[**https://docs.microsoft.com/it-it/ef/ef6/modeling/code-first/migrations/**](https://docs.microsoft.com/it-it/ef/ef6/modeling/code-first/migrations/)

**Migrazioni Code First**

* 23/10/2016
* 9 minuti per la lettura

Migrazioni Code First rappresenta il metodo più adatto per far evolvere lo schema del database dell'applicazione quando si usa il flusso di lavoro Code First. Le migrazioni offrono un set di strumenti che consentono di:

1. Creare un database iniziale che può essere usato con il modello di Entity Framework
2. Generare migrazioni per tenere traccia delle modifiche apportate al modello di Entity Framework
3. Mantenere aggiornato il database con le modifiche

Nella procedura dettagliata seguente viene fornita una panoramica delle Migrazioni Code First in Entity Framework. È possibile eseguire l'intera procedura dettagliata o passare all'argomento desiderato. Vengono trattati i seguenti argomenti:

**Creazione del modello iniziale e del database**

Per iniziare a usare le migrazioni sono necessari un progetto e un modello Code First. In questa procedura dettagliata verranno usati i modelli tradizionali **Blog** e **Post**.

* Creare una nuova applicazione console **MigrationsDemo**
* Aggiungere la versione più recente del pacchetto NuGet **EntityFramework** al progetto
  + **Strumenti –> Library Package Manager (Gestione pacchetti librerie) –> Console di Gestione pacchetti**
  + Eseguire il comando **Install-Package EntityFramework**
* Aggiungere un file **Model.cs** con il codice riportato di seguito. Questo codice definisce una singola classe **Blog** che rappresenta il modello di dominio e una classe **BlogContext** che rappresenta il contesto Code First di Entity Framework

C#Copia

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; }

}

}

* È ora possibile usare il modello per eseguire l'accesso ai dati. Aggiornare il file **Program.cs** con il codice riportato di seguito.

C#Copia

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...");

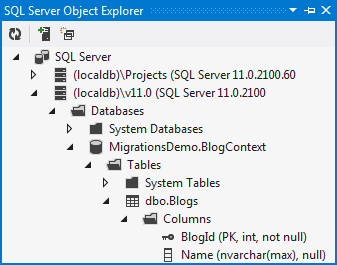
Console.ReadKey();

}

}

}

* Eseguire l'applicazione. Si noti che viene creato un database **MigrationsCodeDemo.BlogContext**.



**Abilitare le migrazioni**

È ora possibile apportare ulteriori modifiche al modello.

* Introdurre una proprietà Url nella classe Blog.

C#Copia

public string Url { get; set; }

Se si esegue di nuovo l'applicazione, viene generata un'eccezione InvalidOperationException con il messaggio *The model backing the 'BlogContext' context has changed since the database was created. Valutare la possibilità di usare migrazioni Code First per aggiornare il database (* [*http://go.microsoft.com/fwlink/?LinkId=238269*](https://go.microsoft.com/fwlink/?LinkId=238269)*).*

Come suggerisce l'eccezione, è ora possibile iniziare a usare Migrazioni Code First. Il primo passaggio consiste nell'abilitare le migrazioni per il contesto.

* Eseguire il comando **Enable-Migrations** nella console di Gestione pacchetti

Il comando aggiunge una cartella **Migrazioni** al progetto. La nuova cartella contiene due file:

* **La classe Configuration.** Questa classe consente di configurare il comportamento di Migrazioni per il contesto. In questa procedura dettagliata verrà usata solo la configurazione predefinita. *Poiché è presente un solo contesto Code First nel progetto, Enable-Migrations ha specificato automaticamente il tipo di contesto a cui si applica questa configurazione.*
* **Una migrazione InitialCreate**. Questa migrazione viene generata poiché è già stato creato automaticamente un database da Code First prima dell'abilitazione delle migrazioni. Il codice in questa migrazione di scaffolding rappresenta gli oggetti che sono già stati creati nel database. In questo caso, si tratta della tabella **Blog** con le colonne **BlogId** e **Name**. Il nome del file include un timestamp che facilita l'ordinamento. *Se il database non fosse già stato creato, questa migrazione InitialCreate non sarebbe stata aggiunta al progetto. La prima volta che viene chiamato Add-Migration, viene invece eseguito lo scaffolding del codice per la creazione delle tabelle in una nuova migrazione.*

**Più modelli con lo stesso database di destinazione**

Quando si usano versioni precedenti a EF6, è possibile usare solo un modello Code First per generare o gestire lo schema di un database. Quando è disponibile una sola tabella **\_\_MigrationsHistory** per ogni database, non è possibile identificare l'appartenenza delle voci ai modelli.

A partire da EF6, la classe **Configuration** include una proprietà **ContextKey**. La proprietà svolge la funzione di identificatore univoco per ogni modello Code First. Una colonna corrispondente nella tabella **\_\_MigrationsHistory** consente a voci di più modelli di condividere la tabella. Per impostazione predefinