName="Title" />
      <ScalarProperty Name="Credits" ColumnName="Credits" />
      <ScalarProperty Name="DepartmentID" ColumnName="DepartmentID" />
    </MappingFragment>
  </EntityTypeMapping>
</EntitySetMapping>
```

Przykład

W poniższym przykładzie przedstawiono **MappingFragment** element jako element podrzędny **obiekcie EntitySetMapping** elementu. Tak jak w przykładzie powyżej właściwości **kurs** typu w modelu koncepcyjnym są mapowane na kolumny **kurs** tabeli w bazie danych.

```
<EntitySetMapping Name="Courses" TypeName="SchoolModel.Course">
  <MappingFragment StoreEntitySet="Course">
    <ScalarProperty Name="CourseID" ColumnName="CourseID" />
    <ScalarProperty Name="Title" ColumnName="Title" />
    <ScalarProperty Name="Credits" ColumnName="Credits" />
    <ScalarProperty Name="DepartmentID" ColumnName="DepartmentID" />
  </MappingFragment>
</EntitySetMapping>
```

Element ModificationFunctionMapping (MSL)

ModificationFunctionMapping elementu w specyfikacji języka (MSL) mapowania mapuje insert, aktualizowanie i usuwanie funkcji typu modelu koncepcyjnego entity do procedur składowanych w bazie danych. **ModificationFunctionMapping** element można również mapować insert i usuwanie funkcji wiele do wielu skojarzeń w modelu koncepcyjnym do procedur składowanych w bazie danych. Musi być zadeklarowany procedur składowanych do modyfikacji, które funkcje są mapowane w modelu magazynu. Aby uzyskać więcej informacji zobacz Element — funkcja (SSDL).

NOTE

Jeśli nie mapują wszystkie trzy insert, update lub delete operacje typu jednostki, do procedur składowanych, niezamapowane operacji zakończy się niepowodzeniem, jeśli wykonywane w czasie wykonywania, i jest generowany, UpdateException.

NOTE

W przypadku funkcji modyfikacji dla jednej jednostki w hierarchii dziedziczenia są mapowane do procedur składowanych, funkcji modyfikacji dla wszystkich typów w hierarchii musi być zamapowana na procedury składowane.

ModificationFunctionMapping element może być elementem podrzędnym elementu EntityTypeMapping lub AssociationSetMapping element.

ModificationFunctionMapping element może mieć następujące elementy podrzędne:

- DeleteFunction (zero lub jeden)
- InsertFunction (zero lub jeden)
- UpdateFunction (zero lub jeden)

Atrybuty nie są stosowane do **ModificationFunctionMapping** elementu.

Przykład

W poniższym przykładzie przedstawiono zestaw mapowania dla jednostek **osób** zestawu jednostek w modelu szkoły. Oprócz mapowania kolumny **osoby** typu jednostki, mapowanie Wstawianie, aktualizowanie i usuwanie funkcji **osoby** typu są wyświetlane. Funkcje, które są mapowane na są deklarowane w modelu magazynu.

```
<EntitySetMapping Name="People">
    <EntityTypeMapping TypeName="SchoolModel.Person">
        <MappingFragment StoreEntitySet="Person">
            <ScalarProperty Name="PersonID" ColumnName="PersonID" />
            <ScalarProperty Name="LastName" ColumnName="LastName" />
            <ScalarProperty Name="FirstName" ColumnName="FirstName" />
            <ScalarProperty Name="HireDate" ColumnName="HireDate" />
            <ScalarProperty Name="EnrollmentDate"
                ColumnName="EnrollmentDate" />
        </MappingFragment>
    </EntityTypeMapping>
    <EntityTypeMapping TypeName="SchoolModel.Person">
        <ModificationFunctionMapping>
            <InsertFunction FunctionName="SchoolModel.Store.InsertPerson">
                <ScalarProperty Name="EnrollmentDate"
                    ParameterName="EnrollmentDate" />
                <ScalarProperty Name="HireDate" ParameterName="HireDate" />
                <ScalarProperty Name="FirstName" ParameterName="FirstName" />
                <ScalarProperty Name="LastName" ParameterName="LastName" />
                <ResultBinding Name="PersonID" ColumnName="NewPersonID" />
            </InsertFunction>
            <UpdateFunction FunctionName="SchoolModel.Store.UpdatePerson">
                <ScalarProperty Name="EnrollmentDate"
                    ParameterName="EnrollmentDate"
                    Version="Current" />
                <ScalarProperty Name="HireDate" ParameterName="HireDate"
                    Version="Current" />
                <ScalarProperty Name="FirstName" ParameterName="FirstName"
                    Version="Current" />
                <ScalarProperty Name="LastName" ParameterName="LastName"
                    Version="Current" />
                <ScalarProperty Name="PersonID" ParameterName="PersonID"
                    Version="Current" />
            </UpdateFunction>
            <DeleteFunction FunctionName="SchoolModel.Store.DeletePerson">
                <ScalarProperty Name="PersonID" ParameterName="PersonID" />
            </DeleteFunction>
        </ModificationFunctionMapping>
    </EntityTypeMapping>
</EntitySetMapping>
```

Przykład

W poniższym przykładzie pokazano skojarzenia mapowanie dla zestawu **CourseInstructor** skojarzeń w modelu szkoły. Oprócz mapowania kolumny **CourseInstructor** skojarzenia, mapowanie funkcje insert i delete **CourseInstructor** skojarzeń są wyświetlane. Funkcje, które są mapowane na są deklarowane w modelu magazynu.

```
<AssociationSetMapping Name="CourseInstructor"
    TypeName="SchoolModel.CourseInstructor"
    StoreEntitySet="CourseInstructor">
    <EndProperty Name="Person">
        <ScalarProperty Name="PersonID" ColumnName="PersonID" />
    </EndProperty>
    <EndProperty Name="Course">
        <ScalarProperty Name="CourseID" ColumnName="CourseID" />
    </EndProperty>
    <ModificationFunctionMapping>
        <InsertFunction FunctionName="SchoolModel.Store.InsertCourseInstructor" >
            <EndProperty Name="Course">
                <ScalarProperty Name="CourseID" ParameterName="courseId"/>
            </EndProperty>
            <EndProperty Name="Person">
                <ScalarProperty Name="PersonID" ParameterName="instructorId"/>
            </EndProperty>
        </InsertFunction>
        <DeleteFunction FunctionName="SchoolModel.Store.DeleteCourseInstructor">
            <EndProperty Name="Course">
                <ScalarProperty Name="CourseID" ParameterName="courseId"/>
            </EndProperty>
            <EndProperty Name="Person">
                <ScalarProperty Name="PersonID" ParameterName="instructorId"/>
            </EndProperty>
        </DeleteFunction>
    </ModificationFunctionMapping>
</AssociationSetMapping>
```

Obiekt QueryView — Element (MSL)

QueryView elementu w mapowaniu language specification (MSL) definiuje tylko do odczytu mapowanie między typie jednostki lub skojarzeń w modelu koncepcyjnego i tabelę w bazie danych. Mapowanie jest zdefiniowana za pomocą zapytanie SQL jednostki, które zostaną ocenione względem model magazynu i express zestawu pod względem jednostki lub skojarzeń w modelu koncepcyjnym wyników. Ponieważ widoków zapytań są tylko do odczytu, nie możesz użyć polecenia standardowych aktualizacji można zaktualizować typów, które są definiowane przez widoki kwerendę. Należy tworzyć aktualizacje do tych typów, przy użyciu funkcji modyfikacji. Aby uzyskać więcej informacji, zobacz jak: funkcje modyfikacji mapy do procedur składowanych.

NOTE

W **QueryView** element, wyrażenia jednostki SQL, które zawierają **GroupBy**, agregacje grupy lub właściwości nawigacji nie są obsługiwane.

QueryView element może być elementem podrzędnym elementu obiekcie EntitySetMapping lub AssociationSetMapping elementu. W pierwszym przypadku widok zapytania definiuje mapowanie tylko do odczytu dla jednostki w modelu koncepcyjnym. W tym ostatnim przypadku widok zapytania definiuje mapowanie tylko do odczytu dla skojarzenia w modelu koncepcyjnym.

NOTE

Jeśli **AssociationSetMapping** element jest skojarzenie z ograniczeniem referencyjnym, **AssociationSetMapping** element jest ignorowany. Aby uzyskać więcej informacji zobacz Element ReferentialConstraint (CSDL).

QueryView elementu nie może mieć żadnych elementów podrzędnych.

Odpowiednie atrybuty

W poniższej tabeli opisano atrybuty, które mogą być stosowane do **QueryView** elementu.

NAZWA ATRYBUTU	JEST WYMAGANY	WARTOŚĆ
Nazwa typu	Nie	Nazwa typu modelu koncepcyjnego, który jest mapowany w widoku zapytania.

Przykład

W poniższym przykładzie przedstawiono **QueryView** element jako element podrzędny elementu **obiekcie EntitySetMapping** elementu i definiuje mapowanie widoku zapytania **działu** typu jednostki w Model szkoły.

```
<EntitySetMapping Name="Departments" >
  <QueryView>
    SELECT VALUE SchoolModel.Department(d.DepartmentID,
                                              d.Name,
                                              d.Budget,
                                              d.StartDate)
    FROM SchoolModelStoreContainer.Department AS d
    WHERE d.Budget > 150000
  </QueryView>
</EntitySetMapping>
```

Ponieważ zapytanie zwraca tylko podzestaw elementów członkowskich **działu** typu w modelu magazynu **działu** typu w modelu szkoły została zmodyfikowana oparte na tym mapowanie w następujący sposób:

```
<EntityType Name="Department">
  <Key>
    <PropertyRef Name="DepartmentID" />
  </Key>
  <Property Type="Int32" Name="DepartmentID" Nullable="false" />
  <Property Type="String" Name="Name" Nullable="false"
            MaxLength="50" FixedLength="false" Unicode="true" />
  <Property Type="Decimal" Name="Budget" Nullable="false"
            Precision="19" Scale="4" />
  <Property Type="DateTime" Name="StartDate" Nullable="false" />
  <NavigationProperty Name="Courses"
    Relationship="SchoolModel.FK_Course_Department"
    FromRole="Department" ToRole="Course" />
</EntityType>
```

Przykład

W następnym przykładzie pokazano **QueryView** element jako element podrzędny **AssociationSetMapping** elementu i definiuje mapowanie tylko do odczytu dla **FK_Course_Department** skojarzeń w modelu szkoły.

```

<EntityContainerMapping StorageEntityContainer="SchoolModelStoreContainer"
    CdmEntityContainer="SchoolEntities">
    <EntitySetMapping Name="Courses" >
        <QueryView>
            SELECT VALUE SchoolModel.Course(c.CourseID,
                c.Title,
                c.Credits)
            FROM SchoolModelStoreContainer.Course AS c
        </QueryView>
    </EntitySetMapping>
    <EntitySetMapping Name="Departments" >
        <QueryView>
            SELECT VALUE SchoolModel.Department(d.DepartmentID,
                d.Name,
                d.Budget,
                d.StartDate)
            FROM SchoolModelStoreContainer.Department AS d
            WHERE d.Budget > 150000
        </QueryView>
    </EntitySetMapping>
    <AssociationSetMapping Name="FK_Course_Department" >
        <QueryView>
            SELECT VALUE SchoolModel.FK_Course_Department(
                CREATEREF(SchoolEntities.Departments, row(c.DepartmentID), SchoolModel.Department),
                CREATEREF(SchoolEntities.Courses, row(c.CourseID)) )
            FROM SchoolModelStoreContainer.Course AS c
        </QueryView>
    </AssociationSetMapping>
</EntityContainerMapping>

```

Komentarze

Można zdefiniować widoki kwerendę w następujących scenariuszach:

- Zdefiniuj jednostki w modelu koncepcyjnym, który nie zawiera wszystkich właściwości jednostki w modelu magazynu. Obejmuje to właściwości, które nie mają przypisane wartości domyślne, a nie obsługują **null** wartości.
- Mapowanie kolumn obliczanych w modelu magazynu do właściwości typów jednostek w modelu koncepcyjnym.
- Zdefiniuj mapowania, gdzie warunków używanych do jednostek w partycji w modelu koncepcyjnym nie są oparte na równości. Po określeniu warunkowego mapowania przy użyciu **warunek** elementu podanym warunku musi być równa określonej wartości. Aby uzyskać więcej informacji zobacz Element warunek (MSL).
- Tej samej kolumny w modelu magazynu są mapowane na wiele typów w modelu koncepcyjnym.
- Wiele typów są mapowane na tej samej tabeli.
- Definiowanie skojarzeń w modelu koncepcyjnym, które nie są oparte na klucze obce w relacyjnej schemacie.
- Użyj niestandardowej logiki biznesowej do ustawiania wartości właściwości w modelu koncepcyjnym. Na przykład można mapować wartość ciągu "T" w źródle danych na wartość **true**, wartość logiczna, w modelu koncepcyjnym.
- Zdefiniuj filtry warunkowe dla wyników zapytań.
- Wymuszanie mniej ograniczeń danych w modelu koncepcyjnym niż w modelu magazynu. Na przykład można wprowadzone właściwości w modelu koncepcyjnym dopuszczającego wartość null, nawet jeśli nie obsługuje kolumn, na który jest mapowany **null** wartości.

Podczas definiowania widoków zapytań dla jednostek obowiązują następujące zastrzeżenia:

- Widoki kwerendę są tylko do odczytu. Można tworzyć tylko aktualizacje do jednostek przy użyciu funkcji modyfikacji.
- Po zdefiniowaniu typu jednostki, widok zapytania można również definiować wszystkie powiązane jednostki przez widoki kwerendę.

- Kiedy mapujesz skojarzenia typu wiele do wielu z jednostką w modelu magazynu, który reprezentuje tabelę łącze w schemacie relacyjnym należałoby zdefiniować **QueryView** element **AssociationSetMapping** element do tej tabeli łącza.
- Widoki kwerendę muszą być zdefiniowane dla wszystkich typów w hierarchii typów. Można to zrobić w następujący sposób:
 - Za pomocą jednego **QueryView** elementu, który określa pojedyncze zapytanie jednostki SQL, które zwraca sumę wszystkich typów jednostek w hierarchii.
 - Za pomocą jednego **QueryView** elementu, który określa pojedyncze zapytanie jednostki SQL, które używa operatora wielkości liter do zwracania określonego typu w hierarchii na podstawie określonego warunku.
 - Przy użyciu dodatkowego **QueryView** elementu dla określonego typu w hierarchii. W takim przypadku należy użyć **TypeName** atrybutu **QueryView** elementu, aby określić typ jednostki dla każdego widoku.
- Po zdefiniowaniu widok zapytania nie można określić **StorageSetName** atrybutu na **obiekcie EntitySetMapping** elementu.
- Po zdefiniowaniu widok zapytania **obiekcie EntitySetMapping** element nie może również zawierać **właściwość** mapowania.

Element ResultBinding (MSL)

ResultBinding elementu w specyfikacji języka (MSL) mapowania mapuje wartości kolumn, które są zwracane przez procedury składowane do właściwości jednostki w modelu koncepcyjnym w przypadku funkcji modyfikacji typu jednostki są mapowane do przechowywanej procedury przedstawione w źródłowej bazie danych. Na przykład, kiedy wartość w kolumnie tożsamości jest zwracany przez wstawiania procedurą składowaną, **ResultBinding** element mapuje zwracanej wartości do właściwości typu jednostki w modelu koncepcyjnym.

ResultBinding element może być element podrzędny elementu **InsertFunction** lub **UpdateFunction** element.

ResultBinding elementu nie może mieć żadnych elementów podrzędnych.

Odpowiednie atrybuty

W poniższej tabeli opisano atrybuty, które mają zastosowanie do **ResultBinding** elementu:

NAZWA ATRYBUTU	JEST WYMAGANY	WARTOŚĆ
Nazwa	Tak	Nazwa właściwości jednostki w modelu koncepcyjnym, który jest mapowany.
NazwaKolumny	Tak	Nazwa kolumny mapowanej.

Przykład

Poniższy przykład jest oparty na modelu szkoły i pokazuje **InsertFunction** element używany do mapowania funkcji wstawiania **osoby** typu jednostki **InsertPerson** Procedura składowana. (**InsertPerson** procedury składowanej jest pokazany poniżej i jest zadeklarowana w modelu magazynu.) A **ResultBinding** element jest używany do mapowania wartości kolumny, która jest zwracana przez procedurę składowaną (**NewPersonID**) z właściwością typu jednostki (**PersonID**).

```

<EntityTypeMapping TypeName="SchoolModel.Person">
  <ModificationFunctionMapping>
    <InsertFunction FunctionName="SchoolModel.Store.InsertPerson">
      <ScalarProperty Name="EnrollmentDate"
                      ParameterName="EnrollmentDate" />
      <ScalarProperty Name="HireDate" ParameterName="HireDate" />
      <ScalarProperty Name="FirstName" ParameterName="FirstName" />
      <ScalarProperty Name="LastName" ParameterName="LastName" />
      <ResultBinding Name="PersonID" ColumnName="NewPersonID" />
    </InsertFunction>
    <UpdateFunction FunctionName="SchoolModel.Store.UpdatePerson">
      <ScalarProperty Name="EnrollmentDate"
                      ParameterName="EnrollmentDate"
                      Version="Current" />
      <ScalarProperty Name="HireDate" ParameterName="HireDate"
                      Version="Current" />
      <ScalarProperty Name="FirstName" ParameterName="FirstName"
                      Version="Current" />
      <ScalarProperty Name="LastName" ParameterName="LastName"
                      Version="Current" />
      <ScalarProperty Name="PersonID" ParameterName="PersonID"
                      Version="Current" />
    </UpdateFunction>
    <DeleteFunction FunctionName="SchoolModel.Store.DeletePerson">
      <ScalarProperty Name="PersonID" ParameterName="PersonID" />
    </DeleteFunction>
  </ModificationFunctionMapping>
</EntityTypeMapping>

```

W tym artykule opisano języka Transact-SQL **InsertPerson** procedura składowana:

```

CREATE PROCEDURE [dbo].[InsertPerson]
    @LastName nvarchar(50),
    @FirstName nvarchar(50),
    @HireDate datetime,
    @EnrollmentDate datetime
    AS
    INSERT INTO dbo.Person (LastName,
                           FirstName,
                           HireDate,
                           EnrollmentDate)
    VALUES (@LastName,
            @FirstName,
            @HireDate,
            @EnrollmentDate);
    SELECT SCOPE_IDENTITY() as NewPersonID;

```

Element ResultMapping (MSL)

ResultMapping elementu w mapowaniu language specification (MSL) definiuje mapowanie między importowanie funkcji, w modelu koncepcyjnym i procedurę składowaną w bazie danych, gdy spełnione są następujące:

- Importowanie funkcji zwraca modelu koncepcyjnego typu jednostki lub typ złożony.
- Nazwy kolumn zwrócone przez procedurę składowaną nie odpowiadają dokładnie nazwy właściwości w typie jednostki lub typ złożony.

Domyślnie mapowania między kolumnami zwrócone przez procedurę składowaną i typie jednostki lub typ złożony opiera się na nazwy kolumn i właściwości. Nazwy kolumn nie odpowiadają dokładnie nazw właściwości, należy użyć **ResultMapping** elementu, aby zdefiniować mapowanie. Aby uzyskać przykład domyślnego mapowania Zobacz Element FunctionImportMapping (MSL).

ResultMapping element jest elementem podrzędnym elementu FunctionImportMapping.

ResultMapping element może mieć następujące elementy podrzędne:

- EntityTypeMapping (zero lub więcej)
- ComplexTypeMapping

Atrybuty nie są stosowane do **ResultMapping** elementu.

Przykład

Należy wziąć pod uwagę następujące procedury składowanej:

```
CREATE PROCEDURE [dbo].[GetGrades]
    @student_Id int
    AS
        SELECT      EnrollmentID as enroll_id,
                    Grade as grade,
                    CourseID as course_id,
                    StudentID as student_id
        FROM dbo.StudentGrade
        WHERE StudentID = @student_Id
```

Ponadto należy wziąć pod uwagę następujące typie modelu koncepcyjnego entity:

```
<EntityType Name="StudentGrade">
    <Key>
        <PropertyRef Name="EnrollmentID" />
    </Key>
    <Property Name="EnrollmentID" Type="Int32" Nullable="false"
              annotation:StoreGeneratedPattern="Identity" />
    <Property Name="CourseID" Type="Int32" Nullable="false" />
    <Property Name="StudentID" Type="Int32" Nullable="false" />
    <Property Name="Grade" Type="Decimal" Precision="3" Scale="2" />
</EntityType>
```

Aby można było utworzyć importu funkcji, który zwraca wystąpienia poprzedni typ jednostki, mapowanie między kolumnami zwrócone przez procedurę składowaną i typ jednostki musi być zdefiniowany w **ResultMapping** elementu:

```
<FunctionImportMapping FunctionImportName="GetGrades"
                           FunctionName="SchoolModel.Store.GetGrades" >
    <ResultMapping>
        <EntityTypeMapping TypeName="SchoolModel.StudentGrade">
            <ScalarProperty Name="EnrollmentID" ColumnName="enroll_id"/>
            <ScalarProperty Name="CourseID" ColumnName="course_id"/>
            <ScalarProperty Name="StudentID" ColumnName="student_id"/>
            <ScalarProperty Name="Grade" ColumnName="grade"/>
        </EntityTypeMapping>
    </ResultMapping>
</FunctionImportMapping>
```

Element ScalarProperty (MSL)

ScalarProperty elementu w specyfikacji języka (MSL) mapowania mapuje właściwość typu jednostki modelu koncepcyjnego, typu złożonego lub skojarzenie tabeli kolumna lub parametr procedurę składowaną w bazie danych.

NOTE

Musi być zadeklarowany procedur składowanych do modyfikacji, które funkcje są mapowane w modelu magazynu. Aby uzyskać więcej informacji zobacz Element — funkcja (SSDL).

ScalarProperty element może być elementem podrzędnym następujące elementy:

- MappingFragment
- InsertFunction
- UpdateFunction
- DeleteFunction
- Właściwości EndProperty
- ComplexProperty
- ResultMapping

Jako element podrzędny elementu **MappingFragment**, **ComplexProperty**, lub **właściwości EndProperty** elementu **ScalarProperty** element mapy właściwości w modelu koncepcyjnym z kolumną w bazie danych. Jako element podrzędny elementu **InsertFunction**, **UpdateFunction**, lub **DeleteFunction** elementu **ScalarProperty** element mapy właściwości w modelu koncepcyjnym do parametru procedury składowanej.

ScalarProperty elementu nie może mieć żadnych elementów podrzędnych.

Odpowiednie atrybuty

Atrybuty, które są stosowane do **ScalarProperty** elementu różnią się w zależności od roli tego elementu.

W poniższej tabeli opisano atrybuty, które są stosowane, kiedy **ScalarProperty** element jest używany do mapowania właściwości modelu koncepcyjnego z kolumną w bazie danych:

NAZWA ATRYBUTU	JEST WYMAGANY	WARTOŚĆ
Nazwa	Tak	Nazwa właściwości modelu koncepcyjnego, który jest mapowany.
NazwaKolumny	Tak	Nazwa kolumny tabeli, który jest mapowany.

W poniższej tabeli opisano atrybuty, które mają zastosowanie do **ScalarProperty** elementu, gdy jest używany do mapowania właściwości modelu koncepcyjnego parametru procedury składowanej:

NAZWA ATRYBUTU	JEST WYMAGANY	WARTOŚĆ
Nazwa	Tak	Nazwa właściwości modelu koncepcyjnego, który jest mapowany.
Nazwa parametru	Tak	Nazwa parametru, który jest mapowany.
Wersja	Nie	Bieżący lub oryginalnego w zależności od tego, czy bieżąca wartość lub oryginalnej wartości właściwości powinny być używane do kontroli współbieżności.

Przykład

W poniższym przykładzie przedstawiono **ScalarProperty** element używany na dwa sposoby:

- Do mapowania właściwości **osoby** typ jednostki do kolumn **osoby** tabeli.
- Do mapowania właściwości **osoby** typu jednostki parametrom **UpdatePerson** procedury składowanej. Procedury składowane są deklarowane w modelu magazynu.

```

<EntitySetMapping Name="People">
  <EntityTypeMapping TypeName="SchoolModel.Person">
    <MappingFragment StoreEntitySet="Person">
      <ScalarProperty Name="PersonID" ColumnName="PersonID" />
      <ScalarProperty Name="LastName" ColumnName="LastName" />
      <ScalarProperty Name="FirstName" ColumnName="FirstName" />
      <ScalarProperty Name="HireDate" ColumnName="HireDate" />
      <ScalarProperty Name="EnrollmentDate"
                     ColumnName="EnrollmentDate" />
    </MappingFragment>
  </EntityTypeMapping>
  <EntityTypeMapping TypeName="SchoolModel.Person">
    <ModificationFunctionMapping>
      <InsertFunction FunctionName="SchoolModel.Store.InsertPerson">
        <ScalarProperty Name="EnrollmentDate"
                       ParameterName="EnrollmentDate" />
        <ScalarProperty Name="HireDate" ParameterName="HireDate" />
        <ScalarProperty Name="FirstName" ParameterName="FirstName" />
        <ScalarProperty Name="LastName" ParameterName="LastName" />
        <ResultBinding Name="PersonID" ColumnName="NewPersonID" />
      </InsertFunction>
      <UpdateFunction FunctionName="SchoolModel.Store.UpdatePerson">
        <ScalarProperty Name="EnrollmentDate"
                       ParameterName="EnrollmentDate"
                       Version="Current" />
        <ScalarProperty Name="HireDate" ParameterName="HireDate"
                       Version="Current" />
        <ScalarProperty Name="FirstName" ParameterName="FirstName"
                       Version="Current" />
        <ScalarProperty Name="LastName" ParameterName="LastName"
                       Version="Current" />
        <ScalarProperty Name="PersonID" ParameterName="PersonID"
                       Version="Current" />
      </UpdateFunction>
      <DeleteFunction FunctionName="SchoolModel.Store.DeletePerson">
        <ScalarProperty Name="PersonID" ParameterName="PersonID" />
      </DeleteFunction>
    </ModificationFunctionMapping>
  </EntityTypeMapping>
</EntitySetMapping>

```

Przykład

W następującym przykładzie pokazano **ScalarProperty** element używany do mapowania Wstawianie i usuwanie funkcji skojarzenia model koncepcyjny do procedur składowanych w bazie danych. Procedury składowane są deklarowane w modelu magazynu.

```

<AssociationSetMapping Name="CourseInstructor"
    TypeName="SchoolModel.CourseInstructor"
    StoreEntitySet="CourseInstructor">
    <EndProperty Name="Person">
        <ScalarProperty Name="PersonID" ColumnName="PersonID" />
    </EndProperty>
    <EndProperty Name="Course">
        <ScalarProperty Name="CourseID" ColumnName="CourseID" />
    </EndProperty>
    <ModificationFunctionMapping>
        <InsertFunction FunctionName="SchoolModel.Store.InsertCourseInstructor" >
            <EndProperty Name="Course">
                <ScalarProperty Name="CourseID" ParameterName="courseId"/>
            </EndProperty>
            <EndProperty Name="Person">
                <ScalarProperty Name="PersonID" ParameterName="instructorId"/>
            </EndProperty>
        </InsertFunction>
        <DeleteFunction FunctionName="SchoolModel.Store.DeleteCourseInstructor">
            <EndProperty Name="Course">
                <ScalarProperty Name="CourseID" ParameterName="courseId"/>
            </EndProperty>
            <EndProperty Name="Person">
                <ScalarProperty Name="PersonID" ParameterName="instructorId"/>
            </EndProperty>
        </DeleteFunction>
    </ModificationFunctionMapping>
</AssociationSetMapping>

```

Element UpdateFunction (MSL)

UpdateFunction elementu w specyfikacji języka (MSL) mapowania mapuje funkcji aktualizacji typu jednostki w modelu koncepcyjnym procedurę składowaną w bazie danych. Musi być zadeklarowany procedur skadowanych do modyfikacji, które funkcje są mapowane w modelu magazynu. Aby uzyskać więcej informacji zobacz Element — funkcja (SSDL).

NOTE

Jeśli nie mapują wszystkie trzy insert, update lub delete operacje typu jednostki, do procedur składowanych, niezamapowane operacji zakończy się niepowodzeniem, jeśli wykonywane w czasie wykonywania, i jest generowany, UpdateException.

UpdateFunction element może być elementem podrzędnym elementu ModificationFunctionMapping i stosowane do elementu EntityTypeMapping.

UpdateFunction element może mieć następujące elementy podrzędne:

- Punkt końcowy skojarzenia (zero lub więcej)
- ComplexProperty (zero lub więcej)
- ResultBinding (zero lub jeden)
- ScarlarProperty (zero lub więcej)

Odpowiednie atrybuty

W poniższej tabeli opisano atrybuty, które mogą być stosowane do **UpdateFunction** elementu.

NAZWA ATRYBUTU	JEST WYMAGANY	WARTOŚĆ
----------------	---------------	---------

NAZWA ATRYBUTU	JEST WYMAGANY	WARTOŚĆ
functionName	Tak	Przestrzeń nazw kwalifikowana nazwa procedury składowanej zmapowanej funkcji update. Procedura składowana musi być zadeklarowana w modelu magazynu.
RowsAffectedParameter	Nie	Nazwa parametru output, która zwraca liczbę uwzględnionych wierszy.

Przykład

Poniższy przykład jest oparty na modelu szkoły i pokazuje **UpdateFunction** element używany do mapowania funkcji aktualizacji **osoby** typu jednostki **UpdatePerson** Procedura składowana. **UpdatePerson** procedury składowanej jest zadeklarowana w modelu magazynu.

```
<EntityTypeMapping TypeName="SchoolModel.Person">
  <ModificationFunctionMapping>
    <InsertFunction FunctionName="SchoolModel.Store.InsertPerson">
      <ScalarProperty Name="EnrollmentDate"
                      ParameterName="EnrollmentDate" />
      <ScalarProperty Name="HireDate" ParameterName="HireDate" />
      <ScalarProperty Name="FirstName" ParameterName="FirstName" />
      <ScalarProperty Name="LastName" ParameterName="LastName" />
      <ResultBinding Name="PersonID" ColumnName="NewPersonID" />
    </InsertFunction>
    <UpdateFunction FunctionName="SchoolModel.Store.UpdatePerson">
      <ScalarProperty Name="EnrollmentDate"
                      ParameterName="EnrollmentDate"
                      Version="Current" />
      <ScalarProperty Name="HireDate" ParameterName="HireDate"
                      Version="Current" />
      <ScalarProperty Name="FirstName" ParameterName="FirstName"
                      Version="Current" />
      <ScalarProperty Name="LastName" ParameterName="LastName"
                      Version="Current" />
      <ScalarProperty Name="PersonID" ParameterName="PersonID"
                      Version="Current" />
    </UpdateFunction>
    <DeleteFunction FunctionName="SchoolModel.Store.DeletePerson">
      <ScalarProperty Name="PersonID" ParameterName="PersonID" />
    </DeleteFunction>
  </ModificationFunctionMapping>
</EntityTypeMapping>
```

Specyfikacja SSDL

13.09.2018 • 56 minutes to read • [Edit Online](#)

Język definicji schematu Store (SSDL) to oparty na standardzie XML język, który opisuje model magazynu w aplikacji Entity Framework.

W aplikacji Entity Framework metadanych modelu magazynu jest ładowany z pliku ssdl (napisanego w SSDL) do wystąpienia System.Data.Metadata.Edm.StoreItemCollection i są dostępne za pomocą metody Klasa System.Data.Metadata.Edm.MetadataWorkspace. Entity Framework używa magazynu metadanych modelu do przekształcania zapytania względem modelu koncepcyjnego polecenia specyficzne dla magazynu.

Entity Framework Designer (Projektant EF) przechowuje informacje o modelu magazynu w pliku edmx w czasie projektowania. W czasie komplikacji w Projektancie jednostki używa informacji w pliku edmx można utworzyć pliku ssdl, które są wymagane przez Entity Framework w czasie wykonywania.

Wersje SSDL są zróżnicowane według przestrzeni nazw XML.

WERSJA SSDL	NAMESPACE XML
SSDL v1	http://schemas.microsoft.com/ado/2006/04/edm/ssdl
SSDL v2	http://schemas.microsoft.com/ado/2009/02/edm/ssdl
SSDL v3	http://schemas.microsoft.com/ado/2009/11/edm/ssdl

Elementu Association (SSDL)

Skojarzenia element języka definicji schematu magazynu (SSDL) określa kolumny tabeli, które uczestniczą w ograniczeniu klucza obcego w bazie danych. Dwa elementy End wymagany element podrzędny Określ tabele końców skojarzenie i liczebność na każdym końcu. Opcjonalny element ReferentialConstraint określa głównym i zależnym końcach asocjacji, a także kolumn uczestniczących w programie. Jeśli nie **ReferentialConstraint** elementu, AssociationSetMapping element może służyć do określania mapowań kolumn dla skojarzenia.

Skojarzenia element może mieć następujących elementów podrzędnych (w podanej kolejności):

- Dokumentacja (zero lub jeden)
- Końcowy (dokładnie dwa)
- ReferentialConstraint (zero lub jeden)
- Elementów adnotacji (zero lub więcej)

Odpowiednie atrybuty

W poniższej tabeli opisano atrybuty, które mogą być stosowane do **scojarzenia** elementu.

NAZWA ATRYBUTU	JEST WYMAGANY	WARTOŚĆ
Nazwa	Tak	Nazwa odpowiedniego ograniczenie klucza obcego w bazie danych.

NOTE

Dowolna liczba atrybutów adnotacji (niestandardowe atrybuty XML) można stosować do **skojarzenia** elementu. Jednak atrybutów niestandardowych, które nie mogą należeć do przestrzeni nazw XML, który jest zarezerwowany dla SSDL. W pełni kwalifikowanej nazwy dowolne dwa atrybuty niestandardowe nie może być taka sama.

Przykład

W poniższym przykładzie przedstawiono **skojarzenia** element, który używa **ReferentialConstraint** elementu, aby określić kolumny, które uczestniczą w **klucza Obcego_CustomerOrders** ograniczenie klucza obcego:

```
<Association Name="FK_CustomerOrders">
    <End Role="Customers"
        Type="ExampleModel.Store.Customers" Multiplicity="1">
            <OnDelete Action="Cascade" />
        </End>
    <End Role="Orders"
        Type="ExampleModel.Store.Orders" Multiplicity="*" />
    <ReferentialConstraint>
        <Principal Role="Customers">
            <PropertyRef Name="CustomerId" />
        </Principal>
        <Dependent Role="Orders">
            <PropertyRef Name="CustomerId" />
        </Dependent>
    </ReferentialConstraint>
</Association>
```

Obiekt AssociationSet — Element (SSDL)

AssociationSet element język definicji schematu magazynu (SSDL) reprezentuje ograniczenie klucza obcego między dwiema tabelami w bazie danych. Kolumny tabeli, które uczestniczą w ograniczeniu klucza obcego są określone w elemencie **Association**. **Skojarzenia** element, który odpowiada danej **AssociationSet** element jest określony w **skojarzenia** atrybutu **AssociationSet** elementu.

SSDL zestawów skojarzeń są mapowane do zestawów skojarzeń CSDL przez **AssociationSetMapping** element. Jednak jeśli zestawu skojarzeń CSDL dla danego skojarzenia CSDL jest zdefiniowana za pomocą elementu **ReferentialConstraint** bez odpowiedniej **AssociationSetMapping** elementu jest konieczne. W takim przypadku **AssociationSetMapping** elementu, zostanie ona zastąpiona mapowania definiującego **ReferentialConstraint** elementu.

AssociationSet element może mieć następujących elementów podrzędnych (w podanej kolejności):

- Dokumentacja (zero lub jeden)
- END (zero lub dwóch)
- Elementów adnotacji (zero lub więcej)

Odpowiednie atrybuty

W poniższej tabeli opisano atrybuty, które mogą być stosowane do **AssociationSet** elementu.

NAZWA ATRYBUTU	JEST WYMAGANY	WARTOŚĆ
Nazwa	Tak	Nazwa ograniczenia klucza obcego, że skojarzenie ustawione reprezentuje.

NAZWA ATRYBUTU	JEST WYMAGANY	WARTOŚĆ
Skojarzenie	Tak	Nazwa skojarzenia, który definiuje kolumny, które uczestniczą w ograniczenie klucza obcego.

NOTE

Dowolna liczba atrybutów adnotacji (niestandardowe atrybuty XML) można stosować do **AssociationSet** elementu. Jednak atrybutów niestandardowych, które nie mogą należeć do przestrzeni nazw XML, który jest zarezerwowany dla SSDL. W pełni kwalifikowanej nazwy dowolne dwa atrybuty niestandardowe nie może być taka sama.

Przykład

W poniższym przykładzie przedstawiono **AssociationSet** elementu, który reprezentuje ograniczenie klucza obcego w bazie danych:

```
<AssociationSet Name="FK_CustomerOrders"
    Association="ExampleModel.Store.FK_CustomerOrders">
    <End Role="Customers" EntitySet="Customers" />
    <End Role="Orders" EntitySet="Orders" />
</AssociationSet>
```

Element CollectionType (SSDL)

CollectionType element język definicji schematu magazynu (SSDL) określa, że typ zwracany przez funkcję jest kolekcją. **CollectionType** element jest elementem podzewnętrznym elementu **ReturnType**. Typ kolekcji jest określony za pomocą elementu podzewnętrznego **RowType**:

NOTE

Dowolna liczba atrybutów adnotacji (niestandardowe atrybuty XML) można stosować do **CollectionType** elementu. Jednak atrybutów niestandardowych, które nie mogą należeć do przestrzeni nazw XML, który jest zarezerwowany dla SSDL. W pełni kwalifikowanej nazwy dowolne dwa atrybuty niestandardowe nie może być taka sama.

Przykład

W poniższym przykładzie pokazano funkcję, która używa **CollectionType** elementu, aby określić, że funkcja zwraca Kolekcja wierszy.

```
<Function Name="GetProducts" IsComposable="true" Schema="dbo">
    <ReturnType>
        <CollectionType>
            <RowType>
                <Property Name="ProductID" Type="int" Nullable="false" />
                <Property Name="CategoryID" Type="bigint" Nullable="false" />
                <Property Name="ProductName" Type="nvarchar" MaxLength="40" Nullable="false" />
                <Property Name="UnitPrice" Type="money" />
                <Property Name="Discontinued" Type="bit" />
            </RowType>
        </CollectionType>
    </ReturnType>
</Function>
```

Element CommandText (SSDL)

CommandText element w języku definicji schematu magazynu (SSDL) jest elementem podrzędnym elementu funkcji, która pozwala na zdefiniowanie instrukcji SQL, który jest wykonywany w bazie danych. **CommandText** elementu pozwala na dodawanie funkcji, która jest podobna do procedury składowanej w bazie danych, ale należy zdefiniować **CommandText** elementu w modelu magazynu.

CommandText elementu nie może mieć elementów podrzędnego. Treść **CommandText** elementu musi być prawidłową instrukcją SQL do podstawowej bazy danych.

Atrybuty nie są stosowane do **CommandText** elementu.

Przykład

W poniższym przykładzie przedstawiono **funkcja** element z elementem podrzędnym **CommandText** elementu. Udostępnianie **UpdateProductInOrder** działać jako metodę dla obiektu ObjectContext importując go do modelu koncepcyjnego.

```
<Function Name="UpdateProductInOrder" IsComposable="false">
  <CommandText>
    UPDATE Orders
    SET ProductId = @productId
    WHERE OrderId = @orderId;
  </CommandText>
  <Parameter Name="productId"
    Mode="In"
    Type="int"/>
  <Parameter Name="orderId"
    Mode="In"
    Type="int"/>
</Function>
```

Element DefiningQuery (SSDL)

DefiningQuery element języka definicji schematu magazynu (SSDL) pozwala na wykonanie instrukcji SQL bezpośrednio w bazie danych. **DefiningQuery** element jest najczęściej używany jak widok bazy danych, ale widok jest zdefiniowany w modelu pamięci masowej zamiast bazy danych. Widok zdefiniowany w **DefiningQuery** elementu mogą być mapowane na typ jednostki w modelu koncepcyjnym za pośrednictwem elementu obiektu EntitySetMapping. Te mapowania są przeznaczone tylko do odczytu.

Poniższa składnia SSDL pokazuje deklaracji **EntitySet** następuje **DefiningQuery** element, który zawiera zapytanie służy do pobierania tego widoku.

```
<Schema>
  <EntitySet Name="Tables" EntityType="Self.STable">
    <DefiningQuery>
      SELECT TABLE_CATALOG,
        'test' as TABLE_SCHEMA,
        TABLE_NAME
      FROM INFORMATION_SCHEMA.TABLES
    </DefiningQuery>
  </EntitySet>
</Schema>
```

Aby włączyć scenariuszach odczytu i zapisu za pośrednictwem widoków, można użyć procedur składowanych platformy Entity Framework. Widok źródła danych lub widoku SQL jednostki można użyć jako tabeli podstawowej dla pobierania danych i przetwarzania przez procedury składowane zmiany.

Möżesz użyć **DefiningQuery** elementu docelowego programu Microsoft SQL Server Compact 3.5. Chociaż program SQL Server Compact 3.5 nie obsługuje procedur przechowywanych, można zaimplementować podobne funkcje za pomocą **DefiningQuery** elementu. Jest innym miejscu, gdzie mogą być przydatne podczas tworzenia

procedur składowanych do pokonania niezgodność typów danych, używany w języku programowania i te źródła danych. Można napisać **DefiningQuery** które pobiera zestaw parametrów, a następnie wywołuje procedurę składowaną z innym zestawem parametrów, na przykład procedury przechowywanej, która powoduje usunięcie danych.

Element zależne (SSDL)

Zależne element język definicji schematu magazynu (SSDL) jest element podrzędnego do elementu **ReferentialConstraint**, który definiuje zależne koniec ograniczenie klucza obcego (nazywany także ograniczenie referencyjne). **Zależne** element określa kolumny (lub kolumny) w tabeli, która będzie odwoływać się do kolumny klucza podstawowego (lub kolumny). **PropertyRef** elementy określ kolumny, które są wywoływane. Główny element określa kolumny klucza podstawowego, które są przywoływane przez kolumny, które są określone w **zależne** elementu.

Zależne element może mieć następujących elementów podrzędnych (w podanej kolejności):

- **PropertyRef** (jeden lub więcej)
- Elementów adnotacji (zero lub więcej)

Odpowiednie atrybuty

W poniższej tabeli opisano atrybuty, które mogą być stosowane do **zależne** elementu.

NAZWA ATRYBUTU	JEST WYMAGANY	WARTOŚĆ
Rola	Tak	Taką samą wartość jak rola atrybut odpowiedni element End (jeśli jest używana); w przeciwnym razie nazwę tabeli, która zawiera kolumna źródłowa odwołania.

NOTE

Dowolna liczba atrybutów adnotacji (niestandardowe atrybuty XML) można stosować do **zależne** elementu. Jednak atrybutów niestandardowych, które nie mogą należeć do przestrzeni nazw XML, który jest zarezerwowany dla CSDL. W pełni kwalifikowanej nazwy dowolne dwa atrybuty niestandardowe nie może być taka sama.

Przykład

W poniższym przykładzie pokazano element Association używający **ReferentialConstraint** elementu, aby określić kolumny, które uczestniczą w **klucza Obcego_CustomerOrders** klucza obcego ograniczenie. **Zależne** element określa **CustomerId** kolumny **kolejności** tabeli jako zależne koniec tego ograniczenia.

```

<Association Name="FK_CustomerOrders">
  <End Role="Customers"
    Type="ExampleModel.Store.Customers" Multiplicity="1">
    <OnDelete Action="Cascade" />
  </End>
  <End Role="Orders"
    Type="ExampleModel.Store.Orders" Multiplicity="*" />
  <ReferentialConstraint>
    <Principal Role="Customers">
      <PropertyRef Name="CustomerId" />
    </Principal>
    <Dependent Role="Orders">
      <PropertyRef Name="CustomerId" />
    </Dependent>
  </ReferentialConstraint>
</Association>

```

Element documentation (SSDL)

Dokumentacji element język definicji schematu magazynu (SSDL) może służyć do Podaj informacje dotyczące obiektu, który jest zdefiniowany w elemencie rodzica.

Dokumentacji element może mieć następujących elementów podrzędnych (w podanej kolejności):

- **Podsumowanie:** Krótki opis elementu nadzecznego. (element zero lub jeden)
- **LongDescription:** rozbudowany opis elementu nadzecznego. (element zero lub jeden)

Odpowiednie atrybuty

Dowolna liczba atrybutów adnotacji (niestandardowe atrybuty XML) można stosować do **dokumentacji** elementu. Jednak atrybutów niestandardowych, które nie mogą należeć do przestrzeni nazw XML, który jest zarezerwowany dla CSDL. W pełni kwalifikowanej nazwy dowolne dwa atrybuty niestandardowe nie może być taka sama.

Przykład

W poniższym przykładzie przedstawiono **dokumentacji** element jako element podrzędny elementu EntityType.

```

<EntityType Name="Customers">
  <Documentation>
    <Summary>Summary here.</Summary>
    <LongDescription>Long description here.</LongDescription>
  </Documentation>
  <Key>
    <PropertyRef Name="CustomerId" />
  </Key>
  <Property Name="CustomerId" Type="int" Nullable="false" />
  <Property Name="Name" Type="nvarchar(max)" Nullable="false" />
</EntityType>

```

Element end (SSDL)

Zakończenia element język definicji schematu magazynu (SSDL) Określa tabelę i liczba wierszy na jednym końcu ograniczenie klucza obcego w bazie danych. **Zakończenia** element może być elementem podrzędnym elementu Association lub elementu AssociationSet. W każdym przypadku możliwe podrzędnych elementów i atrybuty stosowane są różne.

Końcowy Element jako element podrzędny elementu Association

Zakończenia — element (jako element podrzędny elementu **skojarzenia** elementu) Określa tabelę i liczba wierszy na końcu ograniczenie klucza obcego z **typu** i **Liczebność** odpowiednio atrybutów. Koniec ograniczenie

klucza obcego są zdefiniowane jako część skojarzenia SSDL; skojarzenie SSDL musi mieć dokładnie punktów końcowych.

Zakończenia element może mieć następujących elementów podrzędnych (w podanej kolejności):

- Dokumentacja (zero lub jeden element)
- OnDelete (zero lub jeden element)
- Elementów adnotacji (zero lub więcej elementów)

Odpowiednie atrybuty

W poniższej tabeli opisano atrybuty, które mogą być stosowane do **zakończenia** elementu, gdy jest elementem podrzędny elementu **skojarzenia** elementu.

NAZWA ATRYBUTU	JEST WYMAGANY	WARTOŚĆ
Typ	Tak	W pełni kwalifikowana nazwa zestawu jednostek SSDL, który znajduje się na końcu ograniczenie klucza obcego.
Rola	Nie	Wartość rolı atrybutu w elemencie jednostki albo zależnych od odpowiadającego im elementu ReferentialConstraint (jeśli są używane).
Liczebność	Tak	<p>1, od 0 do 1, lub * w zależności od liczby wierszy, które mogą być na końcu ograniczenie klucza obcego. 1 wskazuje, że dokładnie jeden wiersz istnieje na końcu ograniczenie klucza obcego. od 0 do 1 wskazuje, że istnieje zero lub jeden wiersz na końcu ograniczenie klucza obcego. * oznacza, że wartość zero, jeden lub więcej wierszy na końcu ograniczenie klucza obcego.</p>

NOTE

Dowolna liczba atrybutów adnotacji (niestandardowe atrybuty XML) można stosować do **zakończenia** elementu. Jednak atrybutów niestandardowych, które nie mogą należeć do przestrzeni nazw XML, który jest zarezerwowany dla CSDL. W pełni kwalifikowanej nazwy dowolne dwa atrybuty niestandardowe nie może być taka sama.

Przykład

W poniższym przykładzie przedstawiono **skojarzenia** element, który definiuje **klucza Obcego_CustomerOrders** ograniczenie klucza obcego. **Liczebność** wartościom określonym w każdym **zakończenia** elementu wskazują tę liczbę wierszy w **zamówienia** tabeli może być skojarzony z wiersza w **klientów** tabeli, ale tylko jeden wiersz w **klientów** tabeli może być skojarzony z wiersza w **zamówienia** tabeli. Ponadto **OnDelete** element wskazuje, że wszystkie wiersze, w **zamówienia** odwoływać się do danego wiersza w tabeli **klientów** tabeli zostanie usunięte, gdy wiersz **klientów** tabeli zostanie usunięty.

```

<Association Name="FK_CustomerOrders">
  <End Role="Customers"
    Type="ExampleModel.Store.Customers" Multiplicity="1">
    <OnDelete Action="Cascade" />
  </End>
  <End Role="Orders"
    Type="ExampleModel.Store.Orders" Multiplicity="*" />
  <ReferentialConstraint>
    <Principal Role="Customers">
      <PropertyRef Name="CustomerId" />
    </Principal>
    <Dependent Role="Orders">
      <PropertyRef Name="CustomerId" />
    </Dependent>
  </ReferentialConstraint>
</Association>

```

Końcowy Element jako element podrzędny elementu AssociationSet

Zakończenia — element (jako element podrzędny elementu **AssociationSet** elementu) Określa tabelę na jednym końcu ograniczenie klucza obcego w źródłowej bazie danych.

Zakończenia element może mieć następujących elementów podrzędnych (w podanej kolejności):

- Dokumentacja (zero lub jeden)
- Elementów adnotacji (zero lub więcej)

Odpowiednie atrybuty

W poniższej tabeli opisano atrybuty, które mogą być stosowane do **zakończenia** elementu, gdy jest elementem podrzędnym elementu **AssociationSet** elementu.

NAZWA ATRYBUTU	JEST WYMAGANY	WARTOŚĆ
Obiekt EntitySet	Tak	Nazwa zestawu jednostek SSDL, który znajduje się na końcu ograniczenie klucza obcego.
Rola	Nie	Wartość jednej z roli atrybuty określone w jednym zakończenia elementu odpowiednie skojarzenia.

NOTE

Dowolna liczba atrybutów adnotacji (niestandardowe atrybuty XML) można stosować do **zakończenia** elementu. Jednak atrybutów niestandardowych, które nie mogą należeć do przestrzeni nazw XML, który jest zarezerwowany dla CSDL. W pełni kwalifikowanej nazwy dowolne dwa atrybuty niestandardowe nie może być taka sama.

Przykład

W poniższym przykładzie przedstawiono **EntityContainer** element z **AssociationSet** elementu przy użyciu dwóch **zakończenia** elementy:

```

<EntityContainer Name="ExampleModelStoreContainer">
    <EntityType Name="Customers"
        EntityType="ExampleModel.Store.Customers"
        Schema="dbo" />
    <EntityType Name="Orders"
        EntityType="ExampleModel.Store.Orders"
        Schema="dbo" />
    <AssociationSet Name="FK_CustomerOrders"
        Association="ExampleModel.Store.FK_CustomerOrders">
        <End Role="Customers" EntitySet="Customers" />
        <End Role="Orders" EntitySet="Orders" />
    </AssociationSet>
</EntityContainer>

```

Element EntityContainer (SSDL)

EntityContainer element języka definicji schematu magazynu (SSDL) zawiera opis struktury źródła danych w aplikacji Entity Framework: zestawy jednostek SSDL (zdefiniowanymi w elementach EntitySet) reprezentują tabele w bazie danych, typy jednostek SSDL (zdefiniowany w elementach typu EntityType) reprezentują wierszy w tabeli i zestawów skojarzeń (zdefiniowany w obiekcie AssociationSet elementy) reprezentują ograniczenia klucza obcego w bazie danych. Kontener jednostek modelu magazynu jest mapowany na model koncepcyjny kontener jednostek, za pośrednictwem elementu w elemencie EntityContainerMapping.

EntityContainer element może mieć zero lub jeden elementów dokumentacji. Jeśli **dokumentacji** element jest obecny, musi poprzedzać wszystkie inne elementy podrzędne.

EntityContainer element może mieć zero lub więcej z następujących elementów podrzędnych (w podanej kolejności):

- Obiekt EntitySet
- Obiekt AssociationSet
- Elementów adnotacji

Odpowiednie atrybuty

W poniższej tabeli opisano atrybuty, które mogą być stosowane do **EntityContainer** elementu.

NAZWA ATRYBUTU	JEST WYMAGANY	WARTOŚĆ
Nazwa	Tak	Nazwa kontenera jednostek. Ta nazwa nie może zawierać kropek (.).

NOTE

Dowolna liczba atrybutów adnotacji (niestandardowe atrybuty XML) można stosować do **EntityContainer** elementu. Jednak atrybutów niestandardowych, które nie mogą należeć do przestrzeni nazw XML, który jest zarezerwowany dla SSDL. W pełni kwalifikowanej nazwy dowolne dwa atrybuty niestandardowe nie może być taka sama.

Przykład

W poniższym przykładzie przedstawiono **EntityContainer** element, który definiuje dwa zestawy jednostek i jedno skojarzenie zestawu. Należy zauważyć, że nazwy jednostki typu i skojarzenia typów są kwalifikowana przez nazwę przestrzeni nazw modelu koncepcyjnego.

```

<EntityContainer Name="ExampleModelStoreContainer">
    <EntityType Name="Customers"
        EntityType="ExampleModel.Store.Customers"
        Schema="dbo" />
    <EntityType Name="Orders"
        EntityType="ExampleModel.Store.Orders"
        Schema="dbo" />
    <AssociationSet Name="FK_CustomerOrders"
        Association="ExampleModel.Store.FK_CustomerOrders">
        <End Role="Customers" EntitySet="Customers" />
        <End Role="Orders" EntitySet="Orders" />
    </AssociationSet>
</EntityContainer>

```

Element EntitySet (SSDL)

EntitySet element języka definicji schematu magazynu (SSDL) reprezentuje tabelę lub widok w bazie danych. Element **EntityType** SSDL reprezentuje wiersz w tabeli lub widoku. **EntityType** atrybutu **EntitySet** element określa typ jednostki SSDL konkretnego, który reprezentuje wierszy w zestawie jednostek SSDL. Mapowanie między zestaw jednostek CSDL a SSDL zestaw jednostek jest określona w obiekcie **EntitySetMapping** elementu.

EntitySet element może mieć następujących elementów podrzędnych (w podanej kolejności):

- Dokumentacja (zero lub jeden element)
- DefiningQuery (zero lub jeden element)
- Elementów adnotacji

Odpowiednie atrybuty

W poniższej tabeli opisano atrybuty, które mogą być stosowane do **EntitySet** elementu.

NOTE

Niektóre atrybuty (niewymienione w tym) może być kwalifikowana za pomocą **przechowywania** aliasu. Te atrybuty są używane przez kreatora Model aktualizacji, podczas aktualizowania modelu.

NAZWA ATRYBUTU	JEST WYMAGANY	WARTOŚĆ
Nazwa	Tak	Nazwa zestawu jednostek.
Typ entityType	Tak	W pełni kwalifikowana nazwa typu jednostki, dla której zestaw jednostek zawiera wystąpienia.
Schemat	Nie	Schemat bazy danych.
Tabela	Nie	Tabela bazy danych.

NOTE

Dowolna liczba atrybutów adnotacji (niestandardowe atrybuty XML) można stosować do **EntitySet** elementu. Jednak atrybutów niestandardowych, które nie mogą należeć do przestrzeni nazw XML, który jest zarezerwowany dla SSDL. W pełni kwalifikowanej nazwy dowolne dwa atrybuty niestandardowe nie może być taka sama.

Przykład

W poniższym przykładzie przedstawiono **EntityContainer** element, który ma dwa **EntityType** elementy i jeden **AssociationSet** elementu:

```
<EntityContainer Name="ExampleModelStoreContainer">
    <EntityType Name="Customers"
        EntityType="ExampleModel.Store.Customers"
        Schema="dbo" />
    <EntityType Name="Orders"
        EntityType="ExampleModel.Store.Orders"
        Schema="dbo" />
    <AssociationSet Name="FK_CustomerOrders"
        Association="ExampleModel.Store.FK_CustomerOrders">
        <End Role="Customers" EntitySet="Customers" />
        <End Role="Orders" EntitySet="Orders" />
    </AssociationSet>
</EntityContainer>
```

Element EntityType (SSDL)

EntityType element język definicji schematu magazynu (SSDL) reprezentuje wiersz w tabeli lub widoku bazy danych. Element EntitySet SSDL reprezentuje tabelę lub widok, w którym występują wiersze. **EntityType** atrybutu **EntitySet** element określa typ jednostki SSDL konkretnego, który reprezentuje wierszy w zestawie jednostek SSDL. Mapowanie między typem encji SSDL i typ jednostki CSDL jest określony w elemencie **EntityTypeMapping**.

EntityType element może mieć następujących elementów podrzędnych (w podanej kolejności):

- Dokumentacja (zero lub jeden element)
- Klucz (zero lub jeden element)
- Elementów adnotacji

Odpowiednie atrybuty

W poniższej tabeli opisano atrybuty, które mogą być stosowane do **EntityType** elementu.

NAZWA ATRYBUTU	JEST WYMAGANY	WARTOŚĆ
Nazwa	Tak	Nazwa typu jednostki. Ta wartość jest zwykle taka sama jak nazwa tabeli, w którym typ jednostki reprezentuje wiersz. Ta wartość może zawierać nie kropki (.).

NOTE

Dowolna liczba atrybutów adnotacji (niestandardowe atrybuty XML) można stosować do **EntityType** elementu. Jednak atrybutów niestandardowych, które nie mogą należeć do przestrzeni nazw XML, który jest zarezerwowany dla SSDL. W pełni kwalifikowanej nazwy dowolne dwa atrybuty niestandardowe nie może być taka sama.

Przykład

W poniższym przykładzie przedstawiono **EntityType** elementu o dwie właściwości:

```

<EntityType Name="Customers">
  <Documentation>
    <Summary>Summary here.</Summary>
    <LongDescription>Long description here.</LongDescription>
  </Documentation>
  <Key>
    <PropertyRef Name="CustomerId" />
  </Key>
  <Property Name="CustomerId" Type="int" Nullable="false" />
  <Property Name="Name" Type="nvarchar(max)" Nullable="false" />
</EntityType>

```

Function — Element (SSDL)

Funkcja element języka definicji schematu magazynu (SSDL) określa procedury przechowywanej, która istnieje w bazie danych.

Funkcja element może mieć następujących elementów podrzędnych (w podanej kolejności):

- Dokumentacja (zero lub jeden)
- Parametr (zero lub więcej)
- CommandText (zero lub jeden)
- ReturnType (zero lub więcej)
- Elementów adnotacji (zero lub więcej)

Zwracana typ dla funkcji, należy określić albo **ReturnType** element lub **ReturnType** atrybutu (patrz poniżej), ale nie oba.

Procedury składowane, które są określone w modelu magazynu można zimportować do modelu koncepcyjnego aplikacji. Aby uzyskać więcej informacji, zobacz [wykonywanie zapytań za pomocą procedur składowanych](#).

Funkcja elementu może również służyć do definiowania funkcji niestandardowych w modelu magazynu.

Odpowiednie atrybuty

W poniższej tabeli opisano atrybuty, które mogą być stosowane do **funkcja** elementu.

NOTE

Niektóre atrybuty (niewymienione w tym) może być kwalifikowana za pomocą **przechowywania** aliasu. Te atrybuty są używane przez kreatora Model aktualizacji, podczas aktualizowania modelu.

NAZWA ATRYBUTU	JEST WYMAGANY	WARTOŚĆ
Nazwa	Tak	Nazwa procedury składowanej.
ReturnType	Nie	Zwracany typ procedury składowanej.
Aggregate	Nie	Wartość true , Jeśli procedura składowana ma zwracać wartości zagregowanej; w przeciwnym razie False .
Wbudowane	Nie	Wartość true , Jeśli funkcja jest wbudowaną ¹ funkcją; w przeciwnym razie False .

NAZWA ATRYBUTU	JEST WYMAGANY	WARTOŚĆ
StoreFunctionName	Nie	Nazwa procedury składowanej.
NiladicFunction	Nie	Wartość true , Jeśli funkcja znajduje się bez parametrów ² funkcji; False inaczej.
IsComposable	Nie	Wartość true , Jeśli funkcja jest konfigurowalna ³ funkcji; False inaczej.
ParameterTypeSemantics	Nie	Wyliczenie, które definiuje semantykę typów, używany do rozpoznawania przeciążenia funkcji. Wyliczenia jest zdefiniowany w manifeście dostawcy dla definicji funkcji. Wartość domyślna to AllowImplicitConversion .
Schemat	Nie	Nazwa schematu, w którym zdefiniowano procedury składowanej.

¹ wbudowana funkcja jest funkcją, która jest zdefiniowana w bazie danych. Aby uzyskać informacje na temat funkcji, które są zdefiniowane w modelu magazynu Zobacz Element CommandText (SSDL).

² funkcję bez parametrów jest funkcją, która przyjmuje żadnych parametrów i, gdy zostanie wywołana, nie wymaga nawiasów.

³ dwie funkcje są konfigurowalna, jeśli jedna funkcja może zwracać dane wejściowe dla innych funkcji.

NOTE

Dowolna liczba atrybutów adnotacji (niestandardowe atrybuty XML) można stosować do **funkcja** elementu. Jednak atrybutów niestandardowych, które nie mogą należeć do przestrzeni nazw XML, który jest zarezerwowany dla SSDL. W pełni kwalifikowanej nazwy dowolne dwa atrybuty niestandardowe nie może być taka sama.

Przykład

W poniższym przykładzie przedstawiono **funkcja** element, który odpowiada **UpdateOrderQuantity** procedury składowanej. Procedura składowana akceptuje dwa parametry i zwracać wartości.

```
<Function Name="UpdateOrderQuantity"
          Aggregate="false"
          BuiltIn="false"
          NiladicFunction="false"
          IsComposable="false"
          ParameterTypeSemantics="AllowImplicitConversion"
          Schema="dbo">
  <Parameter Name="orderId" Type="int" Mode="In" />
  <Parameter Name="newQuantity" Type="int" Mode="In" />
</Function>
```

Kluczowym elementem (SSDL)

Klucz element język definicji schematu magazynu (SSDL) reprezentuje klucz podstawowy tabeli w bazie danych.

Klucz jest elementem podrzędnym elementu **EntityType**, który reprezentuje wiersz w tabeli. Klucz podstawowy jest zdefiniowany w **klucz** elementu odwołując się do elementów właściwości, które są zdefiniowane na

EntityType elementu.

Klucz element może mieć następujących elementów podrzędnych (w podanej kolejności):

- PropertyRef (jeden lub więcej)
- Elementów adnotacji

Atrybuty nie są stosowane do **klucz** elementu.

Przykład

W poniższym przykładzie przedstawiono **EntityType** element z kluczem, który odwołuje się do jednej właściwości:

```
<EntityType Name="Customers">
  <Documentation>
    <Summary>Summary here.</Summary>
    <LongDescription>Long description here.</LongDescription>
  </Documentation>
  <Key>
    <PropertyRef Name="CustomerId" />
  </Key>
  <Property Name="CustomerId" Type="int" Nullable="false" />
  <Property Name="Name" Type="nvarchar(max)" Nullable="false" />
</EntityType>
```

Element OnDelete (SSDL)

OnDelete element język definicji schematu magazynu (SSDL) odzwierciedla zachowanie bazy danych, po usunięciu wiersza, który uczestniczy w ograniczenie klucza obcego. Jeśli ustawiono akcję **Cascade**, a następnie również zostaną usunięte wiersze, które odwołują się wiersz, który jest usuwany. Jeśli ustawiono akcję **Brak**, a następnie nie zostaną również usunięte wiersze, które odwołują się wiersz, który jest usuwany. **OnDelete** element jest elementem podrzędnym elementu End.

OnDelete element może mieć następujących elementów podrzędnych (w podanej kolejności):

- Dokumentacja (zero lub jeden)
- Elementów adnotacji (zero lub więcej)

Odpowiednie atrybuty

W poniższej tabeli opisano atrybuty, które mogą być stosowane do **OnDelete** elementu.

NAZWA ATRYBUTU	JEST WYMAGANY	WARTOŚĆ
Akcja	Tak	Kaskadowe lub Brak . (Wartość ograniczeniami jest prawidłowy, ale ma takie samo zachowanie jako Brak .)

NOTE

Dowolna liczba atrybutów adnotacji (niestandardowe atrybuty XML) można stosować do **OnDelete** elementu. Jednak atrybutów niestandardowych, które nie mogą należeć do przestrzeni nazw XML, który jest zarezerwowany dla SSDL. W pełni kwalifikowanej nazwy dowolne dwa atrybuty niestandardowe nie może być taka sama.

Przykład

W poniższym przykładzie przedstawiono **skojarzenia** element, który definiuje **klucza Obcego_CustomerOrders** ograniczenie klucza obcego. **OnDelete** element wskazuje, że wszystkie wiersze w **zamówienia** odwoływać się do danego wiersza w tabeli **klientów** tabeli zostanie usunięte, gdy wiersz **Klientów** tabeli zostanie usunięty.

```

<Association Name="FK_CustomerOrders">
  <End Role="Customers"
    Type="ExampleModel.Store.Customers" Multiplicity="1">
    <OnDelete Action="Cascade" />
  </End>
  <End Role="Orders"
    Type="ExampleModel.Store.Orders" Multiplicity="*" />
  <ReferentialConstraint>
    <Principal Role="Customers">
      <PropertyRef Name="CustomerId" />
    </Principal>
    <Dependent Role="Orders">
      <PropertyRef Name="CustomerId" />
    </Dependent>
  </ReferentialConstraint>
</Association>

```

Parameter — Element (SSDL)

Parametru język definicji schematu magazynu (SSDL) element jest elementem podrzędnym elementu funkcji, który określa parametry dla procedury przechowywanej w bazie danych.

Parametru element może mieć następujących elementów podrzędnych (w podanej kolejności):

- Dokumentacja (zero lub jeden)
- Elementów adnotacji (zero lub więcej)

Odpowiednie atrybuty

W poniższej tabeli opisano atrybuty, które mogą być stosowane do **parametru** elementu.

NAZWA ATRYBUTU	JEST WYMAGANY	WARTOŚĆ
Nazwa	Tak	Nazwa parametru.
Typ	Tak	Typ parametru.
Tryb	Nie	W, sie, lub InOut w zależności od tego, czy parametr jest danych wejściowych, danych wyjściowych lub parametr input/output.
Element maxLength	Nie	Maksymalna długość parametru.
Precyzja	Nie	Dokładność parametru.
Skala	Nie	Skala parametru.
SRID	Nie	Identyfikator odwołania przestrzennego systemu. Prawidłowy tylko dla parametrów typów przestrzennych. Aby uzyskać więcej informacji, zobacz SRID i SRID (SQL Server) .

NOTE

Dowolna liczba atrybutów adnotacji (niestandardowe atrybuty XML) można stosować do **parametru** elementu. Jednak atrybutów niestandardowych, które nie mogą należeć do przestrzeni nazw XML, który jest zarezerwowany dla SSDL. W pełni kwalifikowanej nazwy dowolne dwa atrybuty niestandardowe nie może być taka sama.

Przykład

W poniższym przykładzie przedstawiono **funkcja** element, który ma dwa **parametru** elementy, które określają parametry wejściowe:

```
<Function Name="UpdateOrderQuantity"
    Aggregate="false"
    BuiltIn="false"
    NiladicFunction="false"
    IsComposable="false"
    ParameterTypeSemantics="AllowImplicitConversion"
    Schema="dbo">
    <Parameter Name="orderId" Type="int" Mode="In" />
    <Parameter Name="newQuantity" Type="int" Mode="In" />
</Function>
```

Element jednostki (SSDL)

Jednostki element język definicji schematu magazynu (SSDL) jest element podrzędny do elementu **ReferentialConstraint**, który definiuje główny koniec ograniczenie klucza obcego (nazywany także ograniczenie referencyjne). **Jednostki** element określa kolumny klucza podstawowego (lub kolumny) w tabeli, która odwołuje się do niej innej kolumny (lub kolumny). **PropertyRef** elementy Określ kolumny, które są wywoływanie. Element **zależne** Określa kolumny, które odwołują się kolumny klucza podstawowego, które są określone w **jednostki** elementu.

Jednostki element może mieć następujących elementów podrzędnych (w podanej kolejności):

- **PropertyRef** (jeden lub więcej)
- Elementów adnotacji (zero lub więcej)

Odpowiednie atrybuty

W poniższej tabeli opisano atrybuty, które mogą być stosowane do **jednostki** elementu.

NAZWA ATRYBUTU	JEST WYMAGANY	WARTOŚĆ
Rola	Tak	Taką samą wartość jak rola atrybut odpowiedni element End (jeśli jest używana); w przeciwnym razie nazwę tabeli, która zawiera odwołuje się do kolumny.

NOTE

Dowolna liczba atrybutów adnotacji (niestandardowe atrybuty XML) można stosować do **jednostki** elementu. Jednak atrybutów niestandardowych, które nie mogą należeć do przestrzeni nazw XML, który jest zarezerwowany dla CSDL. W pełni kwalifikowanej nazwy dowolne dwa atrybuty niestandardowe nie może być taka sama.

Przykład

W poniższym przykładzie pokazano element Association używający **ReferentialConstraint** elementu, aby

określić kolumny, które uczestniczą w **klucza Obcego_CustomerOrders** klucza obcego ograniczenie. **Jednostki** element Określa **CustomerId** kolumny **klienta** tabeli jako główny koniec tego ograniczenia.

```
<Association Name="FK_CustomerOrders">
  <End Role="Customers"
    Type="ExampleModel.Store.Customers" Multiplicity="1">
    <OnDelete Action="Cascade" />
  </End>
  <End Role="Orders"
    Type="ExampleModel.Store.Orders" Multiplicity="*" />
<ReferentialConstraint>
  <Principal Role="Customers">
    <PropertyRef Name="CustomerId" />
  </Principal>
  <Dependent Role="Orders">
    <PropertyRef Name="CustomerId" />
  </Dependent>
</ReferentialConstraint>
</Association>
```

Property — Element (SSDL)

Właściwość element język definicji schematu magazynu (SSDL) reprezentuje kolumnę w tabeli w bazie danych. **Właściwość** elementy są elementami podrzędnymi typu **EntityType** elementy, które reprezentują wierszy w tabeli. Każdy **właściwość** elementu zdefiniowanego w **EntityType** element reprezentuje kolumnę.

A **właściwość** elementu nie może mieć żadnych elementów podrzędnych.

Odpowiednie atrybuty

W poniższej tabeli opisano atrybuty, które mogą być stosowane do **właściwość** elementu.

NAZWA ATRYBUTU	JEST WYMAGANY	WARTOŚĆ
Nazwa	Tak	Nazwa odpowiednią kolumnę.
Typ	Tak	Typ odpowiednią kolumnę.
Dopuszcza wartości null	Nie	Wartość true , (wartość domyślna) lub False w zależności od tego, czy odpowiednia kolumna może mieć wartości null.
defaultValue	Nie	Wartość domyślna w kolumnie.
Element maxLength	Nie	Maksymalna długość odpowiednią kolumnę.
Wartości	Nie	Wartość true , lub False w zależności od tego, czy odpowiadająca wartość w kolumnie będą przechowywane jako ciąg znaków o stałej długości.
Precyzja	Nie	Dokładność odpowiednią kolumnę.
Skala	Nie	Skala odpowiednią kolumnę.

NAZWA ATRYBUTU	JEST WYMAGANY	WARTOŚĆ
Unicode	Nie	Wartość true , lub False w zależności od tego, czy odpowiadająca wartość w kolumnie będą przechowywane jako ciąg Unicode.
Sortowanie	Nie	Ciąg, który określa kolejność sortowania, które ma być używany w źródle danych.
SRID	Nie	Identyfikator odwołania przestrzennego systemu. Prawidłowy tylko w przypadku właściwości typów przestrzennych. Aby uzyskać więcej informacji, zobacz SRID i SRID (SQL Server) .
Element StoreGeneratedPattern	Nie	Brak, tożsamości (jeśli odpowiadająca wartość w kolumnie jest tożsamością, która jest generowana w bazie danych) lub obliczane (jeśli odpowiadająca wartość w kolumnie jest obliczana w bazie danych). Nie obowiązuje dla właściwości RowType.

NOTE

Dowolna liczba atrybutów adnotacji (niestandardowe atrybuty XML) można stosować do **właściwość** elementu. Jednak atrybutów niestandardowych, które nie mogą należeć do przestrzeni nazw XML, który jest zarezerwowany dla SSDL. W pełni kwalifikowanej nazwy dowolne dwa atrybuty niestandardowe nie może być taka sama.

Przykład

W poniższym przykładzie przedstawiono **EntityType** element z dwóch podrzędnych **właściwość** elementy:

```
<EntityType Name="Customers">
  <Documentation>
    <Summary>Summary here.</Summary>
    <LongDescription>Long description here.</LongDescription>
  </Documentation>
  <Key>
    <PropertyRef Name="CustomerId" />
  </Key>
  <Property Name="CustomerId" Type="int" Nullable="false" />
  <Property Name="Name" Type="nvarchar(max)" Nullable="false" />
</EntityType>
```

Element PropertyRef (SSDL)

PropertyRef element język definicji schematu magazynu (SSDL) odwołuje się do właściwości zdefiniowany dla elementu **EntityType**, aby wskazać, że właściwość wykona jedną z następujących ról:

- Być częścią klucza podstawowego w tabeli, która **EntityType** reprezentuje. Co najmniej jeden **PropertyRef** elementy mogą być używane do definiowania klucza podstawowego. Aby uzyskać więcej informacji zobacz element klucza.
- Być zakończenia zależnych lub jednostki z ograniczeniem referencyjnym. Aby uzyskać więcej informacji zobacz **ReferentialConstraint** element.

PropertyRef element może mieć tylko następujących elementów podrzędnych:

- Dokumentacja (zero lub jeden)
- Elementów adnotacji

Odpowiednie atrybuty

W poniższej tabeli opisano atrybuty, które mogą być stosowane do **PropertyRef** elementu.

NAZWA ATRYBUTU	JEST WYMAGANY	WARTOŚĆ
Nazwa	Tak	Nazwa właściwości, której dotyczy odwołanie.

NOTE

Dowolna liczba atrybutów adnotacji (niestandardowe atrybuty XML) można stosować do **PropertyRef** elementu. Jednak atrybutów niestandardowych, które nie mogą należeć do przestrzeni nazw XML, który jest zarezerwowany dla CSDL. W pełni kwalifikowanej nazwy dowolne dwa atrybuty niestandardowe nie może być taka sama.

Przykład

W poniższym przykładzie przedstawiono **PropertyRef** element używany do definiowania klucza podstawowego, odwołując się do właściwości, która jest zdefiniowana na **EntityType** elementu.

```
<EntityType Name="Customers">
  <Documentation>
    <Summary>Summary here.</Summary>
    <LongDescription>Long description here.</LongDescription>
  </Documentation>
  <Key>
    <PropertyRef Name="CustomerId" />
  </Key>
  <Property Name="CustomerId" Type="int" Nullable="false" />
  <Property Name="Name" Type="nvarchar(max)" Nullable="false" />
</EntityType>
```

Element ReferentialConstraint (SSDL)

ReferentialConstraint element język definicji schematu magazynu (SSDL) reprezentuje ograniczenie klucza obcego (nazywane również ograniczenia integralności referencyjnej) w bazie danych. Kończy się głównym i zależnym ograniczenia są odpowiednio określone przez elementy podrzędne jednostki i zależnych od ustawień lokalnych. Kolumny, które uczestniczą w głównym i zależnym kończy się są przywoływanie z elementami **PropertyRef**.

ReferentialConstraint element jest podrzędnym elementem opcjonalnym elementu **Association**. Jeśli **ReferentialConstraint** element nie jest używany do mapowania ograniczenie klucza obcego, który jest określony w **skojarzenia AssociationSetMapping** element musi być użyty w tym elemencie.

ReferentialConstraint element może mieć następujące elementy podrzędne:

- Dokumentacja (zero lub jeden)
- Podmiot zabezpieczeń (dokładnie jeden)
- Zależnych od ustawień lokalnych (dokładnie jeden)
- Elementów adnotacji (zero lub więcej)

Odpowiednie atrybuty

Dowolna liczba atrybutów adnotacji (niestandardowe atrybuty XML) można stosować do **ReferentialConstraint** elementu. Jednak atrybutów niestandardowych, które nie mogą należeć do przestrzeni nazw XML, który jest zarezerwowany dla SSDL. W pełni kwalifikowanej nazwy dowolne dwa atrybuty niestandardowe nie może być taka sama.

Przykład

W poniższym przykładzie przedstawiono **skojarzenia** element, który używa **ReferentialConstraint** elementu, aby określić kolumny, które uczestniczą w **klucza Obcego_CustomerOrders** ograniczenie klucza obcego:

```
<Association Name="FK_CustomerOrders">
  <End Role="Customers"
    Type="ExampleModel.Store.Customers" Multiplicity="1">
    <OnDelete Action="Cascade" />
  </End>
  <End Role="Orders"
    Type="ExampleModel.Store.Orders" Multiplicity="*" />
  <ReferentialConstraint>
    <Principal Role="Customers">
      <PropertyRef Name="CustomerId" />
    </Principal>
    <Dependent Role="Orders">
      <PropertyRef Name="CustomerId" />
    </Dependent>
  </ReferentialConstraint>
</Association>
```

Element ReturnType (SSDL)

ReturnType element języka definicji schematu magazynu (SSDL) określa typ zwracany dla funkcji, która jest zdefiniowana w **funkcja** elementu. Można również określić typ zwracany z funkcji **ReturnType** atrybutu.

Zwracany typ funkcji jest określony za pomocą **typu** atrybutu lub **ReturnType** elementu.

ReturnType element może mieć następujące elementy podrzędne:

- Typ CollectionType (po jednym)

NOTE

Dowolna liczba atrybutów adnotacji (niestandardowe atrybuty XML) można stosować do **ReturnType** elementu. Jednak atrybutów niestandardowych, które nie mogą należeć do przestrzeni nazw XML, który jest zarezerwowany dla SSDL. W pełni kwalifikowanej nazwy dowolne dwa atrybuty niestandardowe nie może być taka sama.

Przykład

W poniższym przykładzie użyto **funkcja** zwracającego Kolekcja wierszy.

```

<Function Name="GetProducts" IsComposable="true" Schema="dbo">
  <ReturnType>
    <CollectionType>
      <RowType>
        <Property Name="ProductID" Type="int" Nullable="false" />
        <Property Name="CategoryID" Type="bigint" Nullable="false" />
        <Property Name="ProductName" Type="nvarchar" MaxLength="40" Nullable="false" />
        <Property Name="UnitPrice" Type="money" />
        <Property Name="Discontinued" Type="bit" />
      </RowType>
    </CollectionType>
  </ReturnType>
</Function>

```

Element RowType (SSDL)

A **RowType** element język definicji schematu magazynu (SSDL) definiuje strukturę nienazwane jako zwracany typ funkcji zdefiniowanych w magazynie.

A **RowType** element jest elementem podrzędnym **CollectionType** elementu:

A **RowType** element może mieć następujące elementy podrzędne:

- Właściwości (jeden lub więcej)

Przykład

W poniższym przykładzie pokazano funkcję magazynu, która używa **CollectionType** elementu, aby określić, że funkcja zwraca Kolekcja wierszy (jak określono w **RowType** elementu).

```

<Function Name="GetProducts" IsComposable="true" Schema="dbo">
  <ReturnType>
    <CollectionType>
      <RowType>
        <Property Name="ProductID" Type="int" Nullable="false" />
        <Property Name="CategoryID" Type="bigint" Nullable="false" />
        <Property Name="ProductName" Type="nvarchar" MaxLength="40" Nullable="false" />
        <Property Name="UnitPrice" Type="money" />
        <Property Name="Discontinued" Type="bit" />
      </RowType>
    </CollectionType>
  </ReturnType>
</Function>

```

Element schematu (SSDL)

Schematu element w języku definicji schematu magazynu (SSDL) jest głównym elementem definicji modelu magazynu. Zawiera definicje dla obiektów, funkcji i kontenerów, które tworzą model magazynu.

Schematu element może zawierać zero lub więcej z następujących elementów podrzędnych:

- Skojarzenie
- Typ entityType
- Obiekt EntityContainer
- Funkcja

Schematu element używa **Namespace** atrybut do definiowania przestrzeni nazw dla obiektów typu i skojarzenia jednostki w modelu magazynu. W przestrzeni nazw nie dwa obiekty mogą mieć takiej samej nazwie.

Przestrzeń nazw modelu magazynu różni się od przestrzeni nazw XML **schematu** elementu. Przestrzeń nazw

modelu magazynu (zgodnie z definicją **Namespace** atrybutu) to logiczny kontener przeznaczony dla typów jednostek i typów skojarzenia. Przestrzeń nazw XML (wskaazywanym przez **xmlns** atrybutu) z **schematu** element jest domyślny obszar nazw dla elementów podległych i atrybutów **schematu** elementu. Obszary nazw XML w postaci <http://schemas.microsoft.com/ado/YYYY/MM/edm/ssdl> (gdzie RRRR i MM stanowią rok i miesiąc odpowiednio) są zarezerwowane dla SSDL. Niestandardowe elementy i atrybuty nie mogą być w przestrzeni nazw, które mają postać.

Odpowiednie atrybuty

W poniższej tabeli opisano atrybuty mogą być stosowane do **schematu** elementu.

NAZWA ATRYBUTU	JEST WYMAGANY	WARTOŚĆ
Namespace	Tak	Przestrzeń nazw modelu magazynu. Wartość Namespace atrybut jest używany w celu utworzenia w pełni kwalifikowana nazwa typu. Na przykład jeśli EntityType o nazwie <i>klienta</i> znajduje się w przestrzeni nazw ExampleModel.Store, a następnie w pełni kwalifikowana nazwa EntityType jest ExampleModel.Store.Customer. Nie można użyć następujących ciągów jako wartość pozycji Namespace atrybut: systemu , przejściowy , lub Edm . Wartość Namespace atrybut nie może być taka sama jak wartość Namespace atrybutu w elemencie CSDL Schema.
Alias	Nie	Identyfikator używany zamiast nazwy przestrzeni nazw. Na przykład jeśli EntityType o nazwie <i>klienta</i> znajduje się w przestrzeni nazw ExampleModel.Store i wartość Alias atrybut jest <i>StorageModel</i> , wówczas można użyć StorageModel.Customer jako w pełni kwalifikowana nazwa typu EntityType .
Dostawcy	Tak	Dostawca danych.
ProviderManifestToken	Tak	Token, który wskazuje dostawcy, które manifest dostawcy, aby powrócić. Nie format tokenu jest zdefiniowany. Wartości dla tokenu są definiowane przez dostawcę. Uzyskać informacji dotyczących tokenów manifestu dostawcy programu SQL Server zobacz SqlClient programu Entity Framework.

Przykład

W poniższym przykładzie przedstawiono **schematu** element, który zawiera **EntityContainer** element, dwa **EntityType** elementów, a drugi **skojarzenia** elementu.

```
<Schema Namespace="ExampleModel.Store"
  Alias="Self" Provider="System.Data.SqlClient"
  ProviderManifestToken="2008"
  xmlns="http://schemas.microsoft.com/ado/2009/11/edm/ssdl">
  <EntityContainer Name="ExampleModelStoreContainer">
    <EntitySet Name="Customers">
```

```

        EntityType="ExampleModel.Store.Customers"
        Schema="dbo" />
<EntityType Name="Orders"
        EntityType="ExampleModel.Store.Orders"
        Schema="dbo" />
<AssociationSet Name="FK_CustomerOrders"
        Association="ExampleModel.Store.FK_CustomerOrders">
    <End Role="Customers" EntitySet="Customers" />
    <End Role="Orders" EntitySet="Orders" />
</AssociationSet>
</EntityContainer>
<EntityType Name="Customers">
    <Documentation>
        <Summary>Summary here.</Summary>
        <LongDescription>Long description here.</LongDescription>
    </Documentation>
    <Key>
        <PropertyRef Name="CustomerId" />
    </Key>
    <Property Name="CustomerId" Type="int" Nullable="false" />
    <Property Name="Name" Type="nvarchar(max)" Nullable="false" />
</EntityType>
<EntityType Name="Orders" xmlns:c="http://CustomNamespace">
    <Key>
        <PropertyRef Name="OrderId" />
    </Key>
    <Property Name="OrderId" Type="int" Nullable="false"
        c:CustomAttribute="someValue"/>
    <Property Name="ProductId" Type="int" Nullable="false" />
    <Property Name="Quantity" Type="int" Nullable="false" />
    <Property Name="CustomerId" Type="int" Nullable="false" />
    <c:CustomElement>
        Custom data here.
    </c:CustomElement>
</EntityType>
<Association Name="FK_CustomerOrders">
    <End Role="Customers"
        Type="ExampleModel.Store.Customers" Multiplicity="1">
        <OnDelete Action="Cascade" />
    </End>
    <End Role="Orders"
        Type="ExampleModel.Store.Orders" Multiplicity="*" />
<ReferentialConstraint>
    <Principal Role="Customers">
        <PropertyRef Name="CustomerId" />
    </Principal>
    <Dependent Role="Orders">
        <PropertyRef Name="CustomerId" />
    </Dependent>
</ReferentialConstraint>
</Association>
<Function Name="UpdateOrderQuantity"
        Aggregate="false"
        BuiltIn="false"
        NiladicFunction="false"
        IsComposable="false"
        ParameterTypeSemantics="AllowImplicitConversion"
        Schema="dbo">
    <Parameter Name="orderId" Type="int" Mode="In" />
    <Parameter Name="newQuantity" Type="int" Mode="In" />
</Function>
<Function Name="UpdateProductInOrder" IsComposable="false">
    <CommandText>
        UPDATE Orders
        SET ProductId = @productId
        WHERE OrderId = @orderId;
    </CommandText>
    <Parameter Name="productId"
        Mode="In" />
</Function>

```

```
        Mode="In"
        Type="int"/>
    <Parameter Name="orderId"
        Mode="In"
        Type="int"/>
</Function>
</Schema>
```

Atrybuty adnotacji

Atrybuty adnotacji w język definicji schematu magazynu (SSDL) są niestandardowych atrybutów XML w modelu magazynu, które zapewniają dodatkowe metadane na temat elementów w modelu magazynu. Oprócz prawidłowej struktury XML, obowiązują następujące ograniczenia adnotacji atrybutów:

- Atrybuty adnotacji nie może być w przestrzeni nazw XML, który jest zarezerwowany dla SSDL.
- W pełni kwalifikowanej nazwy wszelkie atrybuty dwóch adnotacji nie może być taka sama.

Więcej niż jeden atrybut adnotacji można stosować do danego elementu SSDL. W czasie wykonywania przy użyciu klas w przestrzeni nazw System.Data.Metadata.Edm możliwy jest metadanych elementów adnotacji.

Przykład

W poniższym przykładzie pokazano element EntityType, który ma atrybut adnotacja zastosowana do **OrderId** właściwości. Przykład pokazują również dodawane do elementu adnotacji **EntityType** elementu.

```
<EntityType Name="Orders" xmlns:c="http://CustomNamespace">
    <Key>
        <PropertyRef Name="OrderId" />
    </Key>
    <Property Name="OrderId" Type="int" Nullable="false"
        c:CustomAttribute="someValue"/>
    <Property Name="ProductId" Type="int" Nullable="false" />
    <Property Name="Quantity" Type="int" Nullable="false" />
    <Property Name="CustomerId" Type="int" Nullable="false" />
    <c:CustomElement>
        Custom data here.
    </c:CustomElement>
</EntityType>
```

Elementów adnotacji (SSDL)

Elementy adnotacji w język definicji schematu magazynu (SSDL) są niestandardowe elementy XML w modelu magazynu, które zapewniają dodatkowe metadane na temat modelu magazynu. Oprócz prawidłowej struktury XML, elementów adnotacji obowiązują następujące ograniczenia:

- Elementów adnotacji nie może być w przestrzeni nazw XML, który jest zarezerwowany dla SSDL.
- W pełni kwalifikowanej nazwy dowolne elementy dwóch adnotacji nie może być taka sama.
- Adnotacja elementów musi występować po wszystkich innych elementów podrzędnych danego elementu SSDL.

Więcej niż jeden element adnotacji może być elementem podrzędnym danego elementu SSDL. Począwszy od programu .NET Framework w wersji 4, metadanych elementów adnotacji są dostępne w czasie wykonywania przy użyciu klas w przestrzeni nazw System.Data.Metadata.Edm.

Przykład

W poniższym przykładzie pokazano element EntityType, który ma element adnotacji (**CustomElement**). W przykładzie pokazano również zastosować atrybut adnotacji **OrderId** właściwości.

```

<EntityType Name="Orders" xmlns:c="http://CustomNamespace">
  <Key>
    <PropertyRef Name="OrderId" />
  </Key>
  <Property Name="OrderId" Type="int" Nullable="false"
    c:CustomAttribute="someValue"/>
  <Property Name="ProductId" Type="int" Nullable="false" />
  <Property Name="Quantity" Type="int" Nullable="false" />
  <Property Name="CustomerId" Type="int" Nullable="false" />
  <c:CustomElement>
    Custom data here.
  </c:CustomElement>
</EntityType>

```

Zestawy reguł (SSDL)

Aspekty w języku definicji schematu magazynu (SSDL) reprezentują ograniczenia dotyczące typów kolumn, które są określone w elementach właściwości. Zestawy reguł są implementowane jako atrybuty XML w **właściwość** elementów.

W poniższej tabeli opisano aspekty, które są obsługiwane przez SSDL:

ZESTAW REGUŁ	OPIS
Sortowanie	Określa kolejność sortowania (lub sekwencji sortowania) do użycia podczas przeprowadzania porównania i kolejność operacji na wartościach właściwości.
Wartości	Określa, czy długość wartości kolumny mogą się różnić.
Element maxLength	Określa maksymalną długość wartości kolumny.
Precyzja	Dla właściwości typu dziesiętna , określa liczbę cyfr, może mieć wartości właściwości. Dla właściwości typu czasu , daty/godziny , i DateTimeOffset , określa liczbę cyfr ułamkowych części sekundy w wartości kolumny.
Skala	Określa liczbę cyfr po prawej stronie przecinka dziesiętnego dla wartości kolumny.
Unicode	Wskazuje, czy wartość kolumny jest zapisywana w formacie Unicode.

Definiowanie zapytania — projektancie platformy EF

13.09.2018 • 9 minutes to read • [Edit Online](#)

W tym instruktażu przedstawiono sposób dodawania, definiując kwerendy i odpowiednia jednostka typu do modelu, używając projektancie platformy EF. Definiowanie zapytania jest najczęściej używany do zapewnia funkcje podobne do dostarczony przez widok bazy danych, ale widok jest zdefiniowany w modelu, a nie bazy danych. Definiowanie zapytań pozwala wykonać instrukcję SQL, który jest określony w **DefiningQuery** element z pliku edmx. Aby uzyskać więcej informacji, zobacz **DefiningQuery** w [Specyfikacja SSDL](#).

Korzystając z Definiowanie zapytań, należy zdefiniować typ jednostki w modelu. Typ jednostki jest używany do powierzchni dane udostępniane przez definiowanie zapytań. Należy pamiętać, że udostępniane za pośrednictwem tego typu jednostki danych tylko do odczytu.

Nie można wykonać zapytań sparametryzowanych jako Definiowanie zapytań. Jednak dane można zaktualizować przez mapowanie insert, update i delete funkcji typu jednostki który uwypukli najistotniejsze dane do procedur składowanych. Aby uzyskać więcej informacji, zobacz [wstawiania, aktualizowania i usuwania przy użyciu procedur składowanych](#).

W tym temacie przedstawiono sposób wykonywania następujących zadań.

- Dodaj zapytanie definiujące
- Dodaj typ jednostki do modelu
- Definiowanie zapytania do typu jednostki mapy

Wymagania wstępne

W celu wykonania instrukcji w tym przewodniku potrzebne są następujące elementy:

- Najnowszą wersję programu Visual Studio.
- [Przykładowej bazy danych School](#).

Konfigurowanie projektu

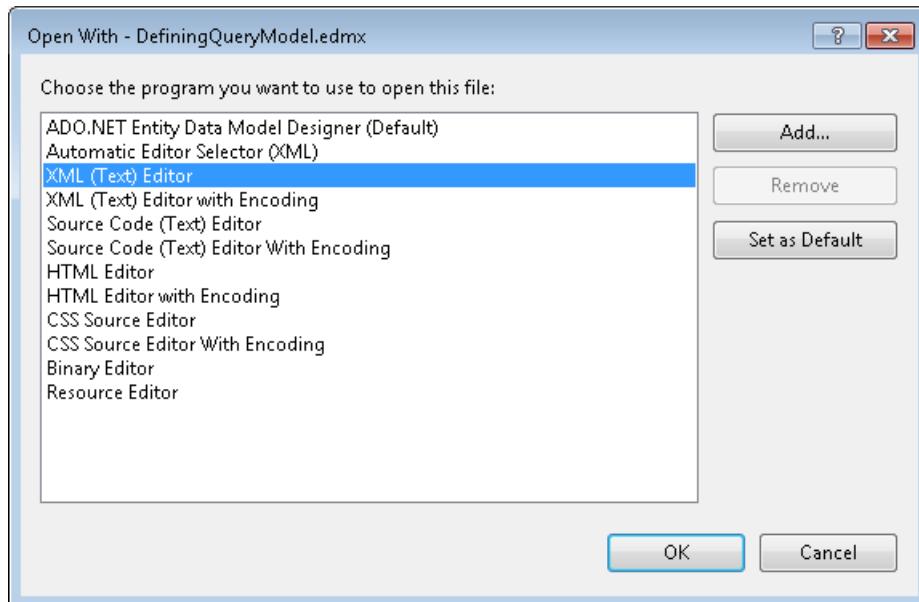
Ten przewodnik korzysta z programu Visual Studio 2012 lub nowszego.

- Otwórz program Visual Studio.
- Na **pliku** menu wskaż **New**, a następnie kliknij przycisk **projektu**.
- W okienku po lewej stronie kliknij **Visual C#**, a następnie wybierz pozycję **aplikację Konsolową** szablonu.
- Wprowadź **DefiningQuerySample** jako nazwę projektu i kliknij przycisk **OK**.

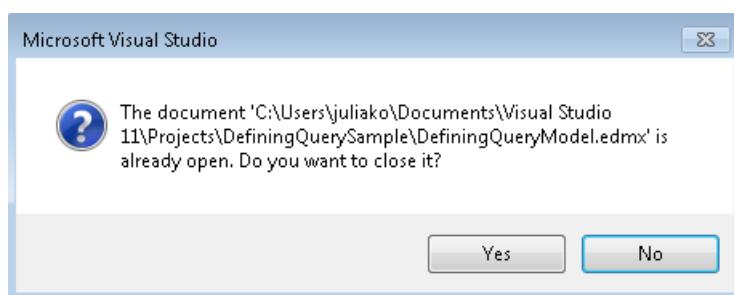
Tworzenie modelu, w oparciu o bazę danych School

- Kliknij prawym przyciskiem myszy nazwę projektu w Eksploratorze rozwiązań, wskaż opcję **Dodaj**, a następnie kliknij przycisk **nowy element**.
- Wybierz **danych** z menu po lewej stronie, a następnie wybierz pozycję **ADO.NET Entity Data Model** w okienku szablonów.
- Wprowadź **DefiningQueryModel.edmx** nazwę pliku, a następnie kliknij przycisk **Dodaj**.
- W oknie dialogowym Wybierz zawartość modelu, wybierz **Generuj z bazy danych**, a następnie kliknij przycisk **dalej**.

- Kliknij przycisk nowe połączenie. W oknie dialogowym właściwości połączenia, wprowadź nazwę serwera (na przykład **(localdb)\mssqllocaldb**), wybierz metodę uwierzytelniania, wpisz **School** nazwy bazy danych, a następnie Kliknij przycisk **OK**. Okno dialogowe Wybierz połączenie danych jest aktualizowana ustawienie połączenia bazy danych.
- W oknie dialogowym Wybierz obiekty bazy danych sprawdź **tabele** węzła. Spowoduje to dodanie wszystkich tabel do **School** modelu.
- Kliknij przycisk **Zakończ**.
- W Eksploratorze rozwiązań kliknij prawym przyciskiem myszy **DefiningQueryModel.edmx** plik i wybierz **Otwórz za pomocą...**.
- Wybierz **Edytor (tekstu) XML**.



- Kliknij przycisk **tak** Jeśli zostanie wyświetlony monit z następującym komunikatem:



Dodaj zapytanie definiujące

W tym kroku, które zostaną użyte zapytanie edytora XML, aby dodać definiujące, a następnie wpisz jednostki SSDL części pliku edmx.

- Dodaj **EntitySet** elementu SSDL części pliku edmx (wiersz 5 do 13). Określ następujące ustawienia:
 - Tylko **nazwa** i **EntityType** atrybuty **EntitySet** są określony element.
 - W pełni kwalifikowana nazwa typu jednostki jest używany w **EntityType** atrybutu.
 - Instrukcja SQL do wykonania jest określona w **DefiningQuery** elementu.

```

<!-- SSDL content -->
<edmx:StorageModels>
    <Schema Namespace="SchoolModel.Store" Alias="Self" Provider="System.Data.SqlClient"
ProviderManifestToken="2008"
xmlns:store="http://schemas.microsoft.com/ado/2007/12/edm/EntityStoreSchemaGenerator"
xmlns="http://schemas.microsoft.com/ado/2009/11/edm/ssdl">
        <EntityContainer Name="SchoolModelStoreContainer">
            <EntityType Name="GradeReport" EntityType="SchoolModel.Store.GradeReport">
                <DefiningQuery>
                    SELECT CourseID, Grade, FirstName, LastName
                    FROM StudentGrade
                    JOIN
                    (SELECT * FROM Person WHERE EnrollmentDate IS NOT NULL) AS p
                    ON StudentID = p.PersonID
                </DefiningQuery>
            </EntityType>
            <EntityType Name="Course" EntityType="SchoolModel.Store.Course" store:Type="Tables" Schema="dbo" />
        </EntityContainer>
    </Schema>
</edmx:StorageModels>

```

- Dodaj **EntityType** element do sekcji SSDL edmx. Plik jak pokazano poniżej. Należy pamiętać o następujących kwestiach:
 - Wartość **nazwa** atrybut odnosi się do wartości **EntityType** atrybutu w **EntityType** element powyżej, mimo że w pełni kwalifikowana nazwa Typ jednostki jest używany w **EntityType** atrybutu.
 - Nazwy właściwości odpowiadają nazwom kolumny zwróconą przez instrukcję SQL w **DefiningQuery** elementu (powyżej).
 - W tym przykładzie klucz jednostki składa się z trzech właściwości w celu zapewnienia unikalną wartość kluczową.

```

<EntityType Name="GradeReport">
    <Key>
        <PropertyRef Name="CourseID" />
        <PropertyRef Name="FirstName" />
        <PropertyRef Name="LastName" />
    </Key>
    <Property Name="CourseID"
        Type="int"
        Nullable="false" />
    <Property Name="Grade"
        Type="decimal"
        Precision="3"
        Scale="2" />
    <Property Name="FirstName"
        Type="nvarchar"
        Nullable="false"
        MaxLength="50" />
    <Property Name="LastName"
        Type="nvarchar"
        Nullable="false"
        MaxLength="50" />
</EntityType>

```

NOTE

Jeśli później uruchomić **Kreator modelu aktualizacji** okno dialogowe, wszelkie zmiany wprowadzone do modelu magazynu, w tym Definiowanie zapytań, zostaną zastąpione.

Dodaj typ jednostki do modelu

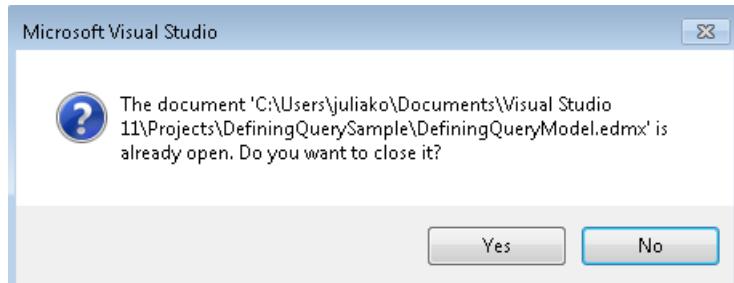
W tym kroku zostanie dodany typ jednostki do modelu koncepcyjnego, za pomocą projektanta EF. Należy

pamiętać o następujących kwestiach:

- **Nazwa** jednostki odnosi się do wartości **EntityType** atrybutu w **EntitySet** powyżej elementu.
- Nazwy właściwości odpowiadają nazwom kolumny zwróconą przez instrukcję SQL w **DefiningQuery** powyżej elementu.
- W tym przykładzie klucz jednostki składa się z trzech właściwości w celu zapewnienia unikalną wartość kluczową.

W Projektancie platformy EF, należy otworzyć modelu.

- Kliknij dwukrotnie DefiningQueryModel.edmx.
- Powiedz **tak** się następujący komunikat:



Zostanie wyświetlona Projektancie jednostki, zapewniającą powierzchnię projektową do edycji modelu.

- Kliknij prawym przyciskiem myszy Projektanta powierzchni i wybierz **Dodaj nowe->jednostki...**.
- Określ **GradeReport** dla nazwy jednostki i **CourseID** dla **właściwości klucza**.
- Kliknij prawym przyciskiem myszy **GradeReport** jednostki, a następnie wybierz pozycję **Dodaj nowe - > właściwość skalarną**.
- Zmienić domyślną nazwę właściwości do **FirstName**.
- Dodaj inną właściwość skalarną i określ **LastName** dla nazwy.
- Dodaj inną właściwość skalarną i określ **klasy korporacyjnej** dla nazwy.
- W **właściwości** oknie zmiany **klasy korporacyjnej** firmy **typu** właściwości **dziesiętna**.
- Wybierz **FirstName** i **LastName** właściwości.
- W **właściwości** oknie zmiany **EntityKey** wartość właściwości **True**.

W rezultacie, następujące elementy zostały dodane do **CSDL** części pliku edmx.

```
<EntitySet Name="GradeReport" EntityType="SchoolModel.GradeReport" />

<EntityType Name="GradeReport">
  ...
</EntityType>
```

Definiowanie zapytania do typu jednostki mapy

W tym kroku użyjemy, okno Szczegóły mapowania, aby zamapować koncepcyjnej i typy jednostek magazynu.

- Kliknij prawym przyciskiem myszy **GradeReport** jednostki na projekt powierzchni i wybierz **mapowania tabeli**.
 - **Szczegóły mapowania** zostanie wyświetlane okno.
 - Wybierz **GradeReport** z **<Dodaj tabelę lub widok>** listy rozwijanej (znajdującej się w **tabelis**).
- Domyślne mapowania między koncepcyjnej i magazynu **GradeReport** typu jednostki są wyświetlane.

Parameter / Column	Operator	Property	Use Original...	Rows Affected Parameter
Functions				
Insert Using InsertPerson				
Parameters				
@ LastName : nvarchar	←	LastName : String	<input type="checkbox"/>	
@ FirstName : nvarchar	←	FirstName : String	<input type="checkbox"/>	
@ HireDate : datetime	←	HireDate : DateTime	<input type="checkbox"/>	
@ EnrollmentDate : datetime	←	EnrollmentDate : DateTime	<input type="checkbox"/>	
@ Discriminator : nvarchar	←	Discriminator : String	<input type="checkbox"/>	
Result Column Bindings				
NewPersonID	→	PersonID : Int32	<input checked="" type="checkbox"/>	

W rezultacie **obiekcie EntitySetMapping** element zostanie dodany do sekcji mapowania pliku edmx.

```
<EntitySetMapping Name="GradeReports">
  <EntityTypeMapping TypeName="IsTypeOf(SchoolModel.GradeReport)">
    <MappingFragment StoreEntitySet="GradeReport">
      <ScalarProperty Name="LastName" ColumnName="LastName" />
      <ScalarProperty Name="FirstName" ColumnName="FirstName" />
      <ScalarProperty Name="Grade" ColumnName="Grade" />
      <ScalarProperty Name="CourseID" ColumnName="CourseID" />
    </MappingFragment>
  </EntityTypeMapping>
</EntitySetMapping>
```

- Kompilowanie aplikacji.

Wywołaj Definiowanie zapytania w kodzie

Definiowanie zapytania można teraz wykonać przy użyciu **GradeReport** typu jednostki.

```
using (var context = new SchoolEntities())
{
  var report = context.GradeReports.FirstOrDefault();
  Console.WriteLine("{0} {1} got {2}",
    report.FirstName, report.LastName, report.Grade);
}
```

Procedury składowane z wielu zestawów wyników

13.09.2018 • 9 minutes to read • [Edit Online](#)

Czasami w przypadku, gdy przy użyciu przechowywanych procedur należy zwrócić więcej niż jeden wynik jest ustawiona. Ten scenariusz jest najczęściej używany do zmniejszenia liczby bazy danych rund wymagane do redagowania na jednym ekranie. Przed EF5 platformy Entity Framework pozwoliłoby procedura składowana wywoływana, ale zwróci tylko pierwszy zestaw wyników do kodu wywołującego.

W tym artykule przedstawiono dwie metody, których można uzyskać dostęp do więcej niż jeden zestaw wyników z procedury składowanej platformy Entity Framework. Taki, który używa tylko kodu i współdziała z obu kod najpierw i projektancie platformy EF i taki, który działa tylko w Projektancie platformy EF. Narzędzia i obsługa interfejsu API dla tej powinna zwiększyć w przyszłych wersjach programu Entity Framework.

Model

Przykłady w niniejszym artykule użyć podstawowa blogu i modelu wpisów, gdzie blogu ma wiele wpisów i wpis należy do jednego blogu. Firma Microsoft używa procedurę składowaną w bazie danych, które zwraca wszystkie blogów i wpisów, podobnie do następującej:

```
CREATE PROCEDURE [dbo].[GetAllBlogsAndPosts]
AS
    SELECT * FROM dbo.Blogs
    SELECT * FROM dbo.Posts
```

Uzyskiwanie dostępu do wielu wyników ustawia przy użyciu kodu

Firma Microsoft może wykonać kod użycia do wystawiania pierwotne polecenia SQL do wykonywania naszego procedury składowanej. Zaletą tego podejścia jest to, że działa zarówno kod najpierw i projektancie platformy EF.

Aby uzyskać wynik wielu ustawia pracy potrzebnych do spadku API obiektu ObjectContext za pomocą interfejsu IObjectContextAdapter.

Gdy będziemy już mieć obiektu ObjectContext, a następnie możemy użyć metody Translate do translacji wyników naszego procedury składowanej do jednostek, które mogą być śledzone i używane w programie EF, jak zwykle. Poniższy przykładowy kod przedstawia to w działaniu.

```

using (var db = new BloggingContext())
{
    // If using Code First we need to make sure the model is built before we open the connection
    // This isn't required for models created with the EF Designer
    db.Database.Initialize(force: false);

    // Create a SQL command to execute the sproc
    var cmd = db.Database.Connection.CreateCommand();
    cmd.CommandText = "[dbo].[GetAllBlogsAndPosts]";

    try
    {

        db.Database.Connection.Open();
        // Run the sproc
        var reader = cmd.ExecuteReader();

        // Read Blogs from the first result set
        var blogs = ((IObjectContextAdapter)db)
            .ObjectContext
            .Translate<Blog>(reader, "Blogs", MergeOption.AppendOnly);

        foreach (var item in blogs)
        {
            Console.WriteLine(item.Name);
        }

        // Move to second result set and read Posts
        reader.NextResult();
        var posts = ((IObjectContextAdapter)db)
            .ObjectContext
            .Translate<Post>(reader, "Posts", MergeOption.AppendOnly);

        foreach (var item in posts)
        {
            Console.WriteLine(item.Title);
        }
    }
    finally
    {
        db.Database.Connection.Close();
    }
}

```

Metoda Translate przyjmuje czytnika, które odebraliśmy, gdy firma Microsoft wykonywane procedury, nazwa obiektu EntitySet i MergeOption. Nazwa obiektu EntitySet będzie taka sama jak właściwość DbSet pochodzącej kontekstu. Wyliczenie MergeOption kontroluje sposób obsługi wyniki, jeśli istnieje już tej samej jednostki w pamięci.

W tym miejscu możemy iterowania po kolekcji blogów przed nazywanym NextResult, jest to ważne, dany kod powyżej, ponieważ pierwszy zestaw wyników, muszą być przetworzone przed przejęciem do następnego zestawu wyników.

Po dwóch przetłumaczyć metody są wywoływane, a następnie jednostek blogu i Post są śledzone przez EF w taki sam sposób jak inne jednostki, a zatem być zmodyfikowany lub usunięty i zapisywane jako normalny.

NOTE

EF nie przyjmuje żadnego mapowania pod uwagę podczas tworzenia jednostki przy użyciu metody translacji. Będą one po prostu zgodne nazwy kolumn w zestawie wyników z nazwami właściwości w Twoich zajęciach.

NOTE

Czy w przypadku ładowania z opóźnieniem, włączone, uzyskiwania dostępu do właściwości wpisy na jednej z jednostek blogu następnie EF połączy się bazy danych do załadowania opóźnieniem wszystkie wpisy, mimo że firma Microsoft zpostała już załadowane je wszystkie. Jest to spowodowane EF nie wiedzieć, czy zostały załadowane wszystkie wpisy lub jeśli istnieje więcej w bazie danych. Jeśli chcesz tego uniknąć, a następnie konieczne będzie wyłączenie ładowania z opóźnieniem.

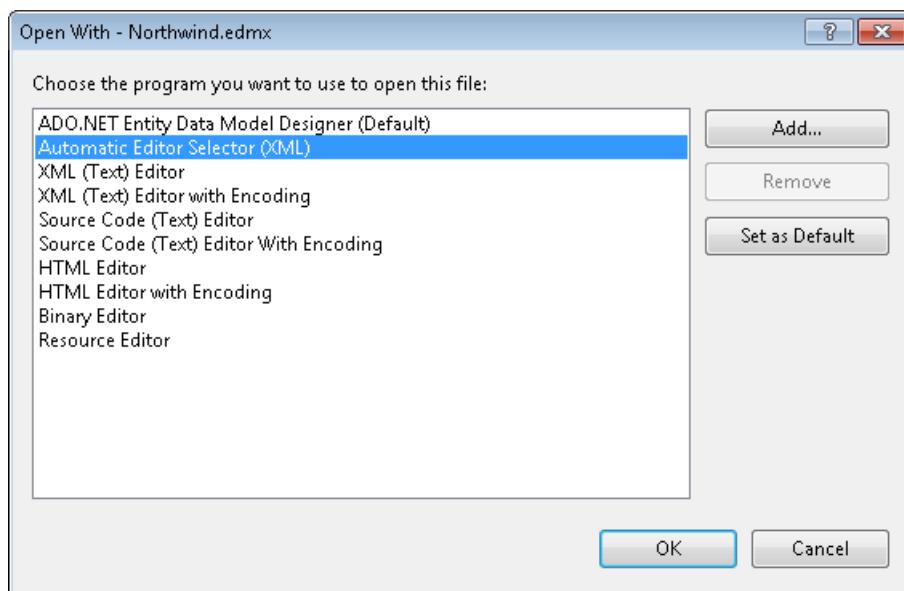
Wiele zestawów wyników za pomocą skonfigurowanej w EDMX

NOTE

Należy wskazać .NET Framework 4.5, aby mieć możliwość skonfigurowania wielu zestawów wyników w EDMX. Jeśli masz na celu platformy .NET 4.0, można użyć metody oparte na kodzie pokazano w poprzedniej sekcji.

Jeśli używasz projektancie platformy EF, można również zmodyfikować model, aby poinformować go o zestawach różne wyniki, które zostaną zwrócone. Warto zapoznać się przed ręcznie jest, że narzędzi nie jest wynikiem wielu ustawić wiedzieć, więc musisz ręcznie edytować plik edmx. Edytowanie pliku edmx, tak jak to działa, ale spowoduje również przerwanie sprawdzania poprawności modelu w programie VS. Dlatego jeśli Weryfikacja modelu będzie zawsze występują błędy.

- W tym celu należy dodać procedurę składowaną do modelu, podobnie jak w przypadku pojedynczego wyniku kwerendy zestawu.
- Po tym, będzie konieczne modelu kliknij prawym przyciskiem myszy i wybierz **Otwórz za pomocą...** następnie **Xml**



Masz jeden raz modelu otwarty jako XML, a następnie należy wykonać następujące czynności:

- Znajdź złożonych importu typ i funkcję w modelu:

```

<!-- CSDL content -->
<edmx:ConceptualModels>

...

<FunctionImport Name="GetAllBlogsAndPosts" ReturnType="Collection(BlogModel.GetAllBlogsAndPosts_Result)"
/>

...

<ComplexType Name="GetAllBlogsAndPosts_Result">
<Property Type="Int32" Name="BlogId" Nullable="false" />
<Property Type="String" Name="Name" Nullable="false" MaxLength="255" />
<Property Type="String" Name="Description" Nullable="true" />
</ComplexType>

...

</edmx:ConceptualModels>

```

- Usuń typ złożony
- Importowanie funkcji należy zaktualizować tak, że jest on mapowany do jednostek, w tym przypadku, który będzie wyglądać następująco:

```

<FunctionImport Name="GetAllBlogsAndPosts">
<ReturnType EntitySet="Blogs" Type="Collection(BlogModel.Blog)" />
<ReturnType EntitySet="Posts" Type="Collection(BlogModel.Post)" />
</FunctionImport>

```

Informuje modelu, czy procedura składowana zwróci dwie kolekcje, jeden z wpisów w blogu i jeden z wpisów post.

- Znajdź element mapowania funkcji:

```

<!-- C-S mapping content -->
<edmx:Mappings>

...

<FunctionImportMapping FunctionImportName="GetAllBlogsAndPosts"
FunctionName="BlogModel.Store.GetAllBlogsAndPosts">
<ResultMapping>
<ComplexTypeMapping TypeName="BlogModel.GetAllBlogsAndPosts_Result">
<ScalarProperty Name="BlogId" ColumnName="BlogId" />
<ScalarProperty Name="Name" ColumnName="Name" />
<ScalarProperty Name="Description" ColumnName="Description" />
</ComplexTypeMapping>
</ResultMapping>
</FunctionImportMapping>

...

</edmx:Mappings>

```

- Zastąp mapowania wynik jednym dla każdej jednostki, które są zwracane, takie jak następujące:

```

<ResultMapping>
  <EntityTypeMapping TypeName = "BlogModel.Blog">
    <ScalarProperty Name="BlogId" ColumnName="BlogId" />
    <ScalarProperty Name="Name" ColumnName="Name" />
    <ScalarProperty Name="Description" ColumnName="Description" />
  </EntityTypeMapping>
</ResultMapping>
<ResultMapping>
  <EntityTypeMapping TypeName="BlogModel.Post">
    <ScalarProperty Name="BlogId" ColumnName="BlogId" />
    <ScalarProperty Name="PostId" ColumnName="PostId"/>
    <ScalarProperty Name="Title" ColumnName="Title" />
    <ScalarProperty Name="Text" ColumnName="Text" />
  </EntityTypeMapping>
</ResultMapping>

```

Istnieje również możliwość mapowania typów złożonych, takiego jak utworzone domyślnie zestawów wyników. W tym celu możesz utworzyć nowy typ złożony, zamiast je, usuwania i używać złożone typy wszędzie, gdyby użyto nazwy jednostek w powyższych przykładach.

Po mapowania te zostały zmienione, można zapisać model i wykonaj następujący kod, aby użyć procedury składowanej:

```

using (var db = new BlogEntities())
{
  var results = db.GetAllBlogsAndPosts();

  foreach (var result in results)
  {
    Console.WriteLine("Blog: " + result.Name);
  }

  var posts = results.GetNextResult<Post>();

  foreach (var result in posts)
  {
    Console.WriteLine("Post: " + result.Title);
  }

  Console.ReadLine();
}

```

NOTE

Ręczna Edycja pliku edmx dla modelu zostaną zastąpione, jeśli kiedykolwiek ponowne wygenerowanie modelu z bazy danych.

Podsumowanie

Zostały tutaj pokazano dwa różne sposoby uzyskiwania dostępu do wielu wyników ustawia używający narzędzia Entity Framework. Obie z nich są równoważne w zależności od potrzeb i preferencji i wybrać ten, który wydaje się najlepiej w przypadku Twojej sytuacji. Planuje obsługę wielu wyników, których zestawy będą ulepszone w przyszłych wersjach programu Entity Framework i, wykonując kroki opisane w tym dokumencie nie będzie już konieczne.

Funkcje z wartościami przechowywanymi w tabeli (funkcji Tvf)

27.09.2018 • 6 minutes to read • [Edit Online](#)

NOTE

EF5 poczawszy tylko — funkcje, interfejsów API itp. z opisem na tej stronie zostały wprowadzone w programie Entity Framework 5. Jeśli używasz starszej wersji, niektóre lub wszystkie informacje, nie ma zastosowania.

Przewodnik krok po kroku i wideo pokazuje, jak mapować zwracających tabelę (funkcji Tvf) za pomocą programu Entity Framework Designer. Ilustruje też sposób wywoływania funkcji TVF w wyniku zapytania LINQ.

Funkcji Tvf są obecnie obsługiwane tylko w bazie danych pierwszego przepływu pracy.

Obsługa funkcji TVF została wprowadzona w programie Entity Framework w wersji 5. Należy pamiętać, że aby korzystać z nowych funkcji, takich jak zwracających tabelę, wyliczeń i typów przestrzennych, należy wskazać .NET Framework 4.5. Program Visual Studio 2012 jest przeznaczony dla platformy .NET 4.5 domyślnie.

Funkcji Tvf są bardzo podobne do procedur składowanych z jedną kluczową różnicą: wynik funkcji TVF jest konfigurowalna. Oznacza to, że wyniki z funkcji TVF mogą być używane w zapytaniu LINQ, ale nie wyników procedury składowanej.

Obejrzyj wideo

Osoba prezentująca: Julia Kornich

[WMV](#) | [MP4](#) | [WMV \(ZIP\)](#)

Wymagania wstępne

Aby ukończyć ten Instruktaż, musisz:

- Zainstaluj [bazę danych School](#).
- Posiadania najnowszej wersji programu Visual Studio

Konfigurowanie projektu

1. Otwórz program Visual Studio
2. Na **pliku** menu wskaż **New**, a następnie kliknij przycisk **projektu**
3. W okienku po lewej stronie kliknij **Visual C#**, a następnie wybierz pozycję **konsoli** szablonu
4. Wprowadź **TVF** jako nazwę projektu i kliknij przycisk **OK**

Dodawanie funkcji TVF w bazie danych

- Wybierz **widok** —> **Eksplorator obiektów SQL Server**
- Jeśli LocalDB nie znajduje się lista serwerów: kliknij prawym przyciskiem myszy **programu SQL Server** i wybierz **dodawania serwera SQL** Użyj domyślnej **uwierzytelniania Windows** do łączenia się z serwerem LocalDB
- Rozwiń węzeł LocalDB

- W węźle bazy danych, kliknij prawym przyciskiem myszy węzeł bazy danych School, a następnie wybierz pozycję **nowe zapytanie...**
- W edytorze języka T-SQL, Wklej poniższą definicję funkcji TVF

```
CREATE FUNCTION [dbo].[GetStudentGradesForCourse]
(@CourseID INT)
RETURNS TABLE
RETURN
SELECT [EnrollmentID],
       [CourseID],
       [StudentID],
       [Grade]
FROM   [dbo].[StudentGrade]
WHERE  CourseID = @CourseID
```

- Kliknij prawym przyciskiem myszy w edytorze języka T-SQL i wybierz pozycję **wykonania**
- Funkcja GetStudentGradesForCourse zostanie dodany do bazy danych School

Tworzenie modelu

1. Kliknij prawym przyciskiem myszy nazwę projektu w Eksploratorze rozwiązań, wskaż opcję **Dodaj**, a następnie kliknij przycisk **nowy element**
2. Wybierz **danych** z menu po lewej stronie, a następnie wybierz pozycję **ADO.NET Entity Data Model** w **szablony** okienko
3. Wprowadź **TVFModel.edmx** nazwę pliku, a następnie kliknij przycisk **Dodaj**
4. W oknie dialogowym Wybierz zawartość modelu, wybierz **Generuj z bazy danych**, a następnie kliknij przycisk **dalej**
5. Kliknij przycisk **nowe połączenie** Enter (**localdb**)\mssql\localdb w tekście nazwy serwera polu wprowadź **School** bazy danych programu nazwę kliknij **OK**
6. W polu Wybierz obiekty bazy danych okna dialogowego w polu **tabel** węzeł **osoby**, **StudentGrade**, i **kurs** tabel
7. Wybierz **GetStudentGradesForCourse** funkcja znajdujący się w folderze **procedur przechowywanych i funkcji** węzła należy pamiętać, że począwszy od programu Visual Studio 2012, Projektant obiektów umożliwia importowanie usługi batch Twoje procedur przechowywanych i funkcji
8. Kliknij przycisk **Zakończ**
9. Zostanie wyświetlona Projektancie jednostki, zapewniającą powierzchnię projektową do edycji modelu. Wszystkie obiekty, które wybrano w **wybierz obiekty bazy danych** okno dialogowe, są dodawane do modelu.
10. Domyślnie kształtu wynik każdego importowanych procedura składowana lub funkcja automatycznie stanie się nowy typ złożony w modelu entity. Ale chcemy mapowania wyniki funkcji GetStudentGradesForCourse jednostki StudentGrade: kliknij prawym przyciskiem myszy projekt powierzchni i wybierz **przeglądarka modeli** w przeglądarce modelu wybierz **Import funkcja**, a następnie kliknij dwukrotnie **GetStudentGradesForCourse** funkcji w edytować funkcji Importuj okno dialogowe, zaznacz **jednostek** i wybierz polecenie **StudentGrade**

Utrwalanie i pobieranie danych

Otwórz plik, w których zdefiniowano metody Main. Dodaj następujący kod do funkcji Main.

Poniższy kod przedstawia sposób tworzenia zapytania, które korzysta z funkcji z wartościami przechowywanymi w tabeli. Zapytanie projektów wyniki na typ anonimowy, który zawiera powiązane tytuł kursu i powiązane uczniom i studentom klasy korporacyjnej, większa lub równa 3.5.

```
using (var context = new SchoolEntities())
{
    var CourseID = 4022;
    var Grade = 3.5M;

    // Return all the best students in the Microeconomics class.
    var students = from s in context.GetStudentGradesForCourse(CourseID)
                  where s.Grade >= Grade
                  select new
                  {
                      s.Person,
                      s.Course.Title
                  };

    foreach (var result in students)
    {
        Console.WriteLine(
            "Couse: {0}, Student: {1} {2}",
            result.Title,
            result.Person.FirstName,
            result.Person.LastName);
    }
}
```

Skompilować i uruchomić aplikację. Program generuje następujące wyniki:

```
Couse: Microeconomics, Student: Arturo Anand
Couse: Microeconomics, Student: Carson Bryant
```

Podsumowanie

W tym przewodniku zobaczyliśmy, jak mapować zwracających tabelę (funkcji Tvf) za pomocą programu Entity Framework Designer. On również przedstawiono sposób wywoływania funkcji TVF w wyniku zapytania LINQ.

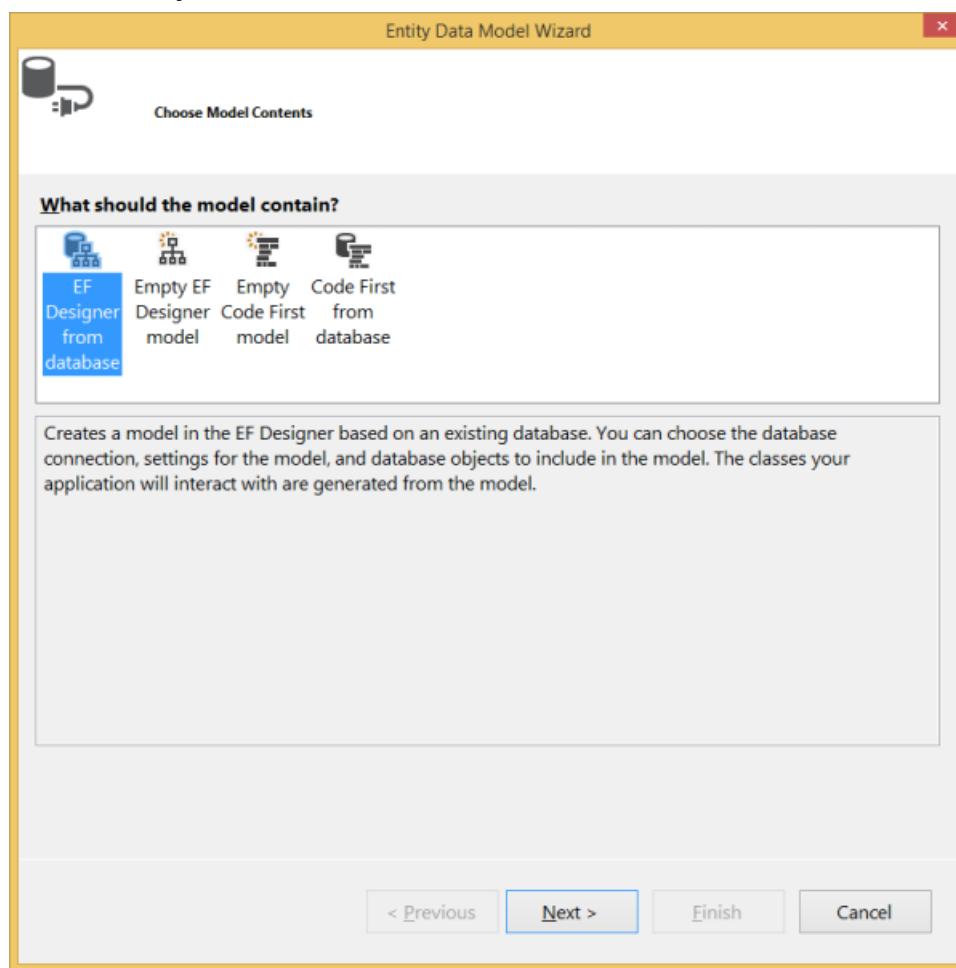
Skróty klawiaturowe projektanta programu Entity Framework

13.09.2018 • 13 minutes to read • [Edit Online](#)

Ta strona zawiera listę skróty klawiaturowe, które są dostępne na różnych ekranach narzędzi Entity Framework Tools for Visual Studio.

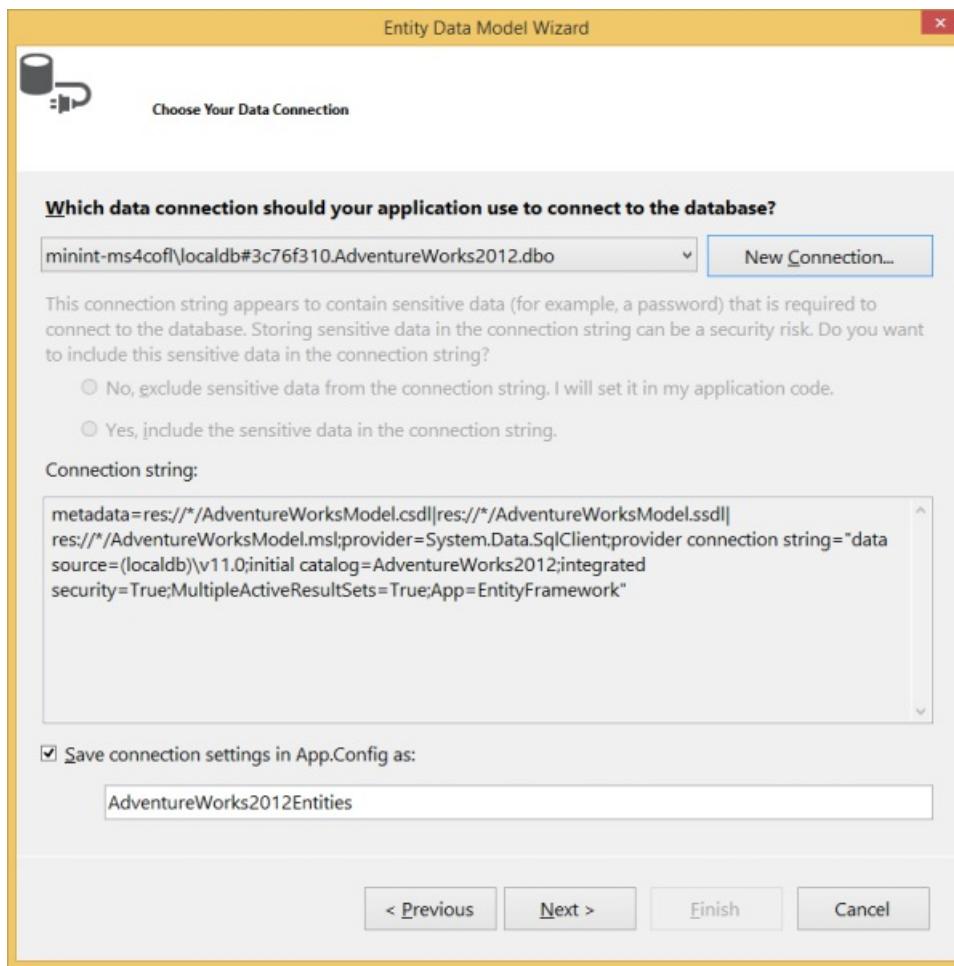
Kreator modelu danych jednostki ADO.NET

Jeden krok: Wybierz zawartość modelu



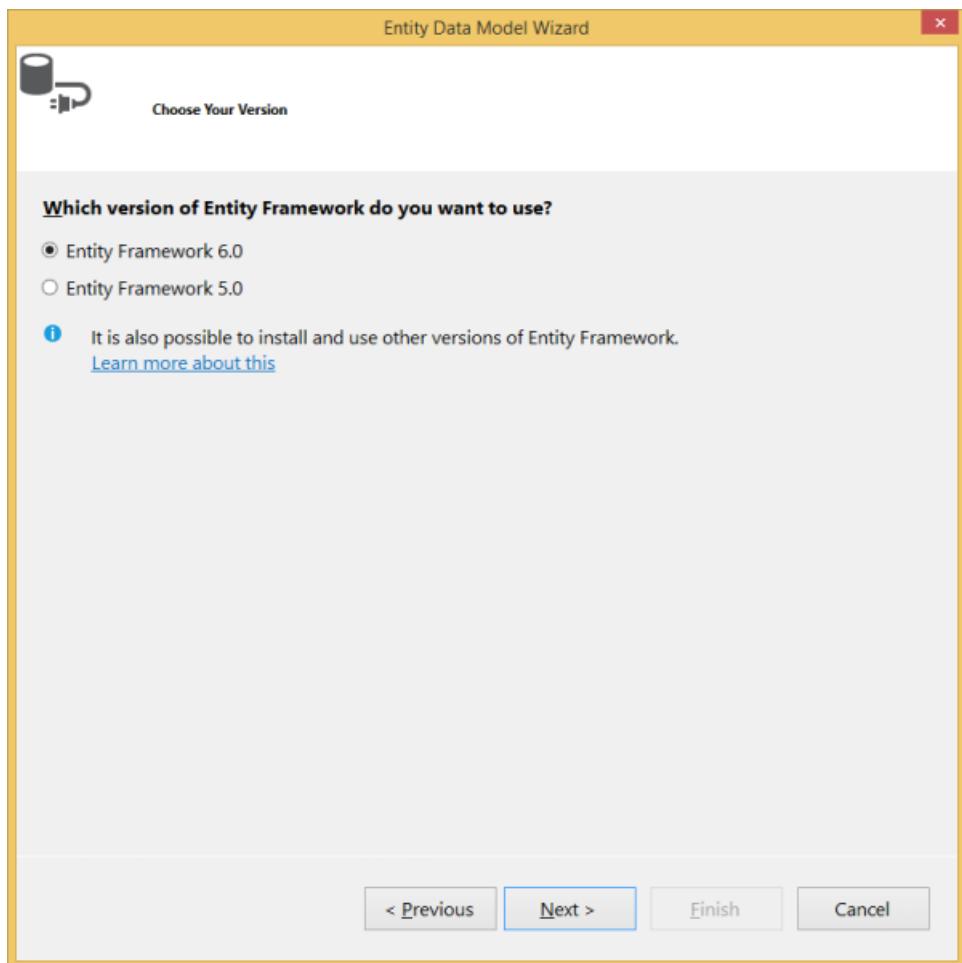
SKRÓT	AKCJA	UWAGI
Alt + n	Przenieś do następnego ekranu	Nie jest dostępna dla wszystkich wyborów zawartość modelu.
ALT + f	Zakończ pracę kreatora	Nie jest dostępna dla wszystkich wyborów zawartość modelu.
ALT + w	Przełączka widok "co modelu może zawierać?" okienko.	

Krok 2: Wybierz połączenie



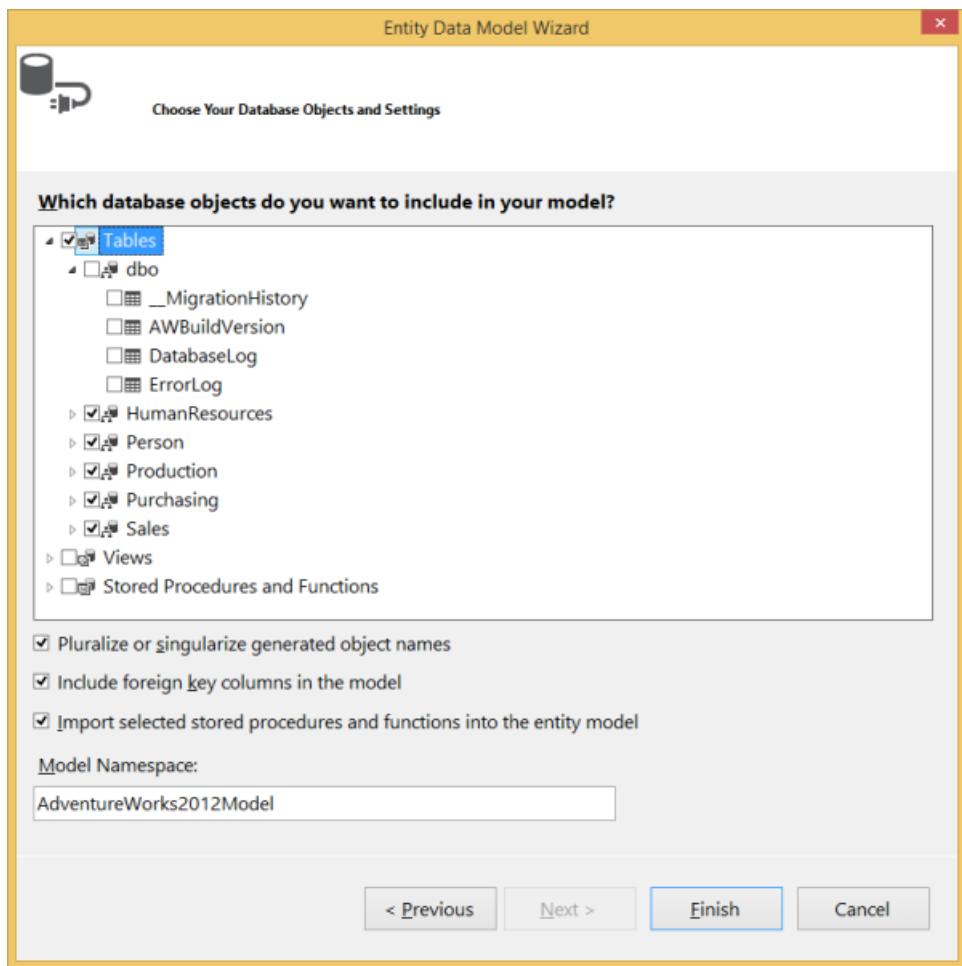
SKRÓT	AKCJA	UWAGI
Alt + n	Przenieś do następnego ekranu	
ALT + p	Przenieś do poprzedniego ekranu	
ALT + w	Przełączka widok "co modelu może zawierać?" okienko.	
Alt + c	Otwórz okno "Właściwości połączenia"	Umożliwia określenie nowego połączenia z bazą danych.
Alt + e	Wyklucz poufne dane z parametrów połączenia	
ALT + i	Dołączenia danych poufnych w parametrach połączenia	
Alt + s	Ustaw opcję "Zapisz ustawienia połączenia w pliku App.Config"	

Krok 3: Wybierz swoją wersję



SKRÓT	AKCJA	UWAGI
Alt + n	Przenieś do następnego ekranu	
ALT + p	Przenieś do poprzedniego ekranu	
ALT + w	Przełącz fokus na wybór wersji platformy Entity Framework	Umożliwia określenie innej wersji programu Entity Framework do użycia w projekcie.

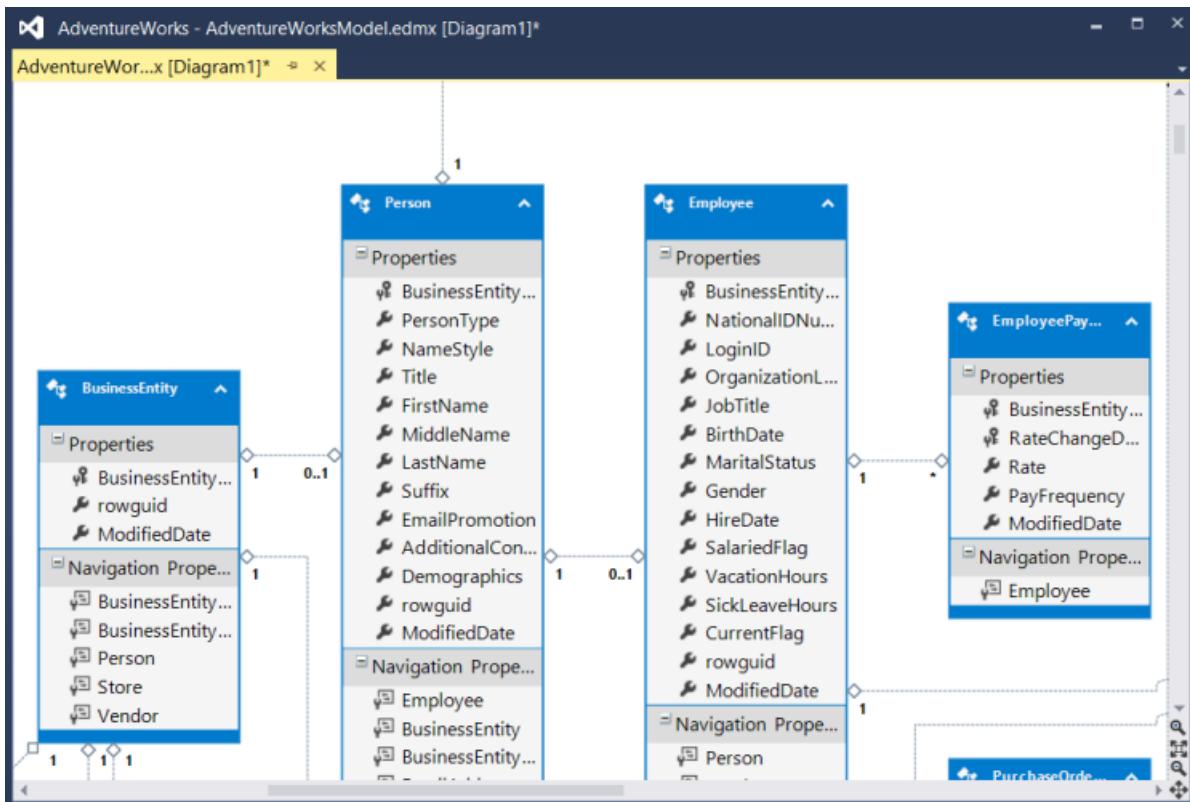
Krok 4: Wybierz obiekty bazy danych i ustawień



SKRÓT	AKCJA	UWAGI
ALT + f	Zakończ pracę kreatora	
ALT + p	Przenieś do poprzedniego ekranu	
ALT + w	Przełącz widok okienko wyboru obiektu bazy danych	Umożliwia określenie obiektów bazy danych do odtwarzania odtwarzane.
Alt + s	Przełącz "Pluralize lub końcówek nazw generowanych obiektów" opcja	
ALT + k	Ustaw opcję "Dołącz kolumny klucza obcego w modelu"	Nie jest dostępna dla wszystkich wyborów zawartość modelu.
ALT + i	Ustaw opcję "Importuj wybrane procedur przechowywanych i funkcji do modelu entity"	Nie jest dostępna dla wszystkich wyborów zawartość modelu.
ALT + m	Przełącz fokus do pola tekstowego "Model Namespace"	Nie jest dostępna dla wszystkich wyborów zawartość modelu.
miejsce	Przełącz zaznaczenie elementu	Jeśli element ma elementy podrzędne, wszystkie elementy podrzędne zostaną także przełączane
po lewej stronie	Zwiń drzewa podrzędne	

SKRÓT	AKCJA	UWAGI
po prawej stronie	Rozwiń drzewo podrzędne	
W górę	Przejdź do poprzedniego elementu w drzewie	
W dół	Przejdź do następnego elementu w drzewie	

Powierzchnia projektowa EF



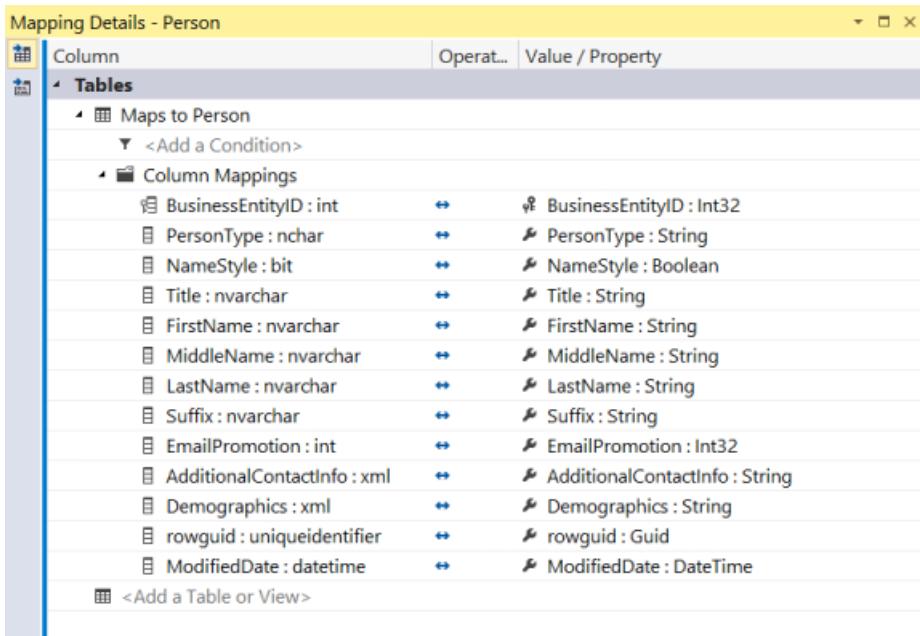
SKRÓT	AKCJA	UWAGI
Wprowadź/miejsca	Przełącz zaznaczenie	Włącza lub wyłącza wybór obiektu z fokusem.
ESC	Anuluj zaznaczenie	Anuluje bieżące zaznaczenie.
CTRL + A	Zaznacz wszystko	Wybiera wszystkie kształty na powierzchni projektowej.
Strzałka w góre	Przenieś w górę	Przenosi wybrany jednostki się przyrostu jednej siatce. Jeśli na liście, przenosi do poprzedniego ograniczonym element równorzędny.
Strzałka w dół	Przenieś w dół	Przenosi wybrany jednostki przyrostu jednej siatce w dół. Jeśli na liście, przechodzi do następnego ograniczonym element równorzędny.

SKRÓT	AKCJA	UWAGI
Strzałka w lewo	Przenieś w lewo	Przenosi wybrany jednostki przyrostu po lewej stronie w jednej siatce. Jeśli na liście, przenosi do poprzedniego ograniczonym element równorzędny.
Strzałka w prawo	Przesuń w prawo	Przenosi wybrany przyrostu siatki jeden jednostki. Jeśli na liście, przechodzi do następnego ograniczonym element równorzędny.
Shift + Strzałka w lewo	Rozmiar kształtu w lewo	Zmniejsza szerokość wybraną jednostkę, z przyrostem jednej siatce.
Shift + Strzałka w prawo	Rozmiar kształtu w prawo	Zwiększa szerokość wybranej jednostki przyrostu jednej siatce.
Strona główna	Pierwszy elementu równorzędnego	Przenosi fokus i wybór, aby pierwszy obiekt na powierzchni projektowej, w tym samym poziomie elementów równorzędnych.
End	Ostatniego elementu równorzędnego	Przenosi fokus i wybór, ostatni obiekt na powierzchni projektowej, w tym samym poziomie elementów równorzędnych.
Ctrl + Home	Pierwszy elementu równorzędnego (focus)	Tak samo jak pierwszy elementu równorzędnego, ale przenosi fokus zamiast przenoszenia fokusu i wybór.
CTRL + End	Ostatniego elementu równorzędnego (focus)	Takie same jak ostatniej komunikacji równorzędnej, ale przenosi fokus zamiast przenoszenia fokusu i wybór.
Karta	Dalej elementu równorzędnego	Przenosi fokus i zaznaczenie do następnego obiektu na powierzchni projektowej, w tym samym poziomie elementów równorzędnych.
Shift + Tab	Poprzednich elementów równorzędnych	Przenosi fokus i zaznaczenie do poprzedniego obiektu na powierzchni projektowej, w tym samym poziomie elementów równorzędnych.
ALT + klawisze Ctrl + Tab	Dalej elementu równorzędnego (focus)	Takie same jak dalej komunikacji równorzędnej, ale przenosi fokus zamiast przenoszenia fokusu i wybór.
ALT + klawisze Ctrl + Shift + Tab	Poprzednich elementów równorzędnych (focus)	Tak samo jak w poprzednich elementów równorzędnych, ale przenosi fokus zamiast przenoszenia fokusu i wybór.

SKRÓT	AKCJA	UWAGI
<	Ascend	Przechodzi do następnego obiektu na projekt powierzchni jeden poziom wyżej w hierarchii. Jeśli nie istnieją żadne kształty powyżej tego kształtu w hierarchii (oznacza to, że obiekt znajduje się bezpośrednio na powierzchni projektowej), diagram jest zaznaczone.
>	Jest elementem podrzędnym elementu	Przechodzi do następnego obiektu zawarte na projekt powierzchni jeden poziom poniżej tego w hierarchii. W przypadku przechowywany obiekt, który nie jest pusta.
CTRL + <	Ascend (focus)	Takie same jak ascend polecenia, ale przenosi fokus bez zaznaczenia.
CTRL + >	Jest elementem podrzędnym elementu (focus)	Takie same jak jest elementem podrzędnym elementu polecenia, ale przenosi fokus bez zaznaczenia.
Shift + End	Należy wykonać, aby połączone	Z jednostki przenosi do jednostki podłączone do tej jednostki.
Del	Usuwanie	Usuń obiekt lub łącznika z diagramu.
Dodatki	Insert	Dodaje nową właściwość do jednostki, po wybraniu nagłówka przedziału "Scalar właściwości" lub samej właściwości.
Strona w góre	Diagram przewiń w górę	Przewija powierzchni projektowej, w przyrostach co równa 75% wysokość powierzchni projektowej aktualnie widoczne.
Strona w dół	Diagram przewiń w dół	Przewija widok powierzchni projektowej w dół.
SHIFT + strona w dół	Diagram przewiń w prawo	Przewija powierzchni projektowej po prawej stronie.
SHIFT + strona w góre	Diagram przewiń w lewo	Przewija powierzchni projektowej po lewej stronie.
F2	Przejdź do trybu edycji	Standardowa skrót klawiaturowy do wprowadzenia trybu edycji dla kontrolki tekstu.
SHIFT + F10	Wyświetla menu skrótów.	Standardowa skrót klawiaturowy do wyświetlania wybranego elementu menu skrótów.

SKRÓT	AKCJA	UWAGI
Kliknięcie lewym przyciskiem Control + Shift + myszy Kontrolka + Shift + kółka myszy do przodu	Powiększenie semantyczne w	Powiększa się w obszarze widoku diagramu pod kursem myszy.
Kliknij prawym przyciskiem myszy formant + Shift + myszy Kontrolka + Shift + kółka myszy do tyłu	Powiększenie semantyczne Out	Powiększa w obszarze widoku diagramu pod kursem myszy. Centra ponownie diagramu, podczas powiększania zbyt daleko dla bieżącego diagramu produkcyjnego.
Control + Shift + '+' CTRL + kółka myszy do przodu	Powiększ	Powiększa się w środku widoku diagramu.
Control + Shift + "-" CTRL + kółka myszy do tyłu	Pomniejsz	Powiększa w obszarze kliknięto widoku diagramu. Centra ponownie diagramu, podczas powiększania zbyt daleko dla bieżącego diagramu produkcyjnego.
Control + Shift + narysować prostokąt z lewego przycisku myszy w dół	Obszar powiększenia	Powiększa tematyka koncentruje się na obszar, który został wybrany. Po przytrzymaj klawisze Shift + formant, pojawi się, że kursor zmienia się na lupy, dzięki czemu można określić obszar umożliwiającą powiększenie fragmentu.
Klawisz Menu kontekstowe + 'M'	Otwórz okno Szczegóły mapowania	Zostanie otwarte okno Szczegóły mapowania, aby edytować mapowania dla wybranej jednostki

Okno Szczegóły mapowania



SKRÓT	AKCJA	UWAGI
Karta	Przełącz kontekst	Przełącza między obszaru okno główne i narzędzi po lewej stronie

SKRÓT	AKCJA	UWAGI
Klawisze strzałek	Nawigacja	Przemieszczać się w wierszach, lub po prawej stronie i po lewej stronie w kolumnach w obszarze głównego okna. Przenoszenie między przyciskami na pasku narzędzi po lewej stronie.
Wprowadź miejsce	Wybierz	Wybierze przycisk na pasku narzędzi po lewej stronie.
Alt + Strzałka w dół	Otwieranie listy	Listę rozwijaną listę, jeśli komórka jest zaznaczona, który ma listy rozwijanej.
Wprowadź	Wybierz z listy	Wybiera element na liście rozwijanej.
ESC	Zamknij listę	Zamyka listy rozwijanej.

Nawigacja w programie Visual Studio

Entity Framework dostarcza również akcje, które mogą mieć niestandardowych skrótów klawiaturowych mapowane (nie skróty są mapowane domyślnie). Aby utworzyć te skróty niestandardowe, kliknij menu Narzędzia, a następnie opcje. W obszarze środowiska wybierz opcję klawiatury. Przewiń w dół na liście w środku dopiero po przeprowadzeniu wybranie żądanego polecenia, wprowadź skrót, w polu tekstowym "Naciśnij klawisze skrótu" i kliknij pozycję przypisania. Skróty możliwe są następujące:

SKRÓT
OtherContextMenus.MicrosoftDataEntityDesignContext.Add.ComplexProperty.ComplexTypes
OtherContextMenus.MicrosoftDataEntityDesignContext.AddCodeGenerationItem
OtherContextMenus.MicrosoftDataEntityDesignContext.AddFunctionImport
OtherContextMenus.MicrosoftDataEntityDesignContext.AddNew.AddEnumType
OtherContextMenus.MicrosoftDataEntityDesignContext.AddNew.Association
OtherContextMenus.MicrosoftDataEntityDesignContext.AddNew.ComplexProperty
OtherContextMenus.MicrosoftDataEntityDesignContext.AddNew.ComplexType
OtherContextMenus.MicrosoftDataEntityDesignContext.AddNew.Entity
OtherContextMenus.MicrosoftDataEntityDesignContext.AddNew.FunctionImport
OtherContextMenus.MicrosoftDataEntityDesignContext.AddNew.Inheritance
OtherContextMenus.MicrosoftDataEntityDesignContext.AddNew.NavigationProperty
OtherContextMenus.MicrosoftDataEntityDesignContext.AddNew.ScalarProperty
OtherContextMenus.MicrosoftDataEntityDesignContext.AddNew.Diagram

SKRÓT

OtherContextMenus.MicrosoftDataEntityDesignContext.AddtoDiagram

OtherContextMenus.MicrosoftDataEntityDesignContext.Close

OtherContextMenus.MicrosoftDataEntityDesignContext.Collapse

OtherContextMenus.MicrosoftDataEntityDesignContext.ConverttoEnum

OtherContextMenus.MicrosoftDataEntityDesignContext.Diagram.CollapseAll

OtherContextMenus.MicrosoftDataEntityDesignContext.Diagram.ExpandAll

OtherContextMenus.MicrosoftDataEntityDesignContext.Diagram.ExportasImage

OtherContextMenus.MicrosoftDataEntityDesignContext.Diagram.LayoutDiagram

OtherContextMenus.MicrosoftDataEntityDesignContext.Edit

OtherContextMenus.MicrosoftDataEntityDesignContext.EntityKey

OtherContextMenus.MicrosoftDataEntityDesignContext.Expand

OtherContextMenus.MicrosoftDataEntityDesignContext.FunctionImportMapping

OtherContextMenus.MicrosoftDataEntityDesignContext.GenerateDatabasefromModel

OtherContextMenus.MicrosoftDataEntityDesignContext.GoToDefinition

OtherContextMenus.MicrosoftDataEntityDesignContext.Grid>ShowGrid

OtherContextMenus.MicrosoftDataEntityDesignContext.Grid.SnaptoGrid

OtherContextMenus.MicrosoftDataEntityDesignContext.IncludeRelated

OtherContextMenus.MicrosoftDataEntityDesignContext.Mapping Details

OtherContextMenus.MicrosoftDataEntityDesignContext.ModelBrowser

OtherContextMenus.MicrosoftDataEntityDesignContext.MoveDiagramstoSeparateFile

OtherContextMenus.MicrosoftDataEntityDesignContext.MoveProperties.Down

OtherContextMenus.MicrosoftDataEntityDesignContext.MoveProperties.Down5

OtherContextMenus.MicrosoftDataEntityDesignContext.MoveProperties.ToBottom

OtherContextMenus.MicrosoftDataEntityDesignContext.MoveProperties.ToTop

OtherContextMenus.MicrosoftDataEntityDesignContext.MoveProperties.Up

SKRÓT

OtherContextMenus.MicrosoftDataEntityDesignContext.MoveProperties.Up5

OtherContextMenus.MicrosoftDataEntityDesignContext.MovetonewDiagram

OtherContextMenus.MicrosoftDataEntityDesignContext.Open

OtherContextMenus.MicrosoftDataEntityDesignContext.Refactor.MoveToNewComplexType

OtherContextMenus.MicrosoftDataEntityDesignContext.Refactor.Rename

OtherContextMenus.MicrosoftDataEntityDesignContext.RemovefromDiagram

OtherContextMenus.MicrosoftDataEntityDesignContext.Rename

OtherContextMenus.MicrosoftDataEntityDesignContext.ScalarPropertyFormat.DisplayName

OtherContextMenus.MicrosoftDataEntityDesignContext.ScalarPropertyFormat.DisplayNameandType

OtherContextMenus.MicrosoftDataEntityDesignContext.Select.BaseType

OtherContextMenus.MicrosoftDataEntityDesignContext.Select.Entity

OtherContextMenus.MicrosoftDataEntityDesignContext.Select.Property

OtherContextMenus.MicrosoftDataEntityDesignContext.Select.Subtype

OtherContextMenus.MicrosoftDataEntityDesignContext.SelectAll

OtherContextMenus.MicrosoftDataEntityDesignContext.SelectAssociation

OtherContextMenus.MicrosoftDataEntityDesignContext.ShowinDiagram

OtherContextMenus.MicrosoftDataEntityDesignContext.ShowinModelBrowser

OtherContextMenus.MicrosoftDataEntityDesignContext.StoredProcedureMapping

OtherContextMenus.MicrosoftDataEntityDesignContext.TableMapping

OtherContextMenus.MicrosoftDataEntityDesignContext.UpdateModelfromDatabase

OtherContextMenus.MicrosoftDataEntityDesignContext.Validate

OtherContextMenus.MicrosoftDataEntityDesignContext.Zoom.10

OtherContextMenus.MicrosoftDataEntityDesignContext.Zoom.100

OtherContextMenus.MicrosoftDataEntityDesignContext.Zoom.125

OtherContextMenus.MicrosoftDataEntityDesignContext.Zoom.150

SKRÓT

OtherContextMenus.MicrosoftDataEntityDesignContext.Zoom.200

OtherContextMenus.MicrosoftDataEntityDesignContext.Zoom.25

OtherContextMenus.MicrosoftDataEntityDesignContext.Zoom.300

OtherContextMenus.MicrosoftDataEntityDesignContext.Zoom.33

OtherContextMenus.MicrosoftDataEntityDesignContext.Zoom.400

OtherContextMenus.MicrosoftDataEntityDesignContext.Zoom.50

OtherContextMenus.MicrosoftDataEntityDesignContext.Zoom.66

OtherContextMenus.MicrosoftDataEntityDesignContext.Zoom.75

OtherContextMenus.MicrosoftDataEntityDesignContext.Zoom.Custom

OtherContextMenus.MicrosoftDataEntityDesignContext.Zoom.ZoomIn

OtherContextMenus.MicrosoftDataEntityDesignContext.Zoom.ZoomOut

OtherContextMenus.MicrosoftDataEntityDesignContext.Zoom.ZoomToFit

View.EntityDataModelBrowser

View.EntityDataModelMappingDetails

Wykonywanie zapytań i wyszukiwanie jednostek

13.09.2018 • 5 minutes to read • [Edit Online](#)

W tym temacie omówiono różne sposoby, które można wyszukiwać dane przy użyciu platformy Entity Framework, w tym LINQ i metody Find. Techniki przedstawione w tym temacie stosuje się jednakowo do modeli utworzonych za pomocą Code First i projektancie platformy EF.

Wyszukiwanie jednostek przy użyciu zapytania

DbSet IDbSet implementować interfejs IQueryble i dlatego może służyć jako punktu wyjścia do pisania zapytania LINQ w odniesieniu do bazy danych. To nie jest właściwym miejscem dla szczegółowe omówienie LINQ, ale poniżej przedstawiono dwa proste przykłady:

```
using (var context = new BloggingContext())
{
    // Query for all blogs with names starting with B
    var blogs = from b in context.Blogs
                where b.Name.StartsWith("B")
                select b;

    // Query for the Blog named ADO.NET Blog
    var blog = context.Blogs
        .Where(b => b.Name == "ADO.NET Blog")
        .FirstOrDefault();
}
```

Należy pamiętać, że DbSet IDbSet zawsze tworzenie zapytań względem bazy danych i zawsze będzie obejmować komunikacji dwustronnej w bazie danych, nawet jeśli zwróconych już istnieje w kontekście. Zapytanie jest wykonywane względem bazy danych po:

- Są wyliczane przez **foreach** (C#) lub **dla każdego** — instrukcja (Visual Basic).
- Jego są wyliczane przez operację kolekcji takich jak **ToArray**, **ToDictionary**, lub **tolist** —.
- Operatory LINQ, takich jak **pierwszy** lub **wszelkie** są określone w najbardziej zewnętrznej część zapytania.
- Następujące metody są wywoływanie: **obciążenia** metody rozszerzenia na DbSet **DbEntityEntry.Reload** i **Database.ExecuteSqlCommand**.

Gdy wyniki są zwracane z bazy danych, obiekty, które nie istnieją w kontekście są dołączone do kontekstu. Jeśli obiekt jest już w kontekście, istniejący obiekt jest zwracany (są bieżące i oryginalne wartości właściwości obiektu we wpisie **nie** zastąpione wartościami bazy danych).

Podczas wykonywania zapytania jednostek, które zostały dodane do kontekstu, ale nie zostały zapisane w bazie danych nie są zwracane jako część zestawu wyników. Aby uzyskać dane, które znajduje się w kontekście, zobacz [dane lokalne](#).

Jeśli zapytanie zwraca żadnych wierszy z bazy danych, wynik będzie pustą kolekcję, zamiast **null**.

Wyszukiwanie jednostek przy użyciu kluczy podstawowych

Metody Find na DbSet używa wartość klucza podstawowego do podejmą próbę odnalezienia śledzone przez kontekst jednostki. Jeśli jednostka nie zostanie znaleziony w kontekście następnie kwerendę otrzymasz do bazy danych można znaleźć jednostki istnieje. Jeśli jednostka nie zostanie odnaleziona w kontekście lub w bazie danych, zwracana jest wartość null.

Znajdź różni się od przy użyciu zapytania na dwa istotne sposoby:

- Przesłania danych do bazy danych będą dostępne tylko w przypadku, jeśli jednostki z danym kluczem nie zostanie znaleziony w kontekście.
- Znajdź zwróci jednostek, które są w stanie dodany. Znajdź zwróci jednostek, które zostały dodane do kontekstu, ale nie zostały zapisane w bazie danych.

Wyszukiwanie jednostek według klucza podstawowego

Poniższy kod pokazuje do niektórych zastosowań wyszukiwania:

```
using (var context = new BloggingContext())
{
    // Will hit the database
    var blog = context.Blogs.Find(3);

    // Will return the same instance without hitting the database
    var blogAgain = context.Blogs.Find(3);

    context.Blogs.Add(new Blog { Id = -1 });

    // Will find the new blog even though it does not exist in the database
    var newBlog = context.Blogs.Find(-1);

    // Will find a User which has a string primary key
    var user = context.Users.Find("johndoe1987");
}
```

Wyszukiwanie jednostki według złożony klucz podstawowy

Entity Framework umożliwia jednostki mają klucze złożone — czyli klucz, który składa się z więcej niż jednej właściwości. Na przykład można mieć jednostki BlogSettings, która reprezentuje ustawienia użytkowników, dla konkretnego blogu. Ponieważ użytkownik będzie istniało tylko jeden BlogSettings dla każdego bloga, które można wykonać następujące akcje wybrała się klucz podstawowy BlogSettings kombinacją BlogId i nazwy użytkownika. Poniższy kod próbuje znaleźć BlogSettings z BlogId = 3, a nazwa użytkownika = "johndoe1987":

```
using (var context = new BloggingContext())
{
    var settings = context.BlogSettings.Find(3, "johndoe1987");
}
```

Należy pamiętać, że w przypadku kluczy złożonych należy użyć ColumnAttribute lub interfejsu API fluent Aby określić kolejność właściwości klucza złożonego. Podczas określania wartości, które tworzą klucz, wywołania do znalezienia należy użyć tej kolejności.

Metoda Load

26.10.2018 • 2 minutes to read • [Edit Online](#)

Istnieje kilka scenariuszy, w których możesz chcieć ładowanie jednostek z bazy danych do kontekstu, bez żadnego z tych jednostek natychmiast działania. Dobrym przykładem Trwa ładowanie jednostek dla wiązania danych, zgodnie z opisem w [dane lokalne](#). Typowym sposobem to jest Napisz zapytanie LINQ, a następnie wywołać tolist — na nim tylko po to, aby od razu odrzucić utworzona listę. Metoda rozszerzenia obciążenia działa podobnie jak tolist — z wyjątkiem tego, aby całkowicie uniknąć tworzenia listy.

Techniki przedstawione w tym temacie stosuje się jednakowo do modeli utworzonych za pomocą Code First i projektancie platformy EF.

Poniżej przedstawiono dwa przykłady użycia obciążenia. Pierwszy pochodzi z aplikacji powiązanie danych formularzy Windows, których obciążenia jest używana do wykonywania zapytań dla jednostek przed powiązaniem do kolekcji lokalnej, zgodnie z opisem w [dane lokalne](#):

```
protected override void OnLoad(EventArgs e)
{
    base.OnLoad(e);

    _context = new ProductContext();

    _context.Categories.Load();
    categoryBindingSource.DataSource = _context.Categories.Local.ToBindingList();
}
```

W drugim przykładzie pokazano przy użyciu ładowania do ładowania filtrowanym kolekcji powiązanych jednostek, zgodnie z opisem w [ładowanie powiązanych jednostek](#):

```
using (var context = new BloggingContext())
{
    var blog = context.Blogs.Find(1);

    // Load the posts with the 'entity-framework' tag related to a given blog
    context.Entry(blog)
        .Collection(b => b.Posts)
        .Query()
        .Where(p => p.Tags.Contains("entity-framework"))
        .Load();
}
```

Dane lokalne

13.09.2018 • 16 minutes to read • [Edit Online](#)

Uruchamianie zapytania LINQ bezpośrednio w odniesieniu do DbSet będą zawsze wysyłały zapytania do bazy danych, ale ma dostęp do danych, który jest aktualnie w pamięci przy użyciu właściwości DbSet.Local. Dostępne dodatkowe informacje, które służy do śledzenia EF dotyczących jednostek za pomocą metody DbContext.Entry i DbContext.ChangeTracker.Entries. Techniki przedstawione w tym temacie stosuje się jednakowo do modeli utworzonych za pomocą Code First i projektancie platformy EF.

Spójrz na dane lokalne przy użyciu lokalnego

Lokalną właściwość DbSet zapewnia prosty dostęp do zestawu jednostek, obecnie są śledzone od kontekstu, które nie zostały oznaczone jako usunięte. Uzyskiwanie dostępu do właściwości lokalnej nigdy nie powoduje, że zapytanie w celu wysłania do bazy danych. Oznacza to, że jest ona zwykle używana po zapytaniu zostało już wykonana. Metoda rozszerzenia obciążenia może służyć do wykonywania zapytania, tak, aby w kontekście śledzi wyniki. Na przykład:

```
using (var context = new BloggingContext())
{
    // Load all blogs from the database into the context
    context.Blogs.Load();

    // Add a new blog to the context
    context.Blogs.Add(new Blog { Name = "My New Blog" });

    // Mark one of the existing blogs as Deleted
    context.Blogs.Remove(context.Blogs.Find(1));

    // Loop over the blogs in the context.
    Console.WriteLine("In Local: ");
    foreach (var blog in context.Blogs.Local)
    {
        Console.WriteLine(
            "Found {0}: {1} with state {2}",
            blog.BlogId,
            blog.Name,
            context.Entry(blog).State);
    }

    // Perform a query against the database.
    Console.WriteLine("\nIn DbSet query: ");
    foreach (var blog in context.Blogs)
    {
        Console.WriteLine(
            "Found {0}: {1} with state {2}",
            blog.BlogId,
            blog.Name,
            context.Entry(blog).State);
    }
}
```

Gdyby dwóch blogów w bazie danych — "ADO.NET Blog" za pomocą BlogId 1 - i "Visual Studio Blog" za pomocą BlogId 2 oczekujemy można następujące dane wyjściowe:

```
In Local:
```

```
Found 0: My New Blog with state Added  
Found 2: The Visual Studio Blog with state Unchanged
```

```
In DbSet query:
```

```
Found 1: ADO.NET Blog with state Deleted  
Found 2: The Visual Studio Blog with state Unchanged
```

Obrazuje to trzy punkty:

- Nowego bloga "Moje nowego bloga" znajduje się w kolekcji lokalnej, nawet jeśli go nie został jeszcze zapisany w bazie danych. Ten blog zawiera klucza podstawowego równą zero, ponieważ baza danych nie ma jeszcze wygenerowany rzeczywistego klucza dla jednostki.
- Nie ma w blogu ADO.NET w kolekcji lokalnej, nawet jeśli jest nadal śledzony przez kontekst. Jest to spowodowane usunięciem z DbSet, w tym samym oznaczając je jako usunięty.
- Gdy DbSet służy do wykonywania zapytania blogu oznaczone do usunięcia (ADO.NET Blog) znajduje się w wynikach i nowego bloga (Mój Blog nowego) jeszcze nie zostały zapisane w bazie danych nie znajduje się w wynikach. Jest to spowodowane DbSet wykonuje zapytanie w bazie danych i wyników zwróconych zawsze odzwierciedla, co znajduje się w bazie danych.

Za pomocą lokalnego do dodawania i usuwania jednostek z kontekstu

Zwraca właściwości lokalnej na DbSet [observablecollection](#) — ze zdarzeniami podłączycieś w taki sposób, że pozostaje on synchronizowany z zawartością kontekstu. Oznacza to, że można dodać lub usuwane z kolekcji lokalnej lub DbSet jednostki. Oznacza to również, że zapytania, które łączą nowe jednostki w kontekście spowoduje kolekcji lokalnej aktualizowana przy użyciu tych jednostek. Na przykład:

```

using (var context = new BloggingContext())
{
    // Load some posts from the database into the context
    context.Posts.Where(p => p.Tags.Contains("entity-framework")).Load();

    // Get the local collection and make some changes to it
    var localPosts = context.Posts.Local;
    localPosts.Add(new Post { Name = "What's New in EF" });
    localPosts.Remove(context.Posts.Find(1));

    // Loop over the posts in the context.
    Console.WriteLine("In Local after entity-framework query: ");
    foreach (var post in context.Posts.Local)
    {
        Console.WriteLine(
            "Found {0}: {1} with state {2}",
            post.Id,
            post.Title,
            context.Entry(post).State);
    }

    var post1 = context.Posts.Find(1);
    Console.WriteLine(
        "State of post 1: {0} is {1}",
        post1.Name,
        context.Entry(post1).State);

    // Query some more posts from the database
    context.Posts.Where(p => p.Tags.Contains("asp.net")).Load();

    // Loop over the posts in the context again.
    Console.WriteLine("\nIn Local after asp.net query: ");
    foreach (var post in context.Posts.Local)
    {
        Console.WriteLine(
            "Found {0}: {1} with state {2}",
            post.Id,
            post.Title,
            context.Entry(post).State);
    }
}

```

Przy założeniu, że mieliśmy kilka wpisów oznakowane za pomocą programu "entity framework" i "asp.net" danych wyjściowych może wyglądać mniej więcej tak:

```

In Local after entity-framework query:
Found 3: EF Designer Basics with state Unchanged
Found 5: EF Code First Basics with state Unchanged
Found 0: What's New in EF with state Added
State of post 1: EF Beginners Guide is Deleted

In Local after asp.net query:
Found 3: EF Designer Basics with state Unchanged
Found 5: EF Code First Basics with state Unchanged
Found 0: What's New in EF with state Added
Found 4: ASP.NET Beginners Guide with state Unchanged

```

Obrazuje to trzy punkty:

- Nowy wpis "What's New in EF" który został dodany do lokalnej kolekcji staje się śledzone przez kontekst w stanie dodany. W związku z tym będzie można wstawić do bazy danych po wywołaniu funkcji SaveChanges.
- Wpis, który został usunięty z kolekcji lokalnej (Przewodnik dla początkujących EF) jest teraz oznaczony jako usunięty w kontekście. W związku z tym będzie można usunąć z bazy danych po wywołaniu funkcji

SaveChanges.

- Dodatkowe wpis (Przewodnik dla początkujących ASP.NET) ładowane do kontekstu z drugiego zapytania jest automatycznie dodawany do kolekcji lokalnej.

Jedno końcowe należy zwrócić uwagę na lokalny jest to, że ponieważ jej wydajności observablecollection — nie nadaje się doskonale dla dużej liczby obiektów. Dlatego jeśli masz do czynienia z tysiącami jednostek w kontekście usługi nie może być wskazane zastosowanie lokalnego.

Przy użyciu lokalnego powiązanie danych WPF

Właściwości lokalnej na DbSet może służyć bezpośrednio do wiązania danych w aplikacji WPF, ponieważ jest on wystąpieniem observablecollection —. Zgodnie z opisem w poprzedniej sekcji, oznacza to, że zostanie automatycznie zsynchronizowany z zawartością kontekstu i zawartość kontekst będzie automatycznie zsynchronizowany z nim. Uwaga: należy do wstępnego wypełniania kolekcji lokalnej przy użyciu danych w celu zapewnienia żadnych czynności, aby powiązać od czasu lokalnego, nigdy nie powoduje zapytania do bazy danych.

To nie jest w odpowiednim miejscu, aby uzyskać pełną próbę powiązanie danych WPF, ale są kluczowe elementy:

- Instalator źródło wiążące
- Powiązywanie lokalną właściwość do zestawu
- Wypełnienie lokalnego przy użyciu zapytania do bazy danych.

Powiązania WPF do właściwości nawigacji

Jeśli przeprowadzasz danych wzorzec/szczegół powiązania, możesz chcieć powiązać widok szczegółów z właściwością nawigacji jednej jednostki. Łatwy sposób na udostępnienie tej pracy jest używać ObservableCollection dla właściwości nawigacji. Na przykład:

```
public class Blog
{
    private readonly ObservableCollection<Post> _posts =
        new ObservableCollection<Post>();

    public int BlogId { get; set; }
    public string Name { get; set; }

    public virtual ObservableCollection<Post> Posts
    {
        get { return _posts; }
    }
}
```

Czyszczenie jednostek w SaveChanges przy użyciu lokalnego

W większości przypadków usunięte z właściwości nawigacji jednostki zostaną automatycznie oznaczone jako usunięte w kontekście. Na przykład usunięcie obiektu wpis z kolekcji Blog.Posts, a następnie opublikuj, nie zostaną automatycznie usunięte po wywołaniu funkcji SaveChanges. Jeśli jest potrzebna do usunięcia może potrzebne do znalezienia tych jednostek delegujące i oznacz je jako usunięte przed wywołaniem funkcji SaveChanges lub jako część zgodnym z przesłoniętą SaveChanges. Na przykład:

```
public override int SaveChanges()
{
    foreach (var post in this.Posts.Local.ToList())
    {
        if (post.Blog == null)
        {
            this.Posts.Remove(post);
        }
    }

    return base.SaveChanges();
}
```

Powyższy kod używa kolekcji lokalnej, aby znaleźć wszystkie wpisy i znaczniki, Wszyscy, którzy nie mają odwołania blogu jako usunięty. Konieczne jest wywołanie tolist —, ponieważ w przeciwnym razie kolekcji zostanie zmodyfikowany przez Usuń wywołanie, gdy jest wyliczany. W innych sytuacjach można tworzyć zapytania bezpośrednio w odniesieniu do lokalnych właściwości bez użycia tolist — najpierw.

Używanie powiązania danych lokalnych i ToBindingList dla Windows Forms

Formularze Windows nie obsługuje powiązanie danych o pełnej wierności bezpośrednio za pomocą observablecollection —. Jednak nadal umożliwia właściwości lokalne DbSet dla powiązania danych korzystać ze wszystkich zalet opisanego w poprzedniej sekcji. Jest to osiągane przy użyciu metody rozszerzenia ToBindingList, co powoduje utworzenie [IBindingList](#) implementacji wspierana przez lokalny observablecollection —.

To nie jest w odpowiednim miejscu, aby uzyskać pełną próbkę powiązanie danych formularzy Windows, ale są kluczowe elementy:

- Konfigurowanie obiektu źródło wiążące
- Powiązywanie właściwości lokalnej przy użyciu Local.ToBindingList() zestawu
- Wypełnij lokalnego przy użyciu zapytania do bazy danych

Pobieranie szczegółowych informacji na temat śledzonych jednostek

Metoda wpis Większość przykładów w tej serii zwrócić DbEntityEntry wystąpienie jednostki. Ten obiekt wpisu, następnie działa jako punktu wyjścia do zbierania informacji o jednostce, takie jak jego bieżący stan, a także do wykonywania operacji na jednostce, takich jak jawne ładowanie powiązanych jednostek.

Metody wpisy zwracają obiekty DbEntityEntry dla wielu lub wszystkich jednostek śledzony przez kontekst. Dzięki temu można zebrać informacje, lub wykonywać operacje na wiele jednostek, a nie tylko pojedynczy wpis. Na przykład:

```

using (var context = new BloggingContext())
{
    // Load some entities into the context
    context.Blogs.Load();
    context.Authors.Load();
    context.Readers.Load();

    // Make some changes
    context.Blogs.Find(1).Title = "The New ADO.NET Blog";
    context.Blogs.Remove(context.Blogs.Find(2));
    context.Authors.Add(new Author { Name = "Jane Doe" });
    context.Readers.Find(1).Username = "johndoe1987";

    // Look at the state of all entities in the context
    Console.WriteLine("All tracked entities: ");
    foreach (var entry in context.ChangeTracker.Entries())
    {
        Console.WriteLine(
            "Found entity of type {0} with state {1}",
            ObjectContext.GetObjectType(entry.Entity.GetType()).Name,
            entry.State);
    }

    // Find modified entities of any type
    Console.WriteLine("\nAll modified entities: ");
    foreach (var entry in context.ChangeTracker.Entries()
        .Where(e => e.State == EntityState.Modified))
    {
        Console.WriteLine(
            "Found entity of type {0} with state {1}",
            ObjectContext.GetObjectType(entry.Entity.GetType()).Name,
            entry.State);
    }

    // Get some information about just the tracked blogs
    Console.WriteLine("\nTracked blogs: ");
    foreach (var entry in context.ChangeTracker.Entries<Blog>())
    {
        Console.WriteLine(
            "Found Blog {0}: {1} with original Name {2}",
            entry.Entity.BlogId,
            entry.Entity.Name,
            entry.Property(p => p.Name).OriginalValue);
    }

    // Find all people (author or reader)
    Console.WriteLine("\nPeople: ");
    foreach (var entry in context.ChangeTracker.Entries<IPerson>())
    {
        Console.WriteLine("Found Person {0}", entry.Entity.Name);
    }
}

```

Można zauważać wprowadzamy klasy autora i czytnika, na przykład — zarówno w ramach tych zajęć implementować interfejs IPerson.

```

public class Author : IPerson
{
    public int AuthorId { get; set; }
    public string Name { get; set; }
    public string Biography { get; set; }
}

public class Reader : IPerson
{
    public int ReaderId { get; set; }
    public string Name { get; set; }
    public string Username { get; set; }
}

public interface IPerson
{
    string Name { get; }
}

```

Założmy, że mamy następujące dane w bazie danych:

Blog o BlogId = 1 i nazwa = "ADO.NET Blog"
 Blog o BlogId = 2, a nazwa = "Blog Visual Studio"
 Blog o BlogId = 3, a nazwa = "Blog programu .NET Framework"
 Tworzenie za pomocą wartość IDAutora = 1 i Name = "Jan Bloggs"
 Czytnik z ReaderId = 1 i Name = "Jan Kowalski"

Dane wyjściowe na uruchamianie kodu będą:

```

All tracked entities:
Found entity of type Blog with state Modified
Found entity of type Blog with state Deleted
Found entity of type Blog with state Unchanged
Found entity of type Author with state Unchanged
Found entity of type Author with state Added
Found entity of type Reader with state Modified

All modified entities:
Found entity of type Blog with state Modified
Found entity of type Reader with state Modified

Tracked blogs:
Found Blog 1: The New ADO.NET Blog with original Name ADO.NET Blog
Found Blog 2: The Visual Studio Blog with original Name The Visual Studio Blog
Found Blog 3: .NET Framework Blog with original Name .NET Framework Blog

People:
Found Person John Doe
Found Person Joe Bloggs
Found Person Jane Doe

```

Te przykłady ilustrują kilka punktów:

- Metody wpisy zwrócić wpisy dla jednostki w wszystkie stany, w tym usunięte. Ten element, aby porównać lokalnego, co wyklucza usunąć jednostki.
- Wpisy dla wszystkich typów jednostek są zwracane, gdy używana jest metoda wpisy nieogólnego. Gdy używana jest metoda ogólnego wpisy wpisy zwarcane są tylko dla jednostek, które są wystąpieniami typu ogólnego. To zostało użyte powyżej można pobrać Wpisy dla wszystkich naszych blogach. Pobierz wpisy dla wszystkich jednostek, które implementują IPerson została również. W tym przykładzie pokazano, typ ogólny nie musi być typu rzeczywistego jednostki.
- LINQ do obiektów może służyć do filtrowania wyników zwróconych. To zostało użyte powyżej można znaleźć

jednostki dowolnego typu, tak dùo, jak zostanù one zmienione.

Nale&yaacute; pamiętać, wyst&apień DbEntityEntry b&ędzie zawsze zawierać Jednostka inną niż null. Wpisów relacji i wpisy zastępcze nie są reprezentowane jako DbEntityEntry wyst&apień, dzi&ęki czemu nie ma potrzeby do filtrowania tych.

Bez śledzenia zapytań

13.09.2018 • 2 minutes to read • [Edit Online](#)

Czasami chcesz wrócić jednostek w wyniku zapytania, ale nie masz tych jednostek można śledzić w kontekście.

Może to spowodować lepszą wydajność podczas wykonywania zapytania dla dużej liczby obiektów w scenariuszach w trybie tylko do odczytu. Techniki przedstawione w tym temacie stosuje się jednakowo do modeli utworzonych za pomocą Code First i projektancie platformy EF.

Nowej metody rozszerzenia AsNoTracking umożliwia dowolny typ kwerendy będą uruchamiane w ten sposób. Na przykład:

```
using (var context = new BloggingContext())
{
    // Query for all blogs without tracking them
    var blogs1 = context.Blogs.AsNoTracking();

    // Query for some blogs without tracking them
    var blogs2 = context.Blogs
        .Where(b => b.Name.Contains(".NET"))
        .AsNoTracking()
        .ToList();
}
```

Pierwotne zapytania SQL

27.09.2018 • 4 minutes to read • [Edit Online](#)

Platformy Entity Framework umożliwia tworzenie zapytań za pomocą LINQ z Twoich zajęciach jednostki. Jednak może być razy, które chcesz uruchamiać zapytania przy użyciu surowego SQL bezpośrednio w bazie danych. W tym wywołaniem procedury składowane, które mogą być przydatne w przypadku Code First modeli, które aktualnie nie obsługują mapowanie procedur składowanych. Techniki przedstawione w tym temacie stosuje się jednakowo do modeli utworzonych za pomocą Code First i projektancie platformy EF.

Pisanie zapytań SQL dla jednostek

Metoda `SqlQuery` na `DbSet` umożliwia pierwotne zapytania SQL, który ma zostać zapisany, który zwróci wystąpień jednostek. Zwracanych obiektów będą śledzone przez kontekst, tak jak powinny, jeśli zostały zwrócone przez zapytanie LINQ. Na przykład:

```
using (var context = new BloggingContext())
{
    var blogs = context.Blogs.SqlQuery("SELECT * FROM dbo.Blogs").ToList();
```

Należy pamiętać, że podobnie jak w przypadku zapytań LINQ, zapytanie nie jest wykonywany do momentu wyliczane są wyniki — w powyższym przykładzie jest to zrobić za pomocą wywołania `ToList` —.

Należy zachować ostrożność przy każdym pierwotne zapytania SQL są zapisywane w dwóch powodów. Po pierwsze zapytanie mają być zapisywane upewnić się, że tylko zwraca jednostek, które są naprawdę żądanego typu. Na przykład podczas korzystania z funkcji, takich jak dziedziczenie jest łatwo można napisać zapytanie, które spowoduje utworzenie jednostek, które są nieprawidłowego typu CLR.

Po drugie niektóre typy pierwotne zapytania SQL ujawnia potencjalne zagrożenia bezpieczeństwa, zwłaszcza w części dotyczącej ataki przez wstrzygnięcie kodu SQL. Upewnij się, że użyto parametrów w zapytaniu w prawidłowy sposób, aby zabezpieczyć się przed takimi atakami.

Trwa ładowanie jednostek z procedur składowanych

`DbSet.SqlQuery` służy do ładowania jednostki w wynikach procedury składowanej. Na przykład poniższy kod wywołuje `dbo`. Procedura `GetBlogs` w bazie danych:

```
using (var context = new BloggingContext())
{
    var blogs = context.Blogs.SqlQuery("dbo.GetBlogs").ToList();
```

Parametry można też przekazać do procedury składowanej przy użyciu następującej składni:

```
using (var context = new BloggingContext())
{
    var blogId = 1;

    var blogs = context.Blogs.SqlQuery("dbo.GetBlogById @p0", blogId).Single();
```

Pisanie zapytań SQL dla typów innych niż jednostek

Zapytanie SQL zwraca wystąpień dowolnego typu, w tym typów pierwotnych, mogą być tworzone przy użyciu metody `SqlQuery` klasy bazy danych. Na przykład:

```
using (var context = new BloggingContext())
{
    var blogNames = context.Database.SqlQuery<string>(
        "SELECT Name FROM dbo.Blogs").ToList();
}
```

Kontekst nigdy nie będą śledzone wyniki zwrócone z `SqlQuery` w bazie danych, nawet jeśli obiekty są wystąpieniami typu jednostki.

Wysyłania poleceń pierwotnych do bazy danych

W poleceniach zapytań nie mogą być wysyłane do bazy danych przy użyciu metody `ExecuteSqlCommand` w bazie danych. Na przykład:

```
using (var context = new BloggingContext())
{
    context.Database.ExecuteSqlCommand(
        "UPDATE dbo.Blogs SET Name = 'Another Name' WHERE BlogId = 1");
}
```

Należy zauważyć, że wszelkie zmiany wprowadzone do danych w bazie danych przy użyciu `ExecuteSqlCommand` nieprzezroczysta dla kontekstu aż do jednostek są załadowane lub ponownie załadować z bazy danych.

Parametry wyjściowe

Jeśli używane są parametry wyjściowe, ich wartości nie będzie dostępne, dopóki wyniki zostały całkowicie odczytane. Jest to spowodowane bazowego zachowanie obiekt `DbDataReader`, zobacz [podczas pobierania danych przy użyciu elementu DataReader](#) Aby uzyskać więcej informacji.

Trwa ładowanie powiązanych jednostek

29.09.2018 • 9 minutes to read • [Edit Online](#)

Entity Framework obsługuje ładowanie powiązanych danych - eager, ładowania i powolne ładowanie jawne. Ładowanie na trzy sposoby. Techniki przedstawione w tym temacie stosuje się jednakowo do modeli utworzonych za pomocą Code First i projektancie platformy EF.

Trwa ładowanie eagerly

Wczesne ładowanie polega, według których kwerendy dla jednego typu obiektu również ładują powiązanych jednostek jako część zapytania. Wczesne ładowanie odbywa się przy użyciu metody `Include`. Na przykład poniższych zapytań załaduje naszych blogach i wszystkie wpisy związane z każdego bloga.

```
using (var context = new BloggingContext())
{
    // Load all blogs and related posts
    var blogs1 = context.Blogs
        .Include(b => b.Posts)
        .ToList();

    // Load one blogs and its related posts
    var blog1 = context.Blogs
        .Where(b => b.Name == "ADO.NET Blog")
        .Include(b => b.Posts)
        .FirstOrDefault();

    // Load all blogs and related posts
    // using a string to specify the relationship
    var blogs2 = context.Blogs
        .Include("Posts")
        .ToList();

    // Load one blog and its related posts
    // using a string to specify the relationship
    var blog2 = context.Blogs
        .Where(b => b.Name == "ADO.NET Blog")
        .Include("Posts")
        .FirstOrDefault();
}
```

Zwróć uwagę, że `Include` metody rozszerzenia w przestrzeni nazw `System.Data.Entity` dlatego upewnij się, że używasz tej przestrzeni nazw.

Trwa ładowanie eagerly wiele poziomów

Istnieje również możliwość eagerly obciążenia na wielu poziomach powiązanych jednostek. Poniższych zapytań Pokaż przykładów jak to zrobić dla kolekcji i odwołanie do właściwości nawigacji.

```

using (var context = new BloggingContext())
{
    // Load all blogs, all related posts, and all related comments
    var blogs1 = context.Blogs
        .Include(b => b.Posts.Select(p => p.Comments))
        .ToList();

    // Load all users, their related profiles, and related avatar
    var users1 = context.Users
        .Include(u => u.Profile.Avatar)
        .ToList();

    // Load all blogs, all related posts, and all related comments
    // using a string to specify the relationships
    var blogs2 = context.Blogs
        .Include("Posts.Comments")
        .ToList();

    // Load all users, their related profiles, and related avatar
    // using a string to specify the relationships
    var users2 = context.Users
        .Include("Profile.Avatar")
        .ToList();
}

```

Należy pamiętać, że nie jest obecnie możliwa filtrowanie danych powiązanych jednostek. Obejmują będą zawsze Przenieś wszystkie powiązane jednostki.

Ładowanie z opóźnieniem

Powolne ładowanie polega na zgodnie z którą jednostki lub kolekcję jednostek jest automatycznie ładowany z bazy danych, uzyskiwania dostępu do właściwości, odnoszące się do jednostki/jednostki po raz pierwszy. Korzystając z typów jednostki POCO, powolne ładowanie odbywa się tworzenia wystąpień typów pochodnych serwera proxy, a następnie zastępowanie właściwości wirtualnego można dodać punktów zaczepienia ładowania. Na przykład korzystając z klasy jednostki blogu, które są zdefiniowane poniżej, pokrewnych wpisów zostanie załadowany po raz pierwszy uzyskano dostęp do właściwości nawigacji wpisy:

```

public class Blog
{
    public int BlogId { get; set; }
    public string Name { get; set; }
    public string Url { get; set; }
    public string Tags { get; set; }

    public virtual ICollection<Post> Posts { get; set; }
}

```

Włącz powolne ładowanie wyłączone do serializacji

Powolne ładowanie i serializacji nie Mieszaj dobrze, a jeśli nie chcesz zachować ostrożność można zakończyć się wykonanie zapytania dotyczącego całą bazę danych, po prostu, ponieważ ładowania z opóźnieniem jest wyłączone. Serializatory większość pracy, uzyskując dostęp do każdej właściwości w wystąpieniu typu. Dostęp do właściwości wyzwala ładowania z opóźnieniem, więc podlegają serializacji z kolejnych jednostek. Na tych jednostkach właściwości są dostępne, a nawet więcej jednostek są ładowane. Jest dobrą praktyką, aby włączyć powolne ładowanie wyłączone przed serializacji jednostki. Poniższe sekcje pokazują, jak to zrobić.

Wyłączenie z opóźnieniem, ładowania dla właściwości określonej nawigacji

Powolne ładowanie kolekcji wpisy można wyłączyć, wprowadzając niewirtualną właściwość wpisy:

```
public class Blog
{
    public int BlogId { get; set; }
    public string Name { get; set; }
    public string Url { get; set; }
    public string Tags { get; set; }

    public ICollection<Post> Posts { get; set; }
}
```

Ładowanie wpisów w kolekcji można nadal osiągnąć przy użyciu wcześnie ładowanie (zobacz *Eagerly ładowania* powyżej) lub metody obciążenia (zobacz *jawnego ładowania* poniżej).

Wyłącz powolne ładowanie dla wszystkich obiektów

Powolne ładowanie można wyłączyć dla wszystkich jednostek w kontekście przez ustawienie flagi na właściwość konfiguracji. Na przykład:

```
public class BloggingContext : DbContext
{
    public BloggingContext()
    {
        this.Configuration.LazyLoadingEnabled = false;
    }
}
```

Ładowanie powiązanych jednostek nadal można osiągnąć, stosując wcześnie ładowanie (zobacz *Eagerly ładowania* powyżej) lub metody obciążenia (zobacz *jawnego ładowania* poniżej).

Jawne ładowanie

Nawet w przypadku powolne ładowanie wyłączone jest nadal możliwe opóźnieniem ładowanie powiązanych jednostek, ale muszą być wykonane za pomocą jawnego wywołania. Aby to zrobić, użyj metodę ładowania przy uruchamianiu powiązanej jednostki. Na przykład:

```
using (var context = new BloggingContext())
{
    var post = context.Posts.Find(2);

    // Load the blog related to a given post
    context.Entry(post).Reference(p => p.Blog).Load();

    // Load the blog related to a given post using a string
    context.Entry(post).Reference("Blog").Load();

    var blog = context.Blogs.Find(1);

    // Load the posts related to a given blog
    context.Entry(blog).Collection(p => p.Posts).Load();

    // Load the posts related to a given blog
    // using a string to specify the relationship
    context.Entry(blog).Collection("Posts").Load();
}
```

Należy pamiętać, że Reference — metoda powinien być używany, jeśli określona jednostka ma właściwość nawigacji, do innego pojedynczej jednostki. Z drugiej strony metoda kolekcji należy używać, jeśli określona jednostka ma właściwości nawigacji kolekcji innych podmiotów.

Stosowanie filtrów, gdy jawnie ładowanie powiązanych jednostek

Metoda zapytania zapewnia dostęp do podstawowego zapytania, które będą używać programu Entity Framework, gdy ładowanie powiązanych jednostek. Następnie służy LINQ do zastosowania filtrów do zapytania przed wykonaniem jego wywołaniem metody rozszerzenia LINQ, takich jak `ToList` —, obciążenia itp. Metoda zapytania mogą być używane z właściwości nawigacji kolekcji i odwołań, ale jest najbardziej użyteczne dla kolekcji, której może służyć do załadowania tylko część kolekcji. Na przykład:

```
using (var context = new BloggingContext())
{
    var blog = context.Blogs.Find(1);

    // Load the posts with the 'entity-framework' tag related to a given blog
    context.Entry(blog)
        .Collection(b => b.Posts)
        .Query()
        .Where(p => p.Tags.Contains("entity-framework"))
        .Load();

    // Load the posts with the 'entity-framework' tag related to a given blog
    // using a string to specify the relationship
    context.Entry(blog)
        .Collection("Posts")
        .Query()
        .Where(p => p.Tags.Contains("entity-framework"))
        .Load();
}
```

Z pomocą metody zapytania jest zazwyczaj najlepiej jest wyłączyć powolne ładowanie dla właściwości nawigacji. Jest tak, ponieważ w przeciwnym razie całej kolekcji może być ładowane automatycznie przez mechanizm powolne ładowanie przed lub po wykonaniu zapytania filtrowanego.

Należy pamiętać, że podczas relacji może być określony jako ciąg zamiast wyrażenia lambda, zwrócony element `IQueryable` nie ogólnego stosowania ciąg, a zatem metoda rzutowanie jest zwykle konieczna przed żadnych użytecznych może odbywać się z nim.

Liczba powiązanych jednostek bez ładowania je przy użyciu zapytań

Czasami warto wiedzieć, ile jednostek są powiązane z inną jednostką w bazie danych bez rzeczywiście ponoszenia kosztów ładowania tych jednostek. Metody zapytania przy użyciu metody liczba LINQ można to zrobić. Na przykład:

```
using (var context = new BloggingContext())
{
    var blog = context.Blogs.Find(1);

    // Count how many posts the blog has
    var postCount = context.Entry(blog)
        .Collection(b => b.Posts)
        .Query()
        .Count();
}
```

Zapisywanie danych przy użyciu platformy Entity Framework 6

13.09.2018 • 2 minutes to read • [Edit Online](#)

W tej sekcji znajdziesz informacje o możliwości śledzenia zmian i co się stanie, gdy należy wywołać funkcję `SaveChanges`, aby utrważyć zmiany do obiektów w bazie danych EF firmy.

Automatyczne wykrywanie zmian

13.09.2018 • 2 minutes to read • [Edit Online](#)

Korzystając z większości obiektów POCO ustalenia, jak jednostki został zmieniony (i w związku z tym aktualizacje, które muszą być wysyłane do bazy danych) odbywa się przez algorytm wykrywa zmiany. Wykryj zmiany działa zostało określony poprzez wykrycie różnic między bieżącej wartości właściwości jednostki i oryginalne wartości właściwości, które są przechowywane w migawce jednostki zapytania lub został dołączony. Techniki przedstawione w tym temacie stosuje się jednakowo do modeli utworzonych za pomocą Code First i projektancie platformy EF.

Domyślnie program Entity Framework wykonuje wykrywać zmiany automatycznie wywołanego następujących metod:

- `DbSet.Find`
- `DbSet.Local`
- `DbSet.Add`
- `DbSet.AddRange`
- `DbSet.Remove`
- `DbSet.RemoveRange`
- `DbSet.Attach`
- `DbContext.SaveChanges`
- `DbContext.GetValidationErrors`
- `DbContext.Entry`
- `DbChangeTracker.Entries`

Wyłączenie automatycznego wykrywania zmian

Jeśli śledzisz partii jednostek w kontekst można wywołać jedną z następujących metod wiele razy w pętli, może zostać znaczne ulepszenia wydajności przez wyłączenie wykrywania zmian na czas trwania pętli. Na przykład:

```
using (var context = new BloggingContext())
{
    try
    {
        context.Configuration.AutoDetectChangesEnabled = false;

        // Make many calls in a loop
        foreach (var blog in aLotOfBlogs)
        {
            context.Blogs.Add(blog);
        }
    }
    finally
    {
        context.Configuration.AutoDetectChangesEnabled = true;
    }
}
```

Nie należy zapominać ponownie włączyć funkcję wykrywania zmian po pętli — używaliśmy try/finally aby upewnić się, nawet wtedy, gdy kod w pętli zgłasza wyjątek zawsze jest ponownie włączone.

Alternatywa wyłączenie i ponowne włączenie pozostaw automatyczne wykrywanie zmian wyłączony na cały czas i albo Kontekst wywołania. `ChangeTracker.DetectChanges` jawnie lub użyj zmienić przerwami proxy śledzenia. Obie

te opcje są zaawansowane i łatwo wprowadzić drobne błędy do aplikacji więc używane ostrożnie.

Jeśli potrzebujesz dodać lub usunąć wiele obiektów z kontekstu, należy wziąć pod uwagę przy użyciu `DbSet.AddRange` i `DbSet.RemoveRange`. Ta metoda automatycznie Wykryj zmiany tylko jeden raz po ukończeniu operacji dodawania lub usuwania.

Praca z stany jednostki

07.12.2018 • 10 minutes to read • [Edit Online](#)

W tym temacie opisano sposób dodawania i dołączać jednostek do kontekstu i jak przetwarza je podczas SaveChanges Entity Framework. Śledzenie stanu jednostek, gdy są one połączone z kontekstem, ale w scenariuszach odłączone lub N-warstwowa można pozwolić, aby wiedzieć, jakie stanu jednostek EF powinien być w zajmuje się platformy Entity Framework. Techniki przedstawione w tym temacie stosuje się jednakowo do modeli utworzonych za pomocą Code First i projektancie platformy EF.

Stany jednostki i SaveChanges

Jednostki może być jeden z pięciu stanów zdefiniowanych przez wyliczenie EntityState. Te stany są następujące:

- Dodano: jednostka jest śledzony przez kontekst, ale jeszcze nie istnieje w bazie danych
- Niezmieniona: jednostka jest śledzony przez kontekst i istnieje w bazie danych i jego wartości właściwości nie zmieniły się od wartości w bazie danych
- Zmodyfikowane: jednostka jest śledzony przez kontekst i istnieje w bazie danych, a niektóre lub wszystkie wartości właściwości zostały zmodyfikowane
- Usunięto: jednostka jest śledzony przez kontekst i istnieje w bazie danych, ale została oznaczona do usunięcia z bazy danych następnym razem, gdy zostanie wywołana SaveChanges
- Odłączony: jednostka nie jest śledzony przez kontekst

SaveChanges wykonuje różne znaczenie w przypadku jednostek w różnych stanach:

- Niezmienione jednostki są nie korzystały SaveChanges. Aktualizacje nie są wysyłane do bazy danych dla obiektów w stanie niezmieniony.
- Dodano jednostki są wstawiane do bazy danych, a następnie stają się zmian podczas SaveChanges zwraca.
- Jednostki zmodyfikowane zostaną zaktualizowane w bazie danych, a następnie stają się zmian podczas SaveChanges zwraca.
- Usuniętych jednostek są usuwane z bazy danych, a następnie zostanie odłączony od kontekstu.

W poniższych przykładach pokazano sposób, w którym można zmienić stanu jednostki lub grafu jednostki.

Dodawanie nowej jednostki do kontekstu

Nowa jednostka można dodać do kontekstu poprzez wywołanie metody Add w DbSet. To powoduje umieszczenie jednostki w stanie Added, co oznacza, że zostanie on wstawiony do bazy danych przy następnym wywołaniu funkcji SaveChanges. Na przykład:

```
using (var context = new BloggingContext())
{
    var blog = new Blog { Name = "ADO.NET Blog" };
    context.Blogs.Add(blog);
    context.SaveChanges();
}
```

Innym sposobem dodania nowego obiektu dla kontekstu jest aby zmienić jego stan dodany. Na przykład:

```
using (var context = new BloggingContext())
{
    var blog = new Blog { Name = "ADO.NET Blog" };
    context.Entry(blog).State = EntityState.Added;
    context.SaveChanges();
}
```

Na koniec możesz dodać nową jednostkę do kontekstu przez podłączenie jej do innej jednostki, która jest już śledzony. Może to być, dodając nową jednostkę do właściwości nawigacji kolekcji innej jednostki lub przez ustawienie właściwości nawigacji odwołanie do innej jednostki, aby wskazywał nową jednostkę. Na przykład:

```
using (var context = new BloggingContext())
{
    // Add a new User by setting a reference from a tracked Blog
    var blog = context.Blogs.Find(1);
    blog.Owner = new User { UserName = "johndoe1987" };

    // Add a new Post by adding to the collection of a tracked Blog
    blog.Posts.Add(new Post { Name = "How to Add Entities" });

    context.SaveChanges();
}
```

Należy zauważyć, że dla wszystkich tych przykładach, jeśli jednostka dodawany ma odwołania do innych jednostek, które nie są jeszcze śledzone następnie te nowe jednostki zostanie także dodana do kontekstu i zostaną wstawione do bazy danych przy następnym wywołaniu funkcji `SaveChanges`.

Dołączanie istniejącej jednostki do kontekstu

Jeśli masz jednostki, który znasz już istnieje w bazie danych, ale który nie jest obecnie śledzony przez kontekst, a następnie będzie widnieć napis kontekstu do śledzenia jednostek przy użyciu metody `Dołącz` na `DbSet`. Jednostki będą w stanie niezmieniony w kontekście. Na przykład:

```
var existingBlog = new Blog { BlogId = 1, Name = "ADO.NET Blog" };

using (var context = new BloggingContext())
{
    context.Blogs.Attach(existingBlog);

    // Do some more work...

    context.SaveChanges();
}
```

Należy pamiętać, że nie zostaną wprowadzone zmiany do bazy danych jeśli `SaveChanges` nazywa się bez wykonania tej czynności manipulowania dołączonych jednostki. Jest tak, ponieważ jednostka jest w stanie niezmieniony.

Innym sposobem na dołączanie istniejącej jednostki do kontekstu jest aby zmienić jego stan `Unchanged`. Na przykład:

```
var existingBlog = new Blog { BlogId = 1, Name = "ADO.NET Blog" };

using (var context = new BloggingContext())
{
    context.Entry(existingBlog).State = EntityState.Unchanged;

    // Do some more work...

    context.SaveChanges();
}
```

Należy pamiętać, że dla obu tych przykładach Jeśli jednostki dołączany ma odwołania do innych jednostek, które nie są jeszcze śledzone następnie te nowe jednostki będą również dołączone do kontekstu w stanie niezmieniony.

Dołączanie do istniejącego, ale zmodyfikowano jednostkę do kontekstu

Jeśli masz jednostki, który znasz już istnieje w bazie danych, ale które mogą wprowadzono zmiany, a następnie będzie widnieć napis kontekstu, aby dołączyć jednostki i ustawić ich stan zmodyfikowane. Na przykład:

```
var existingBlog = new Blog { BlogId = 1, Name = "ADO.NET Blog" };

using (var context = new BloggingContext())
{
    context.Entry(existingBlog).State = EntityState.Modified;

    // Do some more work...

    context.SaveChanges();
}
```

Po zmianie stanu na zmodyfikowane właściwości jednostki zostanie oznaczona jako zmodyfikowana i wartości właściwości wysyłanych do bazy danych po wywołaniu funkcji SaveChanges.

Należy pamiętać, że jeśli jednostka dołączany zawiera odwołania do innych jednostek, które nie są jeszcze śledzone, następnie te nowe jednostki zostanie dołączony do kontekstu w stanie niezmieniony — ich nie zostaną automatycznie wprowadzone zmodyfikowane. Jeśli masz wiele jednostek, które muszą być oznaczone zmodyfikowane należy ustawić stan dla każdego z tych jednostek indywidualnie.

Zmiany stanu jednostki śledzonych

Można zmienić stanu jednostki, która jest już śledzony przez ustawienie właściwości stanu na jej wejścia. Na przykład:

```
var existingBlog = new Blog { BlogId = 1, Name = "ADO.NET Blog" };

using (var context = new BloggingContext())
{
    context.Blogs.Attach(existingBlog);
    context.Entry(existingBlog).State = EntityState.Unchanged;

    // Do some more work...

    context.SaveChanges();
}
```

Należy pamiętać, że wywołanie apletu Dodaj lub Dołącz do jednostki, która jest już śledzony umożliwia także zmiany stanu jednostki. Na przykład wywołanie dołączenia dla jednostki, która jest obecnie w stanie dodany zmieni jego stan na Unchanged.

Wstaw lub zaktualizuj wzorzec

Typowym wzorcem dla niektórych aplikacji jest dodawanie jednostki jako nowy (co powoduje wstawienie bazy danych) lub dołączyć jednostkę jako istniejące i oznaczyć ją jako zmodyfikowana (co powoduje aktualizację bazy danych) w zależności od wartości klucza podstawowego. Na przykład podczas korzystania z kluczy podstawowych całkowitą bazy danych, generowany jest wspólne dla obiektu z kluczem zero jako nowe i jednostki z kluczem różna od zera, jak istniejące. Ten wzorzec można osiągnąć przez ustawienie stanu jednostki, w oparciu o sprawdzenie wartości klucza podstawowego. Na przykład:

```
public void InsertOrUpdate(Blog blog)
{
    using (var context = new BloggingContext())
    {
        context.Entry(blog).State = blog.BlogId == 0 ?
            EntityState.Added :
            EntityState.Modified;

        context.SaveChanges();
    }
}
```

Należy pamiętać, że po zmianie stanu na zmodyfikowane właściwości jednostki zostanie oznaczona jako zmodyfikowana, i wartości właściwości wysyłanych do bazy danych po wywołaniu funkcji SaveChanges.

Praca z wartościami właściwości

13.09.2018 • 15 minutes to read • [Edit Online](#)

W większości przypadków zajmie się Entity Framework śledzeniem stanu, oryginalne wartości oraz bieżące wartości właściwości wystąpienia jednostki. Jednak może być pewnych przypadkach — takich jak rozłączonych scenariuszy —, w której chcesz wyświetlić i modyfikować EF ma informacje o właściwościach. Techniki przedstawione w tym temacie stosuje się jednakowo do modeli utworzonych za pomocą Code First i projektancie platformy EF.

Entity Framework śledzi informacje o dwóch wartości dla każdej właściwości śledzonej jednostki. Bieżąca wartość to, jak wskazuje nazwa, bieżąca wartość właściwości w obiekcie. Oryginalną wartość jest wartością, której właściwość jednostki pobieranych z bazy danych lub została dołączona do kontekstu.

Istnieją dwa mechanizmy ogólne dotyczące pracy z wartości właściwości:

- Wartość jednej właściwości mogą być uzyskane w silnie typizowany sposób przy użyciu metody właściwości.
- Do obiektu `DbPropertyValues` można odczytać wartości dla wszystkich właściwości jednostki. `DbPropertyValues` następnie działa jako obiekt słownika przypominającej umożliwiające odczyt i ustawianie wartości właściwości. Można ustawić wartości w obiekcie `DbPropertyValues` z wartości w innym obiekcie `DbPropertyValues` lub wartości w innym obiekcie, takich jak kopiowanie innej jednostki lub obiektu transferu prostych danych (DTO).

Poniższe sekcje pokazują przykłady przy użyciu zarówno mechanizmów powyżej.

Pobieranie i ustawianie bieżącej lub oryginalna wartość wybranej właściwości

W poniższym przykładzie pokazano, jak można odczytywać i następnie ustawić nową wartość bieżącą wartość właściwości:

```
using (var context = new BloggingContext())
{
    var blog = context.Blogs.Find(3);

    // Read the current value of the Name property
    string currentName1 = context.Entry(blog).Property(u => u.Name).CurrentValue;

    // Set the Name property to a new value
    context.Entry(name).Property(u => u.Name).CurrentValue = "My Fancy Blog";

    // Read the current value of the Name property using a string for the property name
    object currentName2 = context.Entry(blog).Property("Name").CurrentValue;

    // Set the Name property to a new value using a string for the property name
    context.Entry(blog).Property("Name").CurrentValue = "My Boring Blog";
}
```

Użyj właściwości `OriginalValue` zamiast właściwości `CurrentValue` odczytać lub ustawić oryginalnej wartości.

Należy pamiętać, że zwracana wartość jest wpisana jako "object", gdy ciąg jest używany do określenia nazwy właściwości. Z drugiej strony zwrócona wartość zdecydowanie jest wpisane, jeśli wyrażenie lambda jest używany.

Ustawienie wartości właściwości, takie jak to tylko spowoduje oznaczenie właściwości, jak modyfikacji, jeśli nowa wartość jest inna niż poprzednia wartość.

Gdy wartość właściwości jest ustawiana w ten sposób zmiany jest wykrywane automatycznie, nawet wtedy, gdy AutoDetectChanges jest wyłączona.

Pobieranie i ustawianie bieżącej wartości właściwości niezamapowane

Bieżąca wartość właściwości, która nie została zamapowana do bazy danych, również mogą być odczytywane.

Przykład niezamapowane właściwości może być właściwością RssLink na blogu. Ta wartość może być obliczona w oparciu o BlogId i w związku z tym nie mają być przechowywane w bazie danych. Na przykład:

```
using (var context = new BloggingContext())
{
    var blog = context.Blogs.Find(1);
    // Read the current value of an unmapped property
    var rssLink = context.Entry(blog).Property(p => p.RssLink).CurrentValue;

    // Use a string to specify the property name
    var rssLinkAgain = context.Entry(blog).Property("RssLink").CurrentValue;
}
```

Bieżąca wartość można również ustawić, jeśli właściwość udostępnia metody ustawiające.

Odczytywanie wartości właściwości niezamapowane jest przydatne w przypadku, gdy wykonywanie weryfikacji platformy Entity Framework niezamapowane właściwości. Z tego samego powodu bieżących wartości może odczytywać i ustawić dla właściwości jednostki, które nie są obecnie są śledzone przez kontekst. Na przykład:

```
using (var context = new BloggingContext())
{
    // Create an entity that is not being tracked
    var blog = new Blog { Name = "ADO.NET Blog" };

    // Read and set the current value of Name as before
    var currentName1 = context.Entry(blog).Property(u => u.Name).CurrentValue;
    context.Entry(blog).Property(u => u.Name).CurrentValue = "My Fancy Blog";
    var currentName2 = context.Entry(blog).Property("Name").CurrentValue;
    context.Entry(blog).Property("Name").CurrentValue = "My Boring Blog";
}
```

Pamiętaj, że oryginalne wartości nie są dostępne dla właściwości niezamapowane właściwości jednostki, które nie są śledzone przez kontekst.

Sprawdzanie, czy właściwość jest oznaczona jako zmodyfikowana

W poniższym przykładzie pokazano, jak sprawdzić, czy poszczególne właściwość jest oznaczona jako zmodyfikowana:

```
using (var context = new BloggingContext())
{
    var blog = context.Blogs.Find(1);

    var nameIsModified1 = context.Entry(blog).Property(u => u.Name).IsModified;

    // Use a string for the property name
    var nameIsModified2 = context.Entry(blog).Property("Name").IsModified;
}
```

Wartości zmodyfikowane właściwości są wysyłane jako aktualizacje w bazie danych po wywołaniu funkcji SaveChanges.

Oznaczanie jako zmodyfikowana właściwość

W poniższym przykładzie pokazano, jak wymusić pojedynczej właściwości, aby być oznaczona jako zmodyfikowana:

```
using (var context = new BloggingContext())
{
    var blog = context.Blogs.Find(1);

    context.Entry(blog).Property(u => u.Name).IsModified = true;

    // Use a string for the property name
    context.Entry(blog).Property("Name").IsModified = true;
}
```

Oznaczanie właściwość jako zmodyfikowane wymusza aktualizację można wysyłać do bazy danych dla właściwości po wywołaniu funkcji `SaveChanges` nawet, jeśli bieżąca wartość właściwości jest taka sama jak oryginalną wartość.

Nie jest obecnie możliwe zresetować pojedynczej właściwości nie można zmodyfikować po została oznaczona jako zmodyfikowana. Jest to coś, co firma Microsoft planuje się obsługiwać w przyszłej wersji.

Odczytywanie bieżącego, oryginalne i wartości z bazy danych dla wszystkich właściwości jednostki

W poniższym przykładzie pokazano, jak odczytać bieżące wartości, oryginalne wartości i wartości, które faktycznie w bazie danych dla wszystkich zamapowanych właściwości jednostki.

```
using (var context = new BloggingContext())
{
    var blog = context.Blogs.Find(1);

    // Make a modification to Name in the tracked entity
    blog.Name = "My Cool Blog";

    // Make a modification to Name in the database
    context.Database.SqlCommand("update dbo.Blogs set Name = 'My Boring Blog' where Id = 1");

    // Print out current, original, and database values
    Console.WriteLine("Current values:");
    PrintValues(context.Entry(blog).CurrentValues);

    Console.WriteLine("\nOriginal values:");
    PrintValues(context.Entry(blog).OriginalValues);

    Console.WriteLine("\nDatabase values:");
    PrintValues(context.Entry(blog).GetDatabaseValues());
}

public static void PrintValues(DbPropertyValues values)
{
    foreach (var propertyName in values.PropertyNames)
    {
        Console.WriteLine("Property {0} has value {1}",
                          propertyName, values[propertyName]);
    }
}
```

Bieżące wartości są wartościami, które obecnie zawiera właściwości jednostki. Oryginalne wartości są wartościami, które zostały odczytane z bazy danych, gdy badano jednostki. Wartości bazy danych są wartości,

ponieważ są one przechowywane w bazie danych. Pobieranie wartości z bazy danych jest przydatne w przypadku, gdy mógł ulec zmianie wartości w bazie danych, ponieważ badano jednostki, takie jak podczas edytowania równoczesne bazy danych zostały dokonane przez innych użytkowników.

Ustawianie bieżącej lub oryginalnej wartości z innego obiektu

Bieżące i oryginalne wartości śledzonych jednostki mogą być aktualizowane przez skopiowanie wartości z innego obiektu. Na przykład:

```
using (var context = new BloggingContext())
{
    var blog = context.Blogs.Find(1);
    var coolBlog = new Blog { Id = 1, Name = "My Cool Blog" };
    var boringBlog = new BlogDto { Id = 1, Name = "My Boring Blog" };

    // Change the current and original values by copying the values from other objects
    var entry = context.Entry(blog);
    entry.CurrentValues.SetValues(coolBlog);
    entry.OriginalValues.SetValues(boringBlog);

    // Print out current and original values
    Console.WriteLine("Current values:");
    PrintValues(entry.CurrentValues);

    Console.WriteLine("\nOriginal values:");
    PrintValues(entry.OriginalValues);
}

public class BlogDto
{
    public int Id { get; set; }
    public string Name { get; set; }
}
```

Uruchamianie kodu powyżej zostanie wydrukowana:

```
Current values:
Property Id has value 1
Property Name has value My Cool Blog

Original values:
Property Id has value 1
Property Name has value My Boring Blog
```

Ta technika jest czasami używana podczas aktualizowania jednostki przy użyciu wartości z wywołania usługi lub klienta w aplikacji n-warstwowej. Pamiętaj, że obiekt używany nie musi być tego samego typu co podmiot, tak długo, jak długo ma właściwości, których nazwy odpowiadają jednostki. W powyższym przykładzie wystąpienie BlogDTO służy do aktualizacji oryginalnych wartości.

Należy zauważyć, że tylko te właściwości, które są ustawione na różnych wartości podczas kopiowania z innego obiektu zostanie oznaczona jako zmodyfikowana.

Ustawianie bieżącej lub oryginalnej wartości ze słownika

Bieżące i oryginalne wartości śledzonych jednostki mogą być aktualizowane przez skopiowanie wartości ze słownika lub inne struktury danych. Na przykład:

```

using (var context = new BloggingContext())
{
    var blog = context.Blogs.Find(1);

    var newValues = new Dictionary<string, object>
    {
        { "Name", "The New ADO.NET Blog" },
        { "Url", "blogs.msdn.com/adonet" },
    };

    var currentValues = context.Entry(blog).CurrentValues;

    foreach (var propertyName in newValues.Keys)
    {
        currentValues[propertyName] = newValues[propertyName];
    }

    PrintValues(currentValues);
}

```

Użyj właściwości `OriginalValues` zamiast właściwości `CurrentValues` można ustawić oryginalnej wartości.

Ustawianie bieżącej lub oryginalnej wartości ze słownika przy użyciu właściwości

Alternatywa dla użycia `CurrentValues` lub `OriginalValues`, jak pokazano powyżej jest użycie metody właściwości można ustawić wartość każdej właściwości. Może to być pożąданie, gdy należy ustawić wartości właściwości złożonych. Na przykład:

```

using (var context = new BloggingContext())
{
    var user = context.Users.Find("johndoe1987");

    var newValues = new Dictionary<string, object>
    {
        { "Name", "John Doe" },
        { "Location.City", "Redmond" },
        { "Location.State.Name", "Washington" },
        { "Location.State.Code", "WA" },
    };

    var entry = context.Entry(user);

    foreach (var propertyName in newValues.Keys)
    {
        entry.Property(propertyName).CurrentValue = newValues[propertyName];
    }
}

```

W przykładzie powyżej złożonych właściwości są dostępne przy użyciu notacji nazw. Inne sposoby dostępu do właściwości złożonych zobacz dwie sekcje w dalszej części tego tematu sobie konkretnie o złożonych właściwości.

Tworzenie Sklonowany obiekt zawierający bieżące, oryginalny lub wartościami bazy danych

Zwróciła obiekt `DbPropertyValues` `CurrentValues`, `OriginalValues`, lub `GetDatabaseValues` może służyć do tworzenia klonu jednostki. Klonuj ten będzie zawierać wartości właściwości z obiektu `DbPropertyValues` użyty do jego utworzenia. Na przykład:

```
using (var context = new BloggingContext())
{
    var blog = context.Blogs.Find(1);

    var clonedBlog = context.Entry(blog).GetDatabaseValues().ToObject();
}
```

Należy pamiętać, że zwrócony obiekt nie jest obiektem i nie jest śledzony przez kontekst. Zwrócony obiekt nie ma również wszystkie relacje z zestawu do innych obiektów.

Sklonowany obiekt może być przydatna podczas rozwiązywania problemów związanych z równoczesnych aktualizacji w bazie danych, szczególnie gdy interfejs użytkownika, który obejmuje powiązywanie danych do obiektów określonego typu jest używany.

Pobieranie i ustawianie bieżącej lub oryginalnej wartości właściwości złożonych

Wartość cały obiekt złożony można odczytu i zestawu przy użyciu metody właściwości tak, jak można uzyskać właściwość typu pierwotnego. Ponadto możesz przejść do szczegółów obiektu złożonego i odczytu lub ustaw właściwości tego obiektu lub zagnieżdżony obiekt. Oto kilka przykładów:

```

using (var context = new BloggingContext())
{
    var user = context.Users.Find("johndoe1987");

    // Get the Location complex object
    var location = context.Entry(user)
        .Property(u => u.Location)
        .CurrentValue;

    // Get the nested State complex object using chained calls
    var state1 = context.Entry(user)
        .ComplexProperty(u => u.Location)
        .Property(l => l.State)
        .CurrentValue;

    // Get the nested State complex object using a single lambda expression
    var state2 = context.Entry(user)
        .Property(u => u.Location.State)
        .CurrentValue;

    // Get the nested State complex object using a dotted string
    var state3 = context.Entry(user)
        .Property("Location.State")
        .CurrentValue;

    // Get the value of the Name property on the nested State complex object using chained calls
    var name1 = context.Entry(user)
        .ComplexProperty(u => u.Location)
        .ComplexProperty(l => l.State)
        .Property(s => s.Name)
        .CurrentValue;

    // Get the value of the Name property on the nested State complex object using a single lambda expression
    var name2 = context.Entry(user)
        .Property(u => u.Location.State.Name)
        .CurrentValue;

    // Get the value of the Name property on the nested State complex object using a dotted string
    var name3 = context.Entry(user)
        .Property("Location.State.Name")
        .CurrentValue;
}

```

Użyj właściwości `OriginalValue` zamiast właściwości `CurrentValue`, aby pobrać lub ustawić oryginalnej wartości.

Należy pamiętać, że właściwość lub metoda `ComplexProperty` może służyć do dostępu do właściwości złożonej. Jednak metoda `ComplexProperty` należy użyć, jeśli chcesz przejść do obiektu złożonego za pomocą dodatkowych właściwości lub `ComplexProperty` wywołuje.

Za pomocą `DbPropertyValues` uzyskiwania dostępu do właściwości złożonych

Jeśli używasz `CurrentValues`, `OriginalValues` lub `GetDatabaseValues` można pobrać wszystkie bieżące, oryginalnym lub wartości bazy danych dla jednostki, wartości wszystkich właściwości złożone są zwracane jako obiekty zagnieżdżone `DbPropertyValues`. Te zagnieżdżone obiekty mogą następnie służyć do uzyskiwania wartości obiektu złożonego. Na przykład poniższa metoda zostanie wydrukowana wartości wszystkich właściwości, w tym wartości wszelkich złożonych i zagnieżdżonych właściwości złożonych.

```
public static void WritePropertyValues(string parentPropertyName, DbPropertyValues propertyValues)
{
    foreach (var propertyName in propertyValues.PropertyNames)
    {
        var nestedValues = propertyValues[propertyName] as DbPropertyValues;
        if (nestedValues != null)
        {
            WritePropertyValues(parentPropertyName + propertyName + ".", nestedValues);
        }
        else
        {
            Console.WriteLine("Property {0}{1} has value {2}",
                parentPropertyName, propertyName,
                propertyValues[propertyName]);
        }
    }
}
```

Aby wydrukować wszystkie bieżące wartości właściwości metoda będzie wywołana następująco:

```
using (var context = new BloggingContext())
{
    var user = context.Users.Find("johndoe1987");

    WritePropertyValues("", context.Entry(user).CurrentValue);
}
```

Obsługa konfliktów współbieżności

13.09.2018 • 8 minutes to read • [Edit Online](#)

Podjęto próbę zapisać jednostkę w bazie danych w niej danych nie zmienił się od jednostki mamy nadzieję, że został załadowany oznacza, że optymistycznie polega na współbieżność. Jeśli okazuje się, że dane zostały zmienione, a następnie zostanie zgłoszony wyjątek i należy rozwiązać konflikt, zanim spróbujesz ponownie zapisać. W tym temacie omówiono sposób obsługi wyjątków w programie Entity Framework. Techniki przedstawione w tym temacie stosuje się jednakowo do modeli utworzonych za pomocą Code First i projektancie platformy EF.

Ten wpis nie jest właściwym miejscem dla pełne omówienie optymistycznej współbieżności. Poniższe sekcje zakładają pewną wiedzę na temat rozpoznawania współbieżności i wyświetlanie wzorców do wykonywania typowych zadań.

Wiele z tych wzorców wprowadzić korzystanie z tematów omówionych w [Praca z wartościami właściwości](#).

Rozwiązywanie problemów współbieżności, gdy używana jest niezależne skojarzenia (gdzie klucza obcego nie została zamapowana na właściwość w jednostce) jest znacznie bardziej trudniejsze niż w przypadku używania powiązań kluczów obcych. W związku z tym jeśli czy rozdzielenie współbieżności w aplikacji zalecane jest zawsze mapy kluczów obcych do jednostki. Poniższe przykłady założono, że używasz skojarzenia klucza obcego.

`DbUpdateConcurrencyException` jest generowany przez `SaveChanges` po wykryciu wyjątku optymistycznej współbieżności podczas próby zapisania jednostki, która używa skojarzenia klucza obcego.

Rozwiązywanie wyjątków optymistycznej współbieżności za pomocą Reload (bazy danych wins)

Metoda ponowne załadowanie może służyć do zastąpienie bieżących wartości jednostki przy użyciu wartości teraz w bazie danych. Jednostka następnie zwykle znajduje się wstecz do użytkownika w pewnej postaci i użytkownik próbuje ponownie wprowadzić swoje zmiany i Zapisz ponownie. Na przykład:

```
using (var context = new BloggingContext())
{
    var blog = context.Blogs.Find(1);
    blog.Name = "The New ADO.NET Blog";

    bool saveFailed;
    do
    {
        saveFailed = false;

        try
        {
            context.SaveChanges();
        }
        catch (DbUpdateConcurrencyException ex)
        {
            saveFailed = true;

            // Update the values of the entity that failed to save from the store
            ex.Entries.Single().Reload();
        }
    } while (saveFailed);
}
```

Dobrym sposobem, aby zasymulować wyjątku współbieżności jest Ustaw punkt przerwania w wywołaniu funkcji SaveChanges, a następnie zmodyfikuj jednostki, do której jest zapisywany w bazie danych przy użyciu innego narzędzia, takie jak program SQL Management Studio. Można także wstawić wiersza przed SaveChanges można zaktualizować bazy danych bezpośrednio przy użyciu klasy SqlCommand. Na przykład:

```
context.Database.SqlCommand(  
    "UPDATE dbo.Blogs SET Name = 'Another Name' WHERE BlogId = 1");
```

Metoda wpisy na DbUpdateConcurrencyException zwraca DbEntityEntry wystąpień jednostki, których nie można zaktualizować. (Ta właściwość obecnie zawsze zwraca pojedynczej wartości dla współbieżności problemy. Jego może zwracać wiele wartości dla Wyjątki ogólne aktualizacji.) Zamiast w niektórych sytuacjach może być możliwa pobrać Wpisy dla wszystkich obiektów, które mogą być wymagane ponowne załadowanie z bazy danych, i wywołanie Załaduj ponownie dla każdego z nich.

Rozwiązywanie wyjątków optymistycznej współbieżności jako klienta usługi wins

W powyższym przykładzie używającą ponowne ładowanie jest czasami nazywane bazy danych wins lub magazynu usługi wins, ponieważ wartości w jednostce są zastępowane przez wartości z bazy danych. Czasami możesz nie przeciwieństwo i Zastąp wartości w bazie danych z wartościami, które aktualnie w jednostce. To jest czasami nazywane wins klienta i może odbywać się przez uzyskanie wartości bieżącej bazy danych i ustawisz je jako oryginalnych wartości jednostki. (Zobacz [Praca z wartościami właściwości](#) uzyskać informacji na temat bieżących i oryginalnych wartości.) Na przykład:

```
using (var context = new BloggingContext())  
{  
    var blog = context.Blogs.Find(1);  
    blog.Name = "The New ADO.NET Blog";  
  
    bool saveFailed;  
    do  
    {  
        saveFailed = false;  
        try  
        {  
            context.SaveChanges();  
        }  
        catch (DbUpdateConcurrencyException ex)  
        {  
            saveFailed = true;  
  
            // Update original values from the database  
            var entry = ex.Entries.Single();  
            entry.OriginalValues.SetValues(entry.GetDatabaseValues());  
        }  
  
        } while (saveFailed);  
}
```

Niestandardowe rozwiązanie wyjątki optymistycznej współbieżności

Czasami można połączyć ze sobą wartości obecnie w bazie danych z wartościami, które aktualnie w jednostce. Wymaga to zazwyczaj kilka niestandardowych interakcji logiki lub użytkownika. Na przykład mogą stanowić formularz do użytkownika, zawierającą bieżące wartości, wartości w bazie danych, a domyślny zestaw rozwiązane wartości. Użytkownik może następnie edytować rozwiązane wartości zgodnie z potrzebami i byłoby te rozwiązania wartości, które są zapisywane w bazie danych. Można to zrobić przy użyciu obiektów DbPropertyValues zwrócony

z CurrentValues i GetDatabaseValues przy uruchamianiu jednostki. Na przykład:

```
using (var context = new BloggingContext())
{
    var blog = context.Blogs.Find(1);
    blog.Name = "The New ADO.NET Blog";

    bool saveFailed;
    do
    {
        saveFailed = false;
        try
        {
            context.SaveChanges();
        }
        catch (DbUpdateConcurrencyException ex)
        {
            saveFailed = true;

            // Get the current entity values and the values in the database
            var entry = ex.Entries.Single();
            var currentValues = entry.CurrentValues;
            var databaseValues = entry.GetDatabaseValues();

            // Choose an initial set of resolved values. In this case we
            // make the default be the values currently in the database.
            var resolvedValues = databaseValues.Clone();

            // Have the user choose what the resolved values should be
            HaveUserResolveConcurrency(currentValues, databaseValues, resolvedValues);

            // Update the original values with the database values and
            // the current values with whatever the user choose.
            entry.OriginalValues.SetValues(databaseValues);
            entry.CurrentValues.SetValues(resolvedValues);
        }
    } while (saveFailed);
}

public void HaveUserResolveConcurrency(DbPropertyValues currentValues,
                                         DbPropertyValues databaseValues,
                                         DbPropertyValues resolvedValues)
{
    // Show the current, database, and resolved values to the user and have
    // them edit the resolved values to get the correct resolution.
}
```

Niestandardowe rozwiązywanie wyjątki optymistycznej współbieżności przy użyciu obiektów

Powyższy kod używa wystąpienia DbPropertyValues przekazywanie wokół bieżącego, bazy danych i rozwiązane wartości. Czasami może być łatwiejsze do użycia wystąpienia tego typu jednostki dla tego. Można to zrobić przy użyciu metod ToObject i SetValues DbPropertyValues. Na przykład:

```

using (var context = new BloggingContext())
{
    var blog = context.Blogs.Find(1);
    blog.Name = "The New ADO.NET Blog";

    bool saveFailed;
    do
    {
        saveFailed = false;
        try
        {
            context.SaveChanges();
        }
        catch (DbUpdateConcurrencyException ex)
        {
            saveFailed = true;

            // Get the current entity values and the values in the database
            // as instances of the entity type
            var entry = ex.Entries.Single();
            var databaseValues = entry.GetDatabaseValues();
            var databaseValuesAsBlog = (Blog)databaseValues.ToObject();

            // Choose an initial set of resolved values. In this case we
            // make the default be the values currently in the database.
            var resolvedValuesAsBlog = (Blog)databaseValues.ToObject();

            // Have the user choose what the resolved values should be
            HaveUserResolveConcurrency((Blog)entry.Entity,
                databaseValuesAsBlog,
                resolvedValuesAsBlog);

            // Update the original values with the database values and
            // the current values with whatever the user choose.
            entry.OriginalValues.SetValues(databaseValues);
            entry.CurrentValues.SetValues(resolvedValuesAsBlog);
        }
    } while (saveFailed);
}

public void HaveUserResolveConcurrency(Blog entity,
    Blog databaseValues,
    Blog resolvedValues)
{
    // Show the current, database, and resolved values to the user and have
    // them update the resolved values to get the correct resolution.
}

```

Praca z transakcją

13.09.2018 • 14 minutes to read • [Edit Online](#)

NOTE

EF6 począwszy tylko — funkcje, interfejsów API itp. z opisem na tej stronie zostały wprowadzone w programie Entity Framework 6. Jeśli używasz starszej wersji, niektóre lub wszystkie informacje, nie ma zastosowania.

W tym dokumencie opisano, za pomocą transakcji w EF6, w tym rozszerzeń, które dodaliśmy od EF5, aby ułatwić pracę z transakcją.

Domyślne działanie EF

We wszystkich wersjach programu Entity Framework, każde wykonanie **SaveChanges()** do wstawiania, aktualizacji lub Usuń w bazie danych w ramach będzie zawijany w tej operacji w transakcji. Ta transakcja obowiązuje tylko wystarczająco długi, aby wykonać operację, a następnie kończy. Podczas wykonywania innej takich operacji nowa transakcja jest uruchomiona.

Począwszy od platformy EF6 **Database.ExecuteSqlCommand()** domyślnie będzie zawijany polecenia w transakcji, jeśli odpowiedni klucz nie został już istnieje. Istnieją przeciążenia tej metody, które pozwalają na zastąpienie tego zachowania, jeśli chcesz, aby. Również w EF6 wykonywania procedur przechowywanych, które są uwzględnione w modelu za pośrednictwem interfejsów API, takich jak **ObjectContext.ExecuteFunction()** działa tak samo (z tą różnicą, że zachowanie domyślne w tej chwili przesłanianie).

W obu przypadkach poziom izolacji transakcji jest poziom izolacji, niezależnie od dostawcy bazy danych uwzględnia ustawnienie domyślne. Domyślnie na przykład w programie SQL Server to READ COMMITTED.

Entity Framework nie jest zawijany zapytań w ramach transakcji.

Ta funkcja domyślne jest odpowiednie dla wielu użytkowników i jeśli istnieje więc nie trzeba wykonywać dodatkowe czynności w EF6; Wystarczy napisać odpowiedni kod, tak jak zawsze.

Jednak niektóre użytkownicy muszą mieć większą kontrolę nad ich transakcje — to zagadnienie opisano w poniższych sekcjach.

Jak działają interfejsy API

Przed EF6 Entity Framework nalegała na otwieranie połączenia z bazą danych sam (spowodował on wyjątek została przekazana połączenia, który był wcześniej otwarty). Ponieważ transakcji można uruchomić tylko na otwartego połączenia, oznacza to, jedynym sposobem użytkownika można opakować kilka operacji jako jedna transakcja została programu **TransactionScope** lub użyj **ObjectContext.Connection** właściwości i wywoływanie start **Open()** i **BeginTransaction()** bezpośrednio na zwracanego **EntityConnection** obiekt. Dodatkowo wywołania interfejsu API, które skontaktować się z bazą danych zakończy się niepowodzeniem, jeśli transakcji zostało uruchomione na podstawowej połączenia z bazą danych na własną rękę.

NOTE

Ograniczenie tylko przyjmować połączenia zamknięte zostało usunięty w Entity Framework 6. Aby uzyskać więcej informacji, zobacz [zarządzania połączonymi](#).

Począwszy od platformy EF6 ramach teraz zawiera:

1. **Database.BeginTransaction()** : łatwiejszy dla użytkownika uruchomić i realizowania transakcji siebie w ramach istniejącego typu DbContext — co kilka operacji, które można połączyć w ramach tej samej transakcji i dlatego wszystkie zatwierdzone lub wszystkie wycofane jako jeden. Umożliwia użytkownikowi łatwiej określić poziom izolacji transakcji.
2. **Database.UseTransaction()** : pozwalający DbContext używać transakcji, która została uruchomiona poza programem Entity Framework.

Łącząc kilka operacji w jednej transakcji, w tym samym kontekście

Database.BeginTransaction() ma dwa zastąpienia — taki, który przyjmuje jawnego **IsolationLevel** i jedną, która nie przyjmuje żadnych argumentów i używa domyślnego IsolationLevel od podstawowego dostawcy bazy danych. Zwraca zarówno zastąpienia **DbContextTransaction** obiektu, który zapewnia **Commit()** i **Rollback()** metod, których wykonywanie zatwierdzenia i wycofywania na odpowiedni magazyn transakcja.

DbContextTransaction ma być usuwane, gdy została przekazana lub wycofana. Jeden łatwy sposób osiągnąć ten cel jest **using(...){...}** Składnia, która będzie automatycznie wywoływać **Dispose()** przypadku za pomocą bloku kończy:

```
using System;
using System.Collections.Generic;
using System.Data.Entity;
using System.Data.SqlClient;
using System.Linq;
using System.Transactions;

namespace TransactionsExamples
{
    class TransactionsExample
    {
        static void StartOwnTransactionWithinContext()
        {
            using (var context = new BloggingContext())
            {
                using (var dbContextTransaction = context.Database.BeginTransaction())
                {
                    try
                    {
                        context.Database.ExecuteSqlCommand(
                            @"UPDATE Blogs SET Rating = 5" +
                            " WHERE Name LIKE '%Entity Framework%'"
                        );
                }

                var query = context.Posts.Where(p => p.Blog.Rating >= 5);
                foreach (var post in query)
                {
                    post.Title += "[Cool Blog]";
                }

                context.SaveChanges();

                dbContextTransaction.Commit();
            }
            catch (Exception)
            {
                dbContextTransaction.Rollback();
            }
        }
    }
}
```

NOTE

Rozpoczęcie transakcji wymaga, że połączenie podstawowe magazynu jest otwarty. Dlatego wywołanie Database.BeginTransaction() spowoduje otwarcie połączenia, jeśli nie jest już otwarty. Jeśli DbContextTransaction otwarte połączenie następnie zostaną zamknięte go po wywołaniu metody Dispose().

Przekazywanie istniejącej transakcji do kontekstu

Czasami chcesz transakcji jest szersze w zakresie i który obejmuje operacje, w tej samej bazy danych, ale poza programem EF całkowicie. W tym celu należy otworzyć połączenie i rozpoczęć transakcję, samodzielnie i następnie poinformuj EF) do użycia już otwarte połączenia bazy danych i (b) korzystania z istniejącej transakcji dla tego połączenia.

W tym celu należy zdefiniować i użyć konstruktora w swojej klasy kontekstu, który dziedziczy z jednego typu DbContext konstruktorów przyjmujących i) istniejących parametrów połączenia i ii) contextOwnsConnection boolean.

NOTE

Flaga contextOwnsConnection musi być równa FAŁSZ, gdy wywołuje się, w tym scenariuszu. Jest to ważne, ponieważ informuje Entity Framework, go nie powinna zamykać połączenia po jej zakończeniu z nim (na przykład, zobacz wiersz do 4 opisane poniżej):

```
using (var conn = new SqlConnection("..."))
{
    conn.Open();
    using (var context = new BloggingContext(conn, contextOwnsConnection: false))
    {
    }
}
```

Ponadto należy rozpocząć transakcję, samodzielnie (łącznie IsolationLevel, jeśli chcesz uniknąć ustalenie domyślne) i umożliwić Entity Framework wiedzieć, że jest istniejącej transakcji, już uruchomiona w ramach połączenia (zobacz wiersz 33 poniżej).

To bezpłatna, można wykonać operacji bazy danych bezpośrednio na element SqlConnection, samego lub dla kontekstu DbContext. Wszystkie operacje są wykonywane w ramach jednej transakcji. Możesz skorzystać z odpowiedzialności potwierdzenia lub wycofania transakcji i wywoływanie metody Dispose() w nim, a także zamknięcia i usuwania połączenia z bazą danych. Na przykład:

```

using System;
using System.Collections.Generic;
using System.Data.Entity;
using System.Data.SqlClient;
using System.Linq;
using System.Transactions;

namespace TransactionsExamples
{
    class TransactionsExample
    {
        static void UsingExternalTransaction()
        {
            using (var conn = new SqlConnection("..."))
            {
                conn.Open();

                using (var sqlTxn = conn.BeginTransaction(System.Data.IsolationLevel.Snapshot))
                {
                    try
                    {
                        var sqlCommand = new SqlCommand();
                        sqlCommand.Connection = conn;
                        sqlCommand.Transaction = sqlTxn;
                        sqlCommand.CommandText =
                            @"UPDATE Blogs SET Rating = 5" +
                            " WHERE Name LIKE '%Entity Framework%'";
                        sqlCommand.ExecuteNonQuery();

                        using (var context =
                            new BloggingContext(conn, contextOwnsConnection: false))
                        {
                            context.Database.UseTransaction(sqlTxn);

                            var query = context.Posts.Where(p => p.Blog.Rating >= 5);
                            foreach (var post in query)
                            {
                                post.Title += "[Cool Blog]";
                            }
                            context.SaveChanges();
                        }

                        sqlTxn.Commit();
                    }
                    catch (Exception)
                    {
                        sqlTxn.Rollback();
                    }
                }
            }
        }
    }
}

```

Czyszczenie się na transakcję

Można przekazać wartości null do Database.UseTransaction() wyczyści Entity Framework wiedzę na temat bieżącej transakcji. Entity Framework będzie ani zatwierdzenia ani wycofać istniejącej transakcji, gdy to zrobisz, więc używać ostrożnie, i tylko wtedy, gdy wiesz, jest to, co chcesz zrobić.

Błędy w UseTransaction

Wyjątek z Database.UseTransaction() zostanie wyświetlony w przypadku przekazania transakcji po:

- Entity Framework jest już w istniejącej transakcji
- Entity Framework jest już działające w ramach elementu TransactionScope

- Obiekt połączenia w ramach transakcji przekazywany jest pusty. Oznacza to, że transakcja nie jest skojarzona z połączeniem — zazwyczaj jest to znak, że transakcja została już zakończona.
- Obiekt połączenia w ramach transakcji przekazywany jest niezgodna z połączenia programu Entity Framework.

Za pomocą transakcji z innymi funkcjami

Tej sekcji opisano szczegółowo sposób interakcji powyżej transakcji z:

- Elastyczność połączenia
- Metody asynchroniczne
- Transakcje elementu TransactionScope

Elastyczność połączenia

Nowej funkcji elastyczności połączenia nie działa z transakcjami zainicjowanego przez użytkownika. Aby uzyskać więcej informacji, zobacz [ponawiania strategii wykonywania](#).

Programowanie asynchroniczne

Podejście opisane w poprzednich sekcjach musi nie dalsze opcje lub ustawienia do pracy z [asynchronicznego zapytania i Zapisz metody](#). Należy jednak pamiętać, że, w zależności od tego, czy w ramach metod asynchronicznych, może to spowodować długotrwałych transakcji — które z kolei może spowodować zakleszczenia lub blokowania, czyli negatywnie wpływać na wydajność aplikacji ogólnej.

Transakcje elementu TransactionScope

Przed EF6 zalecaną metodą udostępniania większy zakres transakcji było jednocześnie używanie obiektu TransactionScope:

```

using System.Collections.Generic;
using System.Data.Entity;
using System.Data.SqlClient;
using System.Linq;
using System.Transactions;

namespace TransactionsExamples
{
    class TransactionsExample
    {
        static void UsingTransactionScope()
        {
            using (var scope = new TransactionScope(TransactionScopeOption.Required))
            {
                using (var conn = new SqlConnection("..."))
                {
                    conn.Open();

                    var sqlCommand = new SqlCommand();
                    sqlCommand.Connection = conn;
                    sqlCommand.CommandText =
                        @"UPDATE Blogs SET Rating = 5" +
                        " WHERE Name LIKE '%Entity Framework%'";
                    sqlCommand.ExecuteNonQuery();

                    using (var context =
                        new BloggingContext(conn, contextOwnsConnection: false))
                    {
                        var query = context.Posts.Where(p => p.Blog.Rating > 5);
                        foreach (var post in query)
                        {
                            post.Title += "[Cool Blog]";
                        }
                        context.SaveChanges();
                    }
                }

                scope.Complete();
            }
        }
    }
}

```

Element SqlConnection i Entity Framework zarówno używa otoczenia transakcji TransactionScope i dlatego można zatwierdzić ze sobą.

Począwszy od .NET 4.5.1 TransactionScope została zaktualizowana w celu również działać przy użyciu metod asynchronicznych za pomocą użytkowania [TransactionScopeAsyncFlowOption](#) wyliczenia:

```

using System.Collections.Generic;
using System.Data.Entity;
using System.Data.SqlClient;
using System.Linq;
using System.Transactions;

namespace TransactionsExamples
{
    class TransactionsExample
    {
        public static void AsyncTransactionScope()
        {
            using (var scope = new TransactionScope(TransactionScopeAsyncFlowOption.Enabled))
            {
                using (var conn = new SqlConnection("..."))
                {
                    await conn.OpenAsync();

                    var sqlCommand = new SqlCommand();
                    sqlCommand.Connection = conn;
                    sqlCommand.CommandText =
                        @"UPDATE Blogs SET Rating = 5" +
                        " WHERE Name LIKE '%Entity Framework%'";
                    await sqlCommand.ExecuteNonQueryAsync();

                    using (var context = new BloggingContext(conn, contextOwnsConnection: false))
                    {
                        var query = context.Posts.Where(p => p.Blog.Rating > 5);
                        foreach (var post in query)
                        {
                            post.Title += "[Cool Blog]";
                        }

                        await context.SaveChangesAsync();
                    }
                }
            }
        }
    }
}

```

Nadal istnieją pewne ograniczenia związane z podejściem TransactionScope:

- Wymaga programu .NET 4.5.1 lub nowszej, aby pracować z metod asynchronicznymi.
- Jeśli nie masz pewności, masz tylko jedno połączenie nie można używać w chmurze (cloud scenariuszy nie obsługują transakcji rozproszone).
- Nie można łączyć z podejściem Database.UseTransaction() w poprzedniej sekcji.
- Będzie ona zgłaszać wyjątki, jeśli wystawiać wszelkie DDL i nie włączono transakcji rozproszonych za pomocą usługi MSDTC.

Korzyści wynikające z podejścia TransactionScope:

- Zostanie automatycznie uaktualniona lokalnej transakcji do poziomu transakcji rozproszonej. Jeśli więcej niż jedno połączenie do określonej bazy danych lub połączyć połączenie z jedną bazą danych z połączeniem z inną bazą danych w ramach tej samej transakcji (Uwaga: konieczne jest posiadanie Usługa MSDTC skonfigurowane i umożliwiają transakcji rozproszonych, aby to działało).
- Łatwość programowania. Jeśli użytkownik sobie tego życzy transakcji otoczenia i rozdanych z niejawnie w tle, a nie jawnie w obszarze kontrolowanego następnie podejście TransactionScope może wlasnych możesz lepiej.

Podsumowując, za pomocą nowych Database.BeginTransaction() i API Database.UseTransaction() powyżej podejście TransactionScope nie jest już konieczne dla większości użytkowników. Jeśli będziesz nadal używać elementu TransactionScope następnie Pamiętaj o powyższe ograniczenia. Zaleca się przy użyciu metody opisane

w poprzednich sekcjach zamiast tego, gdzie to możliwe.

Sprawdzanie poprawności danych

16.01.2019 • 14 minutes to read • [Edit Online](#)

NOTE

EF4.1 począwszy tylko — funkcje, interfejsów API itp. z opisem na tej stronie zostały wprowadzone w programie Entity Framework 4.1. Jeśli używasz starszej wersji, niektóre lub wszystkie informacje, nie ma zastosowania

Zawartość na tej stronie są zaczerpnięte z artykułu pierwotnie napisane przez Julie Lerman (<http://thedatafarm.com>).

Entity Framework udostępnia wiele różnych funkcji sprawdzania poprawności, które może przekazywać za pośrednictwem interfejsu użytkownika w celu weryfikacji po stronie klienta lub można użyć do weryfikacji po stronie serwera. Po raz pierwszy przy użyciu kodu, można określić operacji sprawdzania poprawności przy użyciu adnotacji lub fluent konfiguracji interfejsu API. Dodatkowe sprawdzanie poprawności i bardziej złożone, można określić w kodzie i będzie działać, czy model ma za sobą kodu po pierwsze, najpierw modelu lub najpierw bazy danych.

Model

Zademonstruję walidacji przy użyciu prostego pary klasy: Blog i Post.

```
public class Blog
{
    public int Id { get; set; }
    public string Title { get; set; }
    public string BloggerName { get; set; }
    public DateTime DateCreated { get; set; }
    public virtual ICollection<Post> Posts { get; set; }
}

public class Post
{
    public int Id { get; set; }
    public string Title { get; set; }
    public DateTime DateCreated { get; set; }
    public string Content { get; set; }
    public int BlogId { get; set; }
    public ICollection<Comment> Comments { get; set; }
}
```

Adnotacje danych

Kod używa pierwsze adnotacje z zestawu System.ComponentModel.DataAnnotations jako jednym ze sposobów konfigurowania kodu pierwszej klasy. Wśród tych adnotacji są tymi, które zapewniają reguły, takie jak wymaganych, MaxLength, MinLength. Liczba aplikacji klienta .NET rozpoznaje również tych adnotacji, na przykład platformy ASP.NET MVC. Obie strony serwera i weryfikacji po stronie klienta przy użyciu tych adnotacji można osiągnąć. Na przykład możesz wymusić właściwości Tytuł blogu wymaganej właściwości.

```
[Required]  
public string Title { get; set; }
```

Bez dodatkowego kodu i znaczników zmiany w aplikacji istniejącej aplikacji MVC przeprowadzi weryfikację po stronie klienta, nawet dynamiczne tworzenie komunikat przy użyciu nazwy właściwości i adnotacji.

Create

Blog

Title
 The Title field is required.

BloggerName

DateCreated

We wpisie kopii metody tego widoku Create, platformy Entity Framework jest używany do zapisywania nowego bloga w bazie danych, ale weryfikacji po stronie klienta dla platformy MVC jest wyzwalany, zanim aplikacja osiągnie ten kod.

Weryfikacji po stronie klienta nie jest jednak punktem dowodu. Użytkownicy mogą mieć wpływ na funkcje przeglądarki lub co gorsza, haker może używać niektórych tricków w celu uniknięcia walidacji interfejsu użytkownika. Ale Entity Framework rozpoznaje adnotację wymagane również i zweryfikuje go.

Jest to prosty sposób testować tę aplikację można wyłączyć funkcję weryfikacji po stronie klienta w MVC. Można to zrobić w pliku web.config aplikacji MVC. W sekcji appSettings ma klucz dla ClientValidationEnabled. Ustawienie wartości false dla tego klucza uniemożliwi interfejs użytkownika wykonywania operacji sprawdzania poprawności.

```
<appSettings>  
  <add key="ClientValidationEnabled" value="false"/>  
  ...  
</appSettings>
```

Nawet w przypadku weryfikacji po stronie klienta, wyłączona uzyskasz tę samą odpowiedź do aplikacji. Komunikat o błędzie "Tytuł pole jest wymagane" będą wyświetlane jako. Z wyjątkiem teraz będą wynik weryfikacji po stronie serwera. Entity Framework będzie wykonywać sprawdzanie poprawności adnotację, wymagane (przed nawet bothers komplikacji i Wstaw polecenie, aby wysłać do bazy danych) i zwrócenie błędu do MVC, co spowoduje wyświetlenie komunikatu.

Interfejs Fluent API

Możesz użyć kodu przez pierwszy interfejs fluent API zamiast adnotacje do Uzyskaj tego samego klienta po stronie serwera po stronie sprawdzania poprawności. Zamiast użycia wymagane, czy pokazano tej weryfikacji MaxLength przy użyciu.

Fluent API konfiguracje są stosowane w kodzie najpierw jest Budowanie modelu z klas. Konfiguracje można wstrzymać poprzez zastąpienie metody OnModelCreating klasy DbContext. Poniżej przedstawiono konfigurację, określając, że właściwość BloggerName nie może być dłuższa niż 10 znaków.

```

public class BlogContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }
    public DbSet<Post> Posts { get; set; }
    public DbSet<Comment> Comments { get; set; }

    protected override void OnModelCreating(DbModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Blog>().Property(p => p.BloggerName).HasMaxLength(10);
    }
}

```

Błędy sprawdzania poprawności generowany na podstawie konfiguracji interfejsu API Fluent nie będzie automatycznie zasięg interfejsu użytkownika, ale można przechwytywać go w kodzie i następnie odpowiedź do niego odpowiednio.

Oto niektóre błąd kodu w klasie BlogController aplikacji, która przechwytuje ten błąd sprawdzania poprawności, gdy Entity Framework próbuje zapisać blog BloggerName, która przekracza maksymalną liczbę znaków 10 obsługi wyjątków.

```

[HttpPost]
public ActionResult Edit(int id, Blog blog)
{
    try
    {
        db.Entry(blog).State = EntityState.Modified;
        db.SaveChanges();
        return RedirectToAction("Index");
    }
    catch(DbEntityValidationException ex)
    {
        var error = ex.EntityValidationErrors.First().ValidationErrors.First();
        this.ModelState.AddModelError(error.PropertyName, error.ErrorMessage);
        return View();
    }
}

```

Sprawdzanie poprawności nie Pobierz automatycznie przekazywane do widoku, dlatego dodatkowy kod, który używa ModelState.AddModelError jest używany. Daje to gwarancję, że szczegóły błędu przenoszone do widoku, który zostanie użyty ValidationMessageFor HtmlHelper do wyświetlenia błędu.

```
@Html.ValidationMessageFor(model => model.BloggerName)
```

IValidableObject

IValidableObject jest interfejsem, który znajduje się w System.ComponentModel.DataAnnotations. Mimo że nie jest częścią interfejsu API programu Entity Framework, można nadal korzystać z go do weryfikacji po stronie serwera w Twoich zajęciach Entity Framework. IValidableObject udostępnia metodę sprawdzania poprawności, która wywoła Entity Framework podczas SaveChanges lub możesz wywołać samodzielnie ilekroć, którą chcesz zweryfikować klasy.

Konfiguracje, takie jak wymagane i MaxLength wykonać validaton według jednego pola. W metodzie sprawdzania poprawności może być jeszcze bardziej złożonej logiki, na przykład porównanie dwóch pól.

W poniższym przykładzie klasa blogu została rozszerzony do zaimplementowania IValidableObject, a następnie podaj regułę, która tytuł i BloggerName nie może być zgodna.

```

public class Blog : IValidatableObject
{
    public int Id { get; set; }
    [Required]
    public string Title { get; set; }
    public string BloggerName { get; set; }
    public DateTime DateCreated { get; set; }
    public virtual ICollection<Post> Posts { get; set; }

    public IEnumerable<ValidationResult> Validate(ValidationContext validationContext)
    {
        if (Title == BloggerName)
        {
            yield return new ValidationResult
                ("Blog Title cannot match Blogger Name", new[] { "Title", "BloggerName" });
        }
    }
}

```

Konstruktor ValidationResult przyjmuje ciąg reprezentujący komunikat o błędzie i Tablica ciągów, które reprezentują nazwy elementów członkowskich, które są skojarzone z weryfikacją. Ponieważ ta weryfikacja, tytuł i BloggerName, zwracane są obie nazwy właściwości.

W przeciwieństwie do sprawdzania poprawności, udostępniony przez interfejs Fluent API ten wynik sprawdzania poprawności zostanie rozpoznane przez widok, a program obslugi wyjątku, który wcześniej używane do dodawania błąd do ModelState nie jest konieczne. Ponieważ obie nazwy właściwości ustawione w ValidationResult, MVC HtmlHelpers wyświetlić komunikat o błędzie dla obu tych właściwości.

Edit

The screenshot shows an 'Edit' form for a 'Blog' entity. The form has three fields: 'Title' (containing 'Julie'), 'BloggerName' (containing 'Julie'), and 'DateCreated' (containing '3/11/2011 12:00:00 AM'). Below each field, there is a red error message: 'Blog Title cannot match Blogger Name'. At the bottom of the form is a 'Save' button.

DbContext.ValidateEntity

Kontekst DbContext ma metodę z możliwością o nazwie ValidateEntity. Po wywołaniu funkcji SaveChanges Entity Framework będzie wywoływać tej metody dla każdej jednostki w pamięci podręcznej, którego stan nie jest bez zmian. Logikę weryfikacji można umieścić bezpośrednio w tutaj lub nawet użycie tej metody do wywołania, na przykład metoda Blog.Validate dodany w poprzedniej sekcji.

Oto przykład zastąpienia ValidateEntity, która weryfikuje nowych wpisów, aby upewnić się, że tytuł wpisu nie został już użyty. Go najpierw sprawdza, czy jednostka jest wpis i że jego stan zostanie dodany. Jeśli tak jest rzeczywiście, następnie szuka w bazie danych, aby sprawdzić, czy jest już wpis o tej samej nazwie. Jeśli istniejący wpis już istnieje, jest tworzony nowy DbEntityValidationResult.

DbEntityValidationResult przechowuje DbEntityEntry i ICollection DbValidationErrors dla pojedynczej jednostki. Na początku tej metody DbEntityValidationResult zostanie uruchomiony, a następnie wszelkie błędy, które zostały wykryte zostaną dodane do swojej kolekcji ValidationErrors.

```

protected override DbEntityValidationResult ValidateEntity (
    System.Data.Entity.Infrastructure.DbEntityEntry entityEntry,
    IDictionary<object, object> items)
{
    var result = new DbEntityValidationResult(entityEntry, new List<DbValidationError>());
    if (entityEntry.Entity is Post && entityEntry.State == EntityState.Added)
    {
        Post post = entityEntry.Entity as Post;
        //check for uniqueness of post title
        if (Posts.Where(p => p.Title == post.Title).Count() > 0)
        {
            result.ValidationErrors.Add(
                new System.Data.Entity.Validation.DbValidationResult("Title",
                "Post title must be unique."));
        }
    }

    if (result.ValidationErrors.Count > 0)
    {
        return result;
    }
    else
    {
        return base.ValidateEntity(entityEntry, items);
    }
}

```

Jawnie wyzwalania sprawdzania poprawności

Po wywołaniu funkcji `SaveChanges` wyzwala wszystkie operacje sprawdzania poprawności omówione w tym artykule. Ale nie trzeba polegać na `SaveChanges`. Możesz sprawdzić w innym miejscu w aplikacji.

`DbContext.GetValidationErrors` wyzwoli wszystkie operacje sprawdzania poprawności, te określone przez adnotacje lub interfejsu API Fluent, sprawdzanie poprawności utworzonych w `IValidatableObject` (na przykład `Blog.Validate`) i sprawdeń wykonywane w `DbContext.ValidateEntity` Metoda.

Poniższy kod wywoła `GetValidationErrors` na bieżącym wystąpieniu typu `DbContext`. `ValidationErrors` są pogrupowane według typu jednostki do `DbValidationResults`. Kod wykonuje iterację najpierw za pomocą `DbValidationResults` zwracany przez metodę, a następnie za pomocą każdego `ValidationResult` wewnętrz.

```

foreach (var validationResults in db.GetValidationErrors())
{
    foreach (var error in validationResults.ValidationErrors)
    {
        Debug.WriteLine(
            "Entity Property: {0}, Error {1}",
            error.PropertyName,
            error.ErrorMessage);
    }
}

```

Inne uwagi dotyczące korzystania z weryfikacji

Poniżej przedstawiono kilka kwestii do rozważenia podczas korzystania z programu Entity Framework sprawdzania poprawności:

- Powolne ładowanie jest wyłączone podczas sprawdzania poprawności.
- EF zostanie przeprowadzona Weryfikacja adnotacje danych na temat właściwości — zamapowany (właściwości, które nie są zamapowane na kolumnę w bazie danych).

- Sprawdzanie poprawności jest wykonywane po zmiany są wykrywane podczas SaveChanges. W przypadku wprowadzenia zmian podczas sprawdzania poprawności jest odpowiedzialny za powiadomienie śledzenie zmian.
- DbUnexpectedValidationException jest generowany, jeśli występują błędy podczas sprawdzania poprawności.
- Aspektami, które platformy Entity Framework zawiera w modelu (maksymalną długość wymaganych itp.) spowoduje, że sprawdzanie poprawności, nawet jeśli nie ma adnotacji danych w Twoich zajęciach i/lub projektancie platformy EF został użyty do utworzenia modelu.
- Reguły pierwszeństwa:
 - Wywołania interfejsu API Fluent zastąpienia odpowiedniego adnotacji danych
- Kolejność wykonywania:
 - Właściwości sprawdzania poprawności jest wcześniejsza niż typ sprawdzania poprawności
 - Sprawdzanie poprawności typu tylko wtedy, gdy właściwość weryfikacja zakończy się powodzeniem
- Jeśli właściwość jest złożony jego weryfikacji będzie również udostępniać program:
 - Właściwość poziom weryfikacji we właściwościach typu złożonego
 - Sprawdzanie poziomu typu na typ złożony, w tym weryfikacji IValidatableObject na typ złożony

Podsumowanie

Sprawdzanie poprawności interfejsu API platformy Entity Framework odgrywa bardzo dobrze za pomocą weryfikacji po stronie klienta w MVC, ale nie trzeba polegać na weryfikacji po stronie klienta. Zajmie się weryfikacją po stronie serwera dla DataAnnotations lub konfiguracje zostały zastosowane przy użyciu kodu pierwszy Fluent interfejsu API platformy Entity Framework.

Przedstawiono także szereg punkty rozszerzeń dostosowywania zachowania przy użyciu interfejsu IValidatableObject lub naciśnij tę wartość do metody DbContet.ValidateEntity. I te ostatnie dwa sposoby sprawdzania poprawności są dostępne za pośrednictwem typu DbContext, czy użyć Code First, pierwszego modelu lub bazy danych pierwszego przepływu pracy do opisania model koncepcyjny.

Zasoby programu Entity Framework

13.09.2018 • 2 minutes to read • [Edit Online](#)

W tym miejscu znajdziesz, łączy i odwołań do dodatkowych informacji związanych z programów EF, takich jak blogi, dostawców innych firm, narzędzi i rozszerzeń, analizy przypadków: miejsca itp.

Blogi dotyczące programu Entity Framework

13.09.2018 • 2 minutes to read • [Edit Online](#)

Oprócz dokumentacji produktu tych blogi może być źródłem przydatne informacje na temat platformy Entity Framework:

Blogi zespołu EF

- [Blog programu .NET — Tag: Entity Framework](#)
- [Blog ADO.NET \(nie jest już w użyciu\)](#)
- [Blog projektowania EF \(nie jest już w użyciu\)](#)

Autorów blogów zespołu EF bieżące i wcześniejsze

- [Arthur Vickers](#)
- [Brice Lambson](#)
- [Diego Vega](#)
- [Rowan Miller](#)
- [Pawel Kadluczka](#)
- [Alex James](#)
- [Zlatko Michailov](#)

Blogerom społeczności EF

- [Julie Lerman](#)
- [Shawn Wildermuth](#)

Analizy przypadków firmy Microsoft dla programu Entity Framework

13.09.2018 • 7 minutes to read • [Edit Online](#)

Analizy przypadków na tej stronie zaznacz kilka projektów rzeczywistych produkcji, które zostały użyte Entity Framework.

NOTE

Szczegółowe wersje tych przypadków nie są już dostępne w witrynie internetowej firmy Microsoft. W związku z tym linki zostały usunięte.

Epicor

Epicor jest firmy zajmującej się oprogramowaniem globalnego dużych (z ponad 400 deweloperzy), który zajmuje się opracowywaniem rozwiązań planowania zasobów przedsiębiorstwa (ERP) dla przedsiębiorstw w więcej niż 150 krajach. Ich produktem, Epicor 9 opiera się na Service-Oriented architektury (SOA) przy użyciu programu .NET Framework. W obliczu wiele żądań klientów, aby zapewnić obsługę Language Integrated Query (LINQ) i chce również zmniejszyć obciążenie na serwerach SQL zaplecza, zespół postanowił uaktualnienie do programu Visual Studio 2010 oraz programu .NET Framework 4.0. Za pomocą programu Entity Framework 4.0, mieli osiągnięciu tych celów, a także znacznie upraszczają projektowania i utrzymania. W szczególności pomocy technicznej programu Entity Framework sformatowanego T4 pozwoliły im przejęciu pełnej kontroli nad ich wygenerowanego kodu oraz automatyczne kompilowanie w funkcji zapisywania wydajności, takich jak wstępnie skompilowanym zapytania i buforowanie.

"Firma Microsoft prowadzone niektóre testy wydajności ostatnio z istniejącym kodem i byliśmy w stanie zredukować żądania do programu SQL Server o 90%. Jest ono spowodowane ADO.NET Entity Framework 4." — Badania nad produktami Erik Johnson, wiceprezes ds.

Wiarygodności rozwiązania

Nabycia planowania wydarzeń system oprogramowania, który będzie utrudniać ich utrzymywanie i rozszerzać w długim okresie, wiarygodności rozwiązania używane programu Visual Studio 2010 do ponownego napisania zaawansowane i łatwe w użyciu Rich Internet Application wbudowane w program Silverlight 4. Przy użyciu platformy .NET RIA Services mieli szybkie tworzenie warstwy usług, na podstawie programu Entity Framework, która uniknąć duplikowania kodu i mogą uzyskać typowe sprawdzania poprawności i logika uwierzytelniania w warstwach.

"Firma Microsoft zostały sprzedane na platformie Entity Framework została wprowadzona, gdy programu Entity Framework 4 okazał się jeszcze lepsze. Zwiększoną narzędzi i łatwiej jest je modyfikować pliki edmx, które definiują modelu koncepcyjnego, model magazynu i mapowanie między tymi modelami... Za pomocą programu Entity Framework, można uzyskać tej warstwy dostępu do danych, pracy w ciągu dnia — i skompiluj go się, jak można przejść. Entity Framework jest naszym warstwy dostępu do danych de facto; Nie wiem, dlaczego każda osoba w takich sytuacjach przydałaby z niej korzystać." — Jan McBride, starszy Deweloper

Wyświetlanie NEC rozwiązania Ameryki

Firma NEC chciała wejścia na rynek cyfrowego reklamy oparte na miejscu za pomocą rozwiązania do korzyści

reklamodawców i właściciele sieci i zwiększyć swoje własne przychody. Aby to zrobić, on uruchamiany parę automatyzowanie procesów ręcznych wymagane dla kampanii ad tradycyjnych aplikacji sieci web. Witryny zostały zbudowane przy użyciu platformy ASP.NET, program Silverlight 3, wywołań AJAX i WCF, wraz z programu Entity Framework w warstwie dostępu do danych na komunikowanie się z programu SQL Server 2008.

"Z programem SQL Server, firma Microsoft uznało, uzyskujemy przepływności potrzebowaliśmy, aby obsługiwać reklamodawców i sieci z informacji w czasie rzeczywistym i niezawodności, aby zapewnić informacje zawarte w naszej aplikacji o krytycznym znaczeniu będzie zawsze dostępny" — Mike Corcoran Dyrektor ds. IT

Wymiary Darwin

Przy użyciu technologii szeroki zakres firmy Microsoft, zespół Darwin określone do utworzenia Evolver — portal online awatara, który klientów można użyć do tworzenia niesamowitych, w rzeczywistości awatary do użytku w gry, animacji i stron sieci społecznościowych. Za pomocą korzyści wydajności platformy Entity Framework i ściąganie w składnikach, takich jak Windows Workflow Foundation (WF) i Windows Server AppFabric (pamięć podrzczna wysoce skalowalnych aplikacji w pamięci) zespół był w stanie dostarczać wspaniałe produktu w 35% niższy czas opracowywania. Pomimo mających członków zespołu Podziel w wielu krajach zespołu, zgodnie z procesem zwinne Wytwarzanie oprogramowania przy użyciu cotygodniowych aktualizacjach.

"Firma Microsoft nie należy utworzyć technologii sake tej technologii. Autostart jest niezwykle istotne, że możemy korzystać z technologii, który oszczędza czas i pieniądze .NET były wyborem dla rozwoju szybki i tani". — Zachary Olsen, architekt

Artykuły srebrne

Z pomocą więcej niż 15 lat doświadczenia w tworzeniu sprzedaży (POS) rozwiązania grup restauracji małych i średnich firm zespół projektowy w artykuły srebrne określone w celu ulepszenia swoich produktów z większą liczbą funkcji na poziomie przedsiębiorstwa w celu przyciągnięcia większej łańcuchy restauracji. Korzystając z najnowszej wersji narzędzi programistycznych firmy Microsoft, byli w stanie tworzyć nowe rozwiązanie cztery razy szybciej niż wcześniej. Najważniejsze nowe funkcje, takie jak LINQ i Entity Framework łatwiejsze przenoszenie z programu Crystal Reports do programu SQL Server 2008 i SQL Server Reporting Services (SSRS) do przechowywania ich danych i raportowaniem.

"Zarządzanie danymi efektywne ma kluczowe znaczenie dla powodzenia artykuły srebrne — i dlatego podjęliśmy decyzję o przyjęcie, SQL Reporting". -Nicholas Romanidis, Dyrektor ds. IT / inżynierii oprogramowania

Przyczynia się do programu Entity Framework 6

13.09.2018 • 2 minutes to read • [Edit Online](#)

Entity Framework 6 jest opracowany przy użyciu modelu typu "open source" w witrynie GitHub. Mimo, że główny zespół Entity Framework w firmie Microsoft koncentruje się na temat dodawania nowych funkcji do platformy Entity Framework Core, a nie oczekujemy, że wszystkie główne funkcje, które mają zostać dodane do platformy Entity Framework 6, firma Microsoft nadal akceptować wkładów.

Wkład produktu, Rozpocznij od [Dodawanie strony typu wiki w naszym repozytorium GitHub](#).

Dotyczące współtworzenia dokumentacji, uruchom odczytu [wskazówki wkład](#) w naszym repozytorium dokumentacji.

Uzyskaj pomoc przy użyciu platformy Entity Framework

13.09.2018 • 2 minutes to read • [Edit Online](#)



Pytania dotyczące korzystania z programów EF

Najlepszym sposobem, aby uzyskać pomoc przy użyciu platformy Entity Framework jest [zadać pytanie w witrynie Stack Overflow](#) przy użyciu **platformy entity framework** tagu.

Jeśli nie jesteś zaznajomiony z witryny Stack Overflow, pamiętaj, aby [zapoznać się ze wskazówkami na zadawanie pytań](#). W szczególności nie umożliwia Stack Overflow zgłoszenie usterki, zadawać pytania plan lub nowych funkcji.



Żądania funkcji i raportów o błędach

Jeśli znaleziono usterkę, która Twoim zdaniem powinny zostać naprawione, ma funkcji, które chcesz zobaczyć zaimplementowane lub pytanie nie można znaleźć odpowiedzi, Utwórz problem w [repozytorium GitHub platformy EF6](#).

Entity Framework słownik

10.10.2018 • 5 minutes to read • [Edit Online](#)

Najpierw kod

Tworzenie modelu Entity Framework przy użyciu kodu. Można wskazać modelu i istniejącej lub nowej bazy danych.

Kontekst

Klasa, która reprezentuje sesję z bazą danych, dzięki czemu możesz do zapytania i Zapisz dane. Kontekst jest pochodną klasy DbContext lub obiektu ObjectContext.

Konwencja (kod: pierwszej)

Reguła, która korzysta z programu Entity Framework wywnioskowania kształtu modelu z klas.

Najpierw bazy danych

Tworzenie modelu Entity Framework, za pomocą projektanta EF, który wspiera istniejącej bazy danych.

Wczesne ładowanie

Wzorzec załadunku, powiązanych danych, gdzie zapytanie dla jednego typu obiektu również ładuje powiązanych jednostek jako część zapytania.

Projektancie platformy EF

Projektant wizualny w programie Visual Studio umożliwia utworzenie modelu Entity Framework, używając okna i wierszy.

Jednostka

Klasa lub obiekt, który reprezentuje dane aplikacji, takich jak klientów, produkty i zamówienia.

Entity Data Model

Model, który opisuje jednostek i relacji między nimi. EF używa EDM do opisu modelu koncepcyjnego, względem którego programy dla deweloperów. EDM opiera się na modelu Relacja jednostki, wynikające z odzyskiwania po awarii. Peter Chen. Podstawowym celem staje się wspólnego modelu danych na zestaw technologii firmy Microsoft dla deweloperów i serwer został pierwotnie opracowana EDM. EDM jest również używane jako część protokołu OData.

jawne ładowanie

Wzorzec załadunku, powiązanych danych, gdzie obiekty powiązane są ładowane przez wywołanie interfejsu API.

Interfejs Fluent API

Interfejs API, który może służyć do konfigurowania modelu Code First.

Skojarzenie klucza obcego

Skojarzenia między jednostkami, której właściwość, która reprezentuje klucz obcy znajduje się w klasie jednostki zależne. Na przykład produkt zawiera właściwość CategoryId.

Identyfikowanie relacji

Relacja, w którym klucz podstawowy jednostki głównej jest częścią klucza podstawowego jednostki zależne. W tego rodzaju relacji jednostki zależne nie może istnieć bez jednostki głównej.

Niezależnie od skojarzenia

Skojarzenia między jednostkami w przypadku, gdy nie ma właściwości reprezentującej klucz obcy w klasie jednostki zależne. Na przykład klasa produktu zawiera relację z kategorii, ale nie ma właściwości CategoryId. Entity Framework śledzi stan skojarzenia, niezależnie od stanu jednostek końców dwóch skojarzenia.

Ładowanie z opóźnieniem

Wzorzec załadunku, powiązanych danych, gdzie obiekty powiązane są ładowane automatycznie podczas uzyskiwania dostępu do właściwości nawigacji.

Najpierw modelu

Tworzenie modelu Entity Framework, za pomocą projektanta EF następnie używany do tworzenia nowej bazy danych.

Właściwość nawigacji

Właściwość obiektu, który odwołuje się do innej jednostki. Na przykład produkt zawiera właściwość nawigacji kategorii i kategoria zawiera właściwość nawigacji produktów.

OBIEKTÓW POCO

Akronim obiektu CLR zwykły stary. Klasa prostych użytkownika, która nie ma zależności za pomocą dowolnej platformy. W kontekście EF, klasa jednostki, która nie pochodzi od EntityObject, implementuje żadnych interfejsów lub niesie ze sobą wszelkie atrybuty zdefiniowane w programie EF. Takie klasy jednostek, które są całkowicie niezależni od framework trwałości są również określane jako "trwałość zakresu".

Odwrotne relacji

Przeciwnym końcem relacji, na przykład produktu. Kategoria i kategorii. Produkt.

Samodzielne śledzenia jednostki

Jednostka utworzona na podstawie szablonu generowania kodu, który pomaga w rozwoju N-warstwowej.

Typ tabeli na konkretny (TPC)

Metoda mapowania dziedziczenia, gdzie każdy typ nieabstrakcyjnej w hierarchii jest mapowany do oddzielnych tabel w bazie danych.

Tabela wg hierarchii (TPH)

Metoda mapowania dziedziczenia, w którym wszystkie typy w hierarchii są mapowane na tej samej tabeli w bazie

danych. Dyskryminator kolumny jest używana do identyfikowania, jaki typ każdy wiersz jest skojarzony.

Tabela wg typu (TPT)

Metoda mapowania dziedziczenia, gdzie wspólne właściwości wszystkich typów w hierarchii są mapowane na tej samej tabeli w bazie danych, ale unikatowe dla każdego typu właściwości są mapowane na osobnej tabeli.

Typ odnajdywania

Proces identyfikacji typów, które powinny być częścią modelu Entity Framework.

Przykładowej bazy danych School

13.09.2018 • 15 minutes to read • [Edit Online](#)

Ten temat zawiera schemat i dane do bazy danych School. Przykładowa baza danych School jest używana w różnych miejscach w dokumentacji programu Entity Framework.

NOTE

Serwer bazy danych, który został zainstalowany przy użyciu programu Visual Studio różni się zależnie od wersji programu Visual Studio, możesz użyć. Zobacz [wersji usługi Visual Studio](#) szczegółowe informacje na temat rozwiązania do zastosowania.

Poniżej przedstawiono kroki umożliwiające utworzenie bazy danych:

- Otwórz program Visual Studio
- **Widok -> Eksploratora serwera**
- Kliknij prawym przyciskiem myszy **połączeń danych** -> **Dodaj połączenie...**
- Jeśli nie jest połączona z bazą danych za pomocą Eksploratora serwera, zanim będzie konieczne wybranie **programu Microsoft SQL Server** jako źródło danych
- Łączenie się z LocalDB lub SQL Express, w zależności od tego, który z nich został zainstalowany
- Wprowadź **School** jako nazwa bazy danych
- Wybierz **OK** i uzyskasz, jeśli chcesz utworzyć nową bazę danych, wybierz opcję **tak**
- Nowa baza danych będzie teraz wyświetlany w Eksploratorze serwera
- Jeśli używasz programu Visual Studio 2012 lub nowszy
 - Kliknij prawym przyciskiem myszy w bazie danych w Eksploratorze serwera i wybierz **nowe zapytanie**
 - Skopiuj następujące instrukcje SQL do nowego zapytania, a następnie kliknij prawym przyciskiem myszy, zapytania i wybierz pozycję **wykonania**
- Jeśli używasz programu Visual Studio 2010
 - Wybierz **danych -> język Transact SQL edytora -> nowego połączenia zapytania...**
 - Wprowadź **.\\SQLEXPRESS** jako nazwę serwera i kliknij przycisk **OK**
 - Wybierz **STESample** bazy danych z listy rozwijanej w górnej części edytora zapytań
 - Skopiuj następujące instrukcje SQL do nowego zapytania, a następnie kliknij prawym przyciskiem myszy, zapytania i wybierz pozycję **wykonaj instrukcję SQL**

```
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO

-- Create the Department table.
IF NOT EXISTS (SELECT * FROM sys.objects
WHERE object_id = OBJECT_ID(N'[dbo].[Department]')
AND type in (N'U'))
BEGIN
CREATE TABLE [dbo].[Department]([DepartmentID] [int] NOT NULL,
[Name] [nvarchar](50) NOT NULL,
[Budget] [money] NOT NULL,
[StartDate] [datetime] NOT NULL,
[Administrator] [int] NULL,
CONSTRAINT [PK_Department] PRIMARY KEY CLUSTERED
```

```

(
[DepartmentID] ASC
)WITH (IGNORE_DUP_KEY = OFF) ON [PRIMARY]) ON [PRIMARY]
END
GO

-- Create the Person table.
IF NOT EXISTS (SELECT * FROM sys.objects
WHERE object_id = OBJECT_ID(N'[dbo].[Person]')
AND type in (N'U'))
BEGIN
CREATE TABLE [dbo].[Person]([PersonID] [int] IDENTITY(1,1) NOT NULL,
[LastName] [nvarchar](50) NOT NULL,
[FirstName] [nvarchar](50) NOT NULL,
[HireDate] [datetime] NULL,
[EnrollmentDate] [datetime] NULL,
[Discriminator] [nvarchar](50) NOT NULL,
CONSTRAINT [PK_School.Student] PRIMARY KEY CLUSTERED
(
[PersonID] ASC
)WITH (IGNORE_DUP_KEY = OFF) ON [PRIMARY]) ON [PRIMARY]
END
GO

-- Create the OnsiteCourse table.
IF NOT EXISTS (SELECT * FROM sys.objects
WHERE object_id = OBJECT_ID(N'[dbo].[OnsiteCourse]')
AND type in (N'U'))
BEGIN
CREATE TABLE [dbo].[OnsiteCourse]([CourseID] [int] NOT NULL,
[Location] [nvarchar](50) NOT NULL,
[Days] [nvarchar](50) NOT NULL,
[Time] [smalldatetime] NOT NULL,
CONSTRAINT [PK_OnsiteCourse] PRIMARY KEY CLUSTERED
(
[CourseID] ASC
)WITH (IGNORE_DUP_KEY = OFF) ON [PRIMARY]) ON [PRIMARY]
END
GO

-- Create the OnlineCourse table.
IF NOT EXISTS (SELECT * FROM sys.objects
WHERE object_id = OBJECT_ID(N'[dbo].[OnlineCourse]')
AND type in (N'U'))
BEGIN
CREATE TABLE [dbo].[OnlineCourse]([CourseID] [int] NOT NULL,
[URL] [nvarchar](100) NOT NULL,
CONSTRAINT [PK_OnlineCourse] PRIMARY KEY CLUSTERED
(
[CourseID] ASC
)WITH (IGNORE_DUP_KEY = OFF) ON [PRIMARY]) ON [PRIMARY]
END
GO

--Create the StudentGrade table.
IF NOT EXISTS (SELECT * FROM sys.objects
WHERE object_id = OBJECT_ID(N'[dbo].[StudentGrade]')
AND type in (N'U'))
BEGIN
CREATE TABLE [dbo].[StudentGrade]([EnrollmentID] [int] IDENTITY(1,1) NOT NULL,
[CourseID] [int] NOT NULL,
[StudentID] [int] NOT NULL,
[Grade] [decimal](3, 2) NULL,
CONSTRAINT [PK_StudentGrade] PRIMARY KEY CLUSTERED
(
[EnrollmentID] ASC
)WITH (IGNORE_DUP_KEY = OFF) ON [PRIMARY]) ON [PRIMARY]
END
GO

```

```

-- Create the CourseInstructor table.
IF NOT EXISTS (SELECT * FROM sys.objects
WHERE object_id = OBJECT_ID(N'[dbo].[CourseInstructor]')
AND type in (N'U'))
BEGIN
CREATE TABLE [dbo].[CourseInstructor]([CourseID] [int] NOT NULL,
[PersonID] [int] NOT NULL,
CONSTRAINT [PK_CourseInstructor] PRIMARY KEY CLUSTERED
(
[CourseID] ASC,
[PersonID] ASC
)WITH (IGNORE_DUP_KEY = OFF) ON [PRIMARY]) ON [PRIMARY]
END
GO

-- Create the Course table.
IF NOT EXISTS (SELECT * FROM sys.objects
WHERE object_id = OBJECT_ID(N'[dbo].[Course]')
AND type in (N'U'))
BEGIN
CREATE TABLE [dbo].[Course]([CourseID] [int] NOT NULL,
[Title] [nvarchar](100) NOT NULL,
[Credits] [int] NOT NULL,
[DepartmentID] [int] NOT NULL,
CONSTRAINT [PK_School.Course] PRIMARY KEY CLUSTERED
(
[CourseID] ASC
)WITH (IGNORE_DUP_KEY = OFF) ON [PRIMARY]) ON [PRIMARY]
END
GO

-- Create the OfficeAssignment table.
IF NOT EXISTS (SELECT * FROM sys.objects
WHERE object_id = OBJECT_ID(N'[dbo].[OfficeAssignment]')
AND type in (N'U'))
BEGIN
CREATE TABLE [dbo].[OfficeAssignment]([InstructorID] [int] NOT NULL,
[Location] [nvarchar](50) NOT NULL,
[Timestamp] [timestamp] NOT NULL,
CONSTRAINT [PK_OfficeAssignment] PRIMARY KEY CLUSTERED
(
[InstructorID] ASC
)WITH (IGNORE_DUP_KEY = OFF) ON [PRIMARY]) ON [PRIMARY]
END
GO

-- Define the relationship between OnsiteCourse and Course.
IF NOT EXISTS (SELECT * FROM sys.foreign_keys
WHERE object_id = OBJECT_ID(N'[dbo].[FK_OnsiteCourse_Course]')
AND parent_object_id = OBJECT_ID(N'[dbo].[OnsiteCourse]'))
ALTER TABLE [dbo].[OnsiteCourse] WITH CHECK ADD
CONSTRAINT [FK_OnsiteCourse_Course] FOREIGN KEY([CourseID])
REFERENCES [dbo].[Course] ([CourseID])
GO
ALTER TABLE [dbo].[OnsiteCourse] CHECK
CONSTRAINT [FK_OnsiteCourse_Course]
GO

-- Define the relationship between OnlineCourse and Course.
IF NOT EXISTS (SELECT * FROM sys.foreign_keys
WHERE object_id = OBJECT_ID(N'[dbo].[FK_OnlineCourse_Course]')
AND parent_object_id = OBJECT_ID(N'[dbo].[OnlineCourse]'))
ALTER TABLE [dbo].[OnlineCourse] WITH CHECK ADD
CONSTRAINT [FK_OnlineCourse_Course] FOREIGN KEY([CourseID])
REFERENCES [dbo].[Course] ([CourseID])
GO
ALTER TABLE [dbo].[OnlineCourse] CHECK
CONSTRAINT [FK_OnlineCourse_Course]

```

```

CONSTRAINT [FK_OnlineCourse_Course]
GO

-- Define the relationship between StudentGrade and Course.
IF NOT EXISTS (SELECT * FROM sys.foreign_keys
WHERE object_id = OBJECT_ID(N'[dbo].[FK_StudentGrade_Course]')
AND parent_object_id = OBJECT_ID(N'[dbo].[StudentGrade]'))
ALTER TABLE [dbo].[StudentGrade] WITH CHECK ADD
CONSTRAINT [FK_StudentGrade_Course] FOREIGN KEY([CourseID])
REFERENCES [dbo].[Course] ([CourseID])
GO

ALTER TABLE [dbo].[StudentGrade] CHECK
CONSTRAINT [FK_StudentGrade_Course]
GO

--Define the relationship between StudentGrade and Student.
IF NOT EXISTS (SELECT * FROM sys.foreign_keys
WHERE object_id = OBJECT_ID(N'[dbo].[FK_StudentGrade_Student]')
AND parent_object_id = OBJECT_ID(N'[dbo].[StudentGrade]'))
ALTER TABLE [dbo].[StudentGrade] WITH CHECK ADD
CONSTRAINT [FK_StudentGrade_Student] FOREIGN KEY([StudentID])
REFERENCES [dbo].[Person] ([PersonID])
GO

ALTER TABLE [dbo].[StudentGrade] CHECK
CONSTRAINT [FK_StudentGrade_Student]
GO

-- Define the relationship between CourseInstructor and Course.
IF NOT EXISTS (SELECT * FROM sys.foreign_keys
WHERE object_id = OBJECT_ID(N'[dbo].[FK_CourseInstructor_Course]')
AND parent_object_id = OBJECT_ID(N'[dbo].[CourseInstructor]'))
ALTER TABLE [dbo].[CourseInstructor] WITH CHECK ADD
CONSTRAINT [FK_CourseInstructor_Course] FOREIGN KEY([CourseID])
REFERENCES [dbo].[Course] ([CourseID])
GO

ALTER TABLE [dbo].[CourseInstructor] CHECK
CONSTRAINT [FK_CourseInstructor_Course]
GO

-- Define the relationship between CourseInstructor and Person.
IF NOT EXISTS (SELECT * FROM sys.foreign_keys
WHERE object_id = OBJECT_ID(N'[dbo].[FK_CourseInstructor_Person]')
AND parent_object_id = OBJECT_ID(N'[dbo].[CourseInstructor]'))
ALTER TABLE [dbo].[CourseInstructor] WITH CHECK ADD
CONSTRAINT [FK_CourseInstructor_Person] FOREIGN KEY([PersonID])
REFERENCES [dbo].[Person] ([PersonID])
GO

ALTER TABLE [dbo].[CourseInstructor] CHECK
CONSTRAINT [FK_CourseInstructor_Person]
GO

-- Define the relationship between Course and Department.
IF NOT EXISTS (SELECT * FROM sys.foreign_keys
WHERE object_id = OBJECT_ID(N'[dbo].[FK_Course_Department]')
AND parent_object_id = OBJECT_ID(N'[dbo].[Course]'))
ALTER TABLE [dbo].[Course] WITH CHECK ADD
CONSTRAINT [FK_Course_Department] FOREIGN KEY([DepartmentID])
REFERENCES [dbo].[Department] ([DepartmentID])
GO

ALTER TABLE [dbo].[Course] CHECK CONSTRAINT [FK_Course_Department]
GO

--Define the relationship between OfficeAssignment and Person.
IF NOT EXISTS (SELECT * FROM sys.foreign_keys
WHERE object_id = OBJECT_ID(N'[dbo].[FK_OfficeAssignment_Person]')
AND parent_object_id = OBJECT_ID(N'[dbo].[OfficeAssignment]'))
ALTER TABLE [dbo].[OfficeAssignment] WITH CHECK ADD
CONSTRAINT [FK_OfficeAssignment_Person] FOREIGN KEY([InstructorID])
REFERENCES [dbo].[Person] ([PersonID])
--
```

```

GO
ALTER TABLE [dbo].[OfficeAssignment] CHECK
CONSTRAINT [FK_OfficeAssignment_Person]
GO

-- Create InsertOfficeAssignment stored procedure.
IF NOT EXISTS (SELECT * FROM sys.objects
WHERE object_id = OBJECT_ID(N'[dbo].[InsertOfficeAssignment]')
AND type in (N'P', N'PC'))
BEGIN
EXEC dbo.sp_executesql @statement = N'
CREATE PROCEDURE [dbo].[InsertOfficeAssignment]
@InstructorID int,
@Location nvarchar(50)
AS
INSERT INTO dbo.OfficeAssignment (InstructorID, Location)
VALUES (@InstructorID, @Location);
IF @@ROWCOUNT > 0
BEGIN
SELECT [Timestamp] FROM OfficeAssignment
WHERE InstructorID=@InstructorID;
END
'
END
GO

--Create the UpdateOfficeAssignment stored procedure.
IF NOT EXISTS (SELECT * FROM sys.objects
WHERE object_id = OBJECT_ID(N'[dbo].[UpdateOfficeAssignment]')
AND type in (N'P', N'PC'))
BEGIN
EXEC dbo.sp_executesql @statement = N'
CREATE PROCEDURE [dbo].[UpdateOfficeAssignment]
@InstructorID int,
@Location nvarchar(50),
@OrigTimestamp timestamp
AS
UPDATE OfficeAssignment SET Location=@Location
WHERE InstructorID=@InstructorID AND [Timestamp]=@OrigTimestamp;
IF @@ROWCOUNT > 0
BEGIN
SELECT [Timestamp] FROM OfficeAssignment
WHERE InstructorID=@InstructorID;
END
'
END
GO

-- Create the DeleteOfficeAssignment stored procedure.
IF NOT EXISTS (SELECT * FROM sys.objects
WHERE object_id = OBJECT_ID(N'[dbo].[DeleteOfficeAssignment]')
AND type in (N'P', N'PC'))
BEGIN
EXEC dbo.sp_executesql @statement = N'
CREATE PROCEDURE [dbo].[DeleteOfficeAssignment]
@InstructorID int
AS
DELETE FROM OfficeAssignment
WHERE InstructorID=@InstructorID;
'
END
GO

-- Create the DeletePerson stored procedure.
IF NOT EXISTS (SELECT * FROM sys.objects
WHERE object_id = OBJECT_ID(N'[dbo].[DeletePerson]')
AND type in (N'P', N'PC'))
BEGIN
EXEC dbo.sp_executesql @statement = N'

```

```

CREATE PROCEDURE [dbo].[DeletePerson]
@PersonID int
AS
DELETE FROM Person WHERE PersonID = @PersonID;
'

END
GO

-- Create the UpdatePerson stored procedure.
IF NOT EXISTS (SELECT * FROM sys.objects
WHERE object_id = OBJECT_ID(N'[dbo].[UpdatePerson]')
AND type in (N'P', N'PC'))
BEGIN
EXEC dbo.sp_executesql @statement = N'
CREATE PROCEDURE [dbo].[UpdatePerson]
@PersonID int,
@LastName nvarchar(50),
@FirstName nvarchar(50),
@HireDate datetime,
@EnrollmentDate datetime,
@Discriminator nvarchar(50)
AS
UPDATE Person SET LastName=@LastName,
FirstName=@FirstName,
HireDate=@HireDate,
EnrollmentDate=@EnrollmentDate,
Discriminator=@Discriminator
WHERE PersonID=@PersonID;
'

END
GO

-- Create the InsertPerson stored procedure.
IF NOT EXISTS (SELECT * FROM sys.objects
WHERE object_id = OBJECT_ID(N'[dbo].[InsertPerson]')
AND type in (N'P', N'PC'))
BEGIN
EXEC dbo.sp_executesql @statement = N'
CREATE PROCEDURE [dbo].[InsertPerson]
@LastName nvarchar(50),
@FirstName nvarchar(50),
@HireDate datetime,
@EnrollmentDate datetime,
@Discriminator nvarchar(50)
AS
INSERT INTO dbo.Person (LastName,
FirstName,
HireDate,
EnrollmentDate,
Discriminator)
VALUES (@LastName,
@FirstName,
@HireDate,
@EnrollmentDate,
@Discriminator);
SELECT SCOPE_IDENTITY() as NewPersonID;
'

END
GO

-- Create GetStudentGrades stored procedure.
IF NOT EXISTS (SELECT * FROM sys.objects
WHERE object_id = OBJECT_ID(N'[dbo].[GetStudentGrades]')
AND type in (N'P', N'PC'))
BEGIN
EXEC dbo.sp_executesql @statement = N'
CREATE PROCEDURE [dbo].[GetStudentGrades]
@studentID int
AS

```

```

SELECT EnrollmentID, Grade, CourseID, StudentID FROM dbo.StudentGrade
WHERE StudentID = @StudentID
'

END
GO

-- Create GetDepartmentName stored procedure.
IF NOT EXISTS (SELECT * FROM sys.objects
WHERE object_id = OBJECT_ID(N'[dbo].[GetDepartmentName]')
AND type in (N'P', N'PC'))
BEGIN
EXEC dbo.sp_executesql @statement = N'
CREATE PROCEDURE [dbo].[GetDepartmentName]
@ID int,
@Name nvarchar(50) OUTPUT
AS
SELECT @Name = Name FROM Department
WHERE DepartmentID = @ID
'
END
GO

-- Insert data into the Person table.
USE School
GO
SET IDENTITY_INSERT dbo.Person ON
GO
INSERT INTO dbo.Person (PersonID, LastName, FirstName, HireDate, EnrollmentDate, Discriminator)
VALUES (1, 'Abercrombie', 'Kim', '1995-03-11', null, 'Instructor');
INSERT INTO dbo.Person (PersonID, LastName, FirstName, HireDate, EnrollmentDate, Discriminator)
VALUES (2, 'Barzdukas', 'Gytis', null, '2005-09-01', 'Student');
INSERT INTO dbo.Person (PersonID, LastName, FirstName, HireDate, EnrollmentDate, Discriminator)
VALUES (3, 'Justice', 'Peggy', null, '2001-09-01', 'Student');
INSERT INTO dbo.Person (PersonID, LastName, FirstName, HireDate, EnrollmentDate, Discriminator)
VALUES (4, 'Fakhouri', 'Fadi', '2002-08-06', null, 'Instructor');
INSERT INTO dbo.Person (PersonID, LastName, FirstName, HireDate, EnrollmentDate, Discriminator)
VALUES (5, 'Harui', 'Roger', '1998-07-01', null, 'Instructor');
INSERT INTO dbo.Person (PersonID, LastName, FirstName, HireDate, EnrollmentDate, Discriminator)
VALUES (6, 'Li', 'Yan', null, '2002-09-01', 'Student');
INSERT INTO dbo.Person (PersonID, LastName, FirstName, HireDate, EnrollmentDate, Discriminator)
VALUES (7, 'Norman', 'Laura', null, '2003-09-01', 'Student');
INSERT INTO dbo.Person (PersonID, LastName, FirstName, HireDate, EnrollmentDate, Discriminator)
VALUES (8, 'Olivotto', 'Nino', null, '2005-09-01', 'Student');
INSERT INTO dbo.Person (PersonID, LastName, FirstName, HireDate, EnrollmentDate, Discriminator)
VALUES (9, 'Tang', 'Wayne', null, '2005-09-01', 'Student');
INSERT INTO dbo.Person (PersonID, LastName, FirstName, HireDate, EnrollmentDate, Discriminator)
VALUES (10, 'Alonso', 'Meredith', null, '2002-09-01', 'Student');
INSERT INTO dbo.Person (PersonID, LastName, FirstName, HireDate, EnrollmentDate, Discriminator)
VALUES (11, 'Lopez', 'Sophia', null, '2004-09-01', 'Student');
INSERT INTO dbo.Person (PersonID, LastName, FirstName, HireDate, EnrollmentDate, Discriminator)
VALUES (12, 'Browning', 'Meredith', null, '2000-09-01', 'Student');
INSERT INTO dbo.Person (PersonID, LastName, FirstName, HireDate, EnrollmentDate, Discriminator)
VALUES (13, 'Anand', 'Arturo', null, '2003-09-01', 'Student');
INSERT INTO dbo.Person (PersonID, LastName, FirstName, HireDate, EnrollmentDate, Discriminator)
VALUES (14, 'Walker', 'Alexandra', null, '2000-09-01', 'Student');
INSERT INTO dbo.Person (PersonID, LastName, FirstName, HireDate, EnrollmentDate, Discriminator)
VALUES (15, 'Powell', 'Carson', null, '2004-09-01', 'Student');
INSERT INTO dbo.Person (PersonID, LastName, FirstName, HireDate, EnrollmentDate, Discriminator)
VALUES (16, 'Jai', 'Damien', null, '2001-09-01', 'Student');
INSERT INTO dbo.Person (PersonID, LastName, FirstName, HireDate, EnrollmentDate, Discriminator)
VALUES (17, 'Carlson', 'Robyn', null, '2005-09-01', 'Student');
INSERT INTO dbo.Person (PersonID, LastName, FirstName, HireDate, EnrollmentDate, Discriminator)
VALUES (18, 'Zheng', 'Roger', '2004-02-12', null, 'Instructor');
INSERT INTO dbo.Person (PersonID, LastName, FirstName, HireDate, EnrollmentDate, Discriminator)
VALUES (19, 'Bryant', 'Carson', null, '2001-09-01', 'Student');
INSERT INTO dbo.Person (PersonID, LastName, FirstName, HireDate, EnrollmentDate, Discriminator)
VALUES (20, 'Suarez', 'Robyn', null, '2004-09-01', 'Student');
INSERT INTO dbo.Person (PersonID, LastName, FirstName, HireDate, EnrollmentDate, Discriminator)
VALUES (21, 'Holt', 'Roger', null, '2004-09-01', 'Student');

```

```
INSERT INTO dbo.Person (PersonID, LastName, FirstName, HireDate, EnrollmentDate, Discriminator)
VALUES (22, 'Alexander', 'Carson', null, '2005-09-01', 'Student');
INSERT INTO dbo.Person (PersonID, LastName, FirstName, HireDate, EnrollmentDate, Discriminator)
VALUES (23, 'Morgan', 'Isaiah', null, '2001-09-01', 'Student');
INSERT INTO dbo.Person (PersonID, LastName, FirstName, HireDate, EnrollmentDate, Discriminator)
VALUES (24, 'Martin', 'Randall', null, '2005-09-01', 'Student');
INSERT INTO dbo.Person (PersonID, LastName, FirstName, HireDate, EnrollmentDate, Discriminator)
VALUES (25, 'Kapoor', 'Candace', '2001-01-15', null, 'Instructor');
INSERT INTO dbo.Person (PersonID, LastName, FirstName, HireDate, EnrollmentDate, Discriminator)
VALUES (26, 'Rogers', 'Cody', null, '2002-09-01', 'Student');
INSERT INTO dbo.Person (PersonID, LastName, FirstName, HireDate, EnrollmentDate, Discriminator)
VALUES (27, 'Serrano', 'Stacy', '1999-06-01', null, 'Instructor');
INSERT INTO dbo.Person (PersonID, LastName, FirstName, HireDate, EnrollmentDate, Discriminator)
VALUES (28, 'White', 'Anthony', null, '2001-09-01', 'Student');
INSERT INTO dbo.Person (PersonID, LastName, FirstName, HireDate, EnrollmentDate, Discriminator)
VALUES (29, 'Griffin', 'Rachel', null, '2004-09-01', 'Student');
INSERT INTO dbo.Person (PersonID, LastName, FirstName, HireDate, EnrollmentDate, Discriminator)
VALUES (30, 'Shan', 'Alicia', null, '2003-09-01', 'Student');
INSERT INTO dbo.Person (PersonID, LastName, FirstName, HireDate, EnrollmentDate, Discriminator)
VALUES (31, 'Stewart', 'Jasmine', '1997-10-12', null, 'Instructor');
INSERT INTO dbo.Person (PersonID, LastName, FirstName, HireDate, EnrollmentDate, Discriminator)
VALUES (32, 'Xu', 'Kristen', '2001-7-23', null, 'Instructor');
INSERT INTO dbo.Person (PersonID, LastName, FirstName, HireDate, EnrollmentDate, Discriminator)
VALUES (33, 'Gao', 'Erica', null, '2003-01-30', 'Student');
INSERT INTO dbo.Person (PersonID, LastName, FirstName, HireDate, EnrollmentDate, Discriminator)
VALUES (34, 'Van Houten', 'Roger', '2000-12-07', null, 'Instructor');
GO
SET IDENTITY_INSERT dbo.Person OFF
GO
```

```
-- Insert data into the Department table.
INSERT INTO dbo.Department (DepartmentID, [Name], Budget, StartDate, Administrator)
VALUES (1, 'Engineering', 350000.00, '2007-09-01', 2);
INSERT INTO dbo.Department (DepartmentID, [Name], Budget, StartDate, Administrator)
VALUES (2, 'English', 120000.00, '2007-09-01', 6);
INSERT INTO dbo.Department (DepartmentID, [Name], Budget, StartDate, Administrator)
VALUES (4, 'Economics', 200000.00, '2007-09-01', 4);
INSERT INTO dbo.Department (DepartmentID, [Name], Budget, StartDate, Administrator)
VALUES (7, 'Mathematics', 250000.00, '2007-09-01', 3);
GO
```

```
-- Insert data into the Course table.
INSERT INTO dbo.Course (CourseID, Title, Credits, DepartmentID)
VALUES (1050, 'Chemistry', 4, 1);
INSERT INTO dbo.Course (CourseID, Title, Credits, DepartmentID)
VALUES (1061, 'Physics', 4, 1);
INSERT INTO dbo.Course (CourseID, Title, Credits, DepartmentID)
VALUES (1045, 'Calculus', 4, 7);
INSERT INTO dbo.Course (CourseID, Title, Credits, DepartmentID)
VALUES (2030, 'Poetry', 2, 2);
INSERT INTO dbo.Course (CourseID, Title, Credits, DepartmentID)
VALUES (2021, 'Composition', 3, 2);
INSERT INTO dbo.Course (CourseID, Title, Credits, DepartmentID)
VALUES (2042, 'Literature', 4, 2);
INSERT INTO dbo.Course (CourseID, Title, Credits, DepartmentID)
VALUES (4022, 'Microeconomics', 3, 4);
INSERT INTO dbo.Course (CourseID, Title, Credits, DepartmentID)
VALUES (4041, 'Macroeconomics', 3, 4);
INSERT INTO dbo.Course (CourseID, Title, Credits, DepartmentID)
VALUES (4061, 'Quantitative', 2, 4);
INSERT INTO dbo.Course (CourseID, Title, Credits, DepartmentID)
VALUES (3141, 'Trigonometry', 4, 7);
GO
```

```
-- Insert data into the OnlineCourse table.
INSERT INTO dbo.OnlineCourse (CourseID, URL)
VALUES (2030, 'http://www.fineartschool.net/Poetry');
```

```
-- Insert data into the OnlineCourse table.
INSERT INTO dbo.OnlineCourse (CourseID, URL)
VALUES (2021, 'http://www.fineartschool.net/Composition');
INSERT INTO dbo.OnlineCourse (CourseID, URL)
VALUES (4041, 'http://www.fineartschool.net/Macroeconomics');
INSERT INTO dbo.OnlineCourse (CourseID, URL)
VALUES (3141, 'http://www.fineartschool.net/Trigonometry');

--Insert data into OnsiteCourse table.
INSERT INTO dbo.OnsiteCourse (CourseID, Location, Days, [Time])
VALUES (1050, '123 Smith', 'MTWH', '11:30');
INSERT INTO dbo.OnsiteCourse (CourseID, Location, Days, [Time])
VALUES (1061, '234 Smith', 'TWHF', '13:15');
INSERT INTO dbo.OnsiteCourse (CourseID, Location, Days, [Time])
VALUES (1045, '121 Smith', 'MWHF', '15:30');
INSERT INTO dbo.OnsiteCourse (CourseID, Location, Days, [Time])
VALUES (4061, '22 Williams', 'TH', '11:15');
INSERT INTO dbo.OnsiteCourse (CourseID, Location, Days, [Time])
VALUES (2042, '225 Adams', 'MTWH', '11:00');
INSERT INTO dbo.OnsiteCourse (CourseID, Location, Days, [Time])
VALUES (4022, '23 Williams', 'MWF', '9:00');

-- Insert data into the CourseInstructor table.
INSERT INTO dbo.CourseInstructor(CourseID, PersonID)
VALUES (1050, 1);
INSERT INTO dbo.CourseInstructor(CourseID, PersonID)
VALUES (1061, 31);
INSERT INTO dbo.CourseInstructor(CourseID, PersonID)
VALUES (1045, 5);
INSERT INTO dbo.CourseInstructor(CourseID, PersonID)
VALUES (2030, 4);
INSERT INTO dbo.CourseInstructor(CourseID, PersonID)
VALUES (2021, 27);
INSERT INTO dbo.CourseInstructor(CourseID, PersonID)
VALUES (2042, 25);
INSERT INTO dbo.CourseInstructor(CourseID, PersonID)
VALUES (4022, 18);
INSERT INTO dbo.CourseInstructor(CourseID, PersonID)
VALUES (4041, 32);
INSERT INTO dbo.CourseInstructor(CourseID, PersonID)
VALUES (4061, 34);
GO

--Insert data into the OfficeAssignment table.
INSERT INTO dbo.OfficeAssignment(InstructorID, Location)
VALUES (1, '17 Smith');
INSERT INTO dbo.OfficeAssignment(InstructorID, Location)
VALUES (4, '29 Adams');
INSERT INTO dbo.OfficeAssignment(InstructorID, Location)
VALUES (5, '37 Williams');
INSERT INTO dbo.OfficeAssignment(InstructorID, Location)
VALUES (18, '143 Smith');
INSERT INTO dbo.OfficeAssignment(InstructorID, Location)
VALUES (25, '57 Adams');
INSERT INTO dbo.OfficeAssignment(InstructorID, Location)
VALUES (27, '271 Williams');
INSERT INTO dbo.OfficeAssignment(InstructorID, Location)
VALUES (31, '131 Smith');
INSERT INTO dbo.OfficeAssignment(InstructorID, Location)
VALUES (32, '203 Williams');
INSERT INTO dbo.OfficeAssignment(InstructorID, Location)
VALUES (34, '213 Smith');

-- Insert data into the StudentGrade table.
INSERT INTO dbo.StudentGrade (CourseID, StudentID, Grade)
VALUES (2021, 2, 4);
INSERT INTO dbo.StudentGrade (CourseID, StudentID, Grade)
VALUES (2030, 2, 3.5);
INSERT INTO dbo.StudentGrade (CourseID, StudentID, Grade)
VALUES (2021, 3, 3);
```

```
VALUES (2021, 1, 3);
INSERT INTO dbo.StudentGrade (CourseID, StudentID, Grade)
VALUES (2030, 3, 4);
INSERT INTO dbo.StudentGrade (CourseID, StudentID, Grade)
VALUES (2021, 6, 2.5);
INSERT INTO dbo.StudentGrade (CourseID, StudentID, Grade)
VALUES (2042, 6, 3.5);
INSERT INTO dbo.StudentGrade (CourseID, StudentID, Grade)
VALUES (2021, 7, 3.5);
INSERT INTO dbo.StudentGrade (CourseID, StudentID, Grade)
VALUES (2042, 7, 4);
INSERT INTO dbo.StudentGrade (CourseID, StudentID, Grade)
VALUES (2021, 8, 3);
INSERT INTO dbo.StudentGrade (CourseID, StudentID, Grade)
VALUES (2042, 8, 3);
INSERT INTO dbo.StudentGrade (CourseID, StudentID, Grade)
VALUES (4041, 9, 3.5);
INSERT INTO dbo.StudentGrade (CourseID, StudentID, Grade)
VALUES (4041, 10, null);
INSERT INTO dbo.StudentGrade (CourseID, StudentID, Grade)
VALUES (4041, 11, 2.5);
INSERT INTO dbo.StudentGrade (CourseID, StudentID, Grade)
VALUES (4041, 12, null);
INSERT INTO dbo.StudentGrade (CourseID, StudentID, Grade)
VALUES (4022, 14, 3);
INSERT INTO dbo.StudentGrade (CourseID, StudentID, Grade)
VALUES (4022, 13, 4);
INSERT INTO dbo.StudentGrade (CourseID, StudentID, Grade)
VALUES (4061, 13, 4);
INSERT INTO dbo.StudentGrade (CourseID, StudentID, Grade)
VALUES (4041, 14, 3);
INSERT INTO dbo.StudentGrade (CourseID, StudentID, Grade)
VALUES (4022, 15, 2.5);
INSERT INTO dbo.StudentGrade (CourseID, StudentID, Grade)
VALUES (4022, 16, 2);
INSERT INTO dbo.StudentGrade (CourseID, StudentID, Grade)
VALUES (4022, 17, null);
INSERT INTO dbo.StudentGrade (CourseID, StudentID, Grade)
VALUES (4022, 19, 3.5);
INSERT INTO dbo.StudentGrade (CourseID, StudentID, Grade)
VALUES (4061, 20, 4);
INSERT INTO dbo.StudentGrade (CourseID, StudentID, Grade)
VALUES (4061, 21, 2);
INSERT INTO dbo.StudentGrade (CourseID, StudentID, Grade)
VALUES (4022, 22, 3);
INSERT INTO dbo.StudentGrade (CourseID, StudentID, Grade)
VALUES (4041, 22, 3.5);
INSERT INTO dbo.StudentGrade (CourseID, StudentID, Grade)
VALUES (4061, 22, 2.5);
INSERT INTO dbo.StudentGrade (CourseID, StudentID, Grade)
VALUES (4022, 23, 3);
INSERT INTO dbo.StudentGrade (CourseID, StudentID, Grade)
VALUES (1045, 23, 1.5);
INSERT INTO dbo.StudentGrade (CourseID, StudentID, Grade)
VALUES (1061, 24, 4);
INSERT INTO dbo.StudentGrade (CourseID, StudentID, Grade)
VALUES (1061, 25, 3);
INSERT INTO dbo.StudentGrade (CourseID, StudentID, Grade)
VALUES (1050, 26, 3.5);
INSERT INTO dbo.StudentGrade (CourseID, StudentID, Grade)
VALUES (1061, 26, 3);
INSERT INTO dbo.StudentGrade (CourseID, StudentID, Grade)
VALUES (1061, 27, 3);
INSERT INTO dbo.StudentGrade (CourseID, StudentID, Grade)
VALUES (1045, 28, 2.5);
INSERT INTO dbo.StudentGrade (CourseID, StudentID, Grade)
VALUES (1050, 28, 3.5);
INSERT INTO dbo.StudentGrade (CourseID, StudentID, Grade)
VALUES (1050, 28, 3.5);
```

```
INSERT INTO dbo.StudentGrade (CourseID, StudentID, Grade)
VALUES (1061, 29, 4);
INSERT INTO dbo.StudentGrade (CourseID, StudentID, Grade)
VALUES (1050, 30, 3.5);
INSERT INTO dbo.StudentGrade (CourseID, StudentID, Grade)
VALUES (1061, 30, 4);
GO
```

Entity Framework Tools i rozszerzenia

13.09.2018 • 2 minutes to read • [Edit Online](#)

IMPORTANT

Rozszerzenia są skompilowanymi z różnych źródeł i nie są przechowywane jako część programu Entity Framework. Rozważając rozszerzenia innych firm, pamiętaj ocenić jakość, licencjonowanie, zgodności i pomocy technicznej, itp., aby upewnić się, że spełniają one wymagania.

Entity Framework została popularnych Obiektywnie przez wiele lat. Poniżej przedstawiono kilka przykładów, narzędzi bezpłatnych i płatnych i rozszerzeń opracowanych dla niego:

- [EF Power Tools Community Edition](#)
- [Profiler EF](#)
- [Profiler ORM](#)
- [LINQPad](#)
- [LLBLGen Pro](#)
- [Narzędzia Huagati DBML/EDMX](#)
- [Entity Developer](#)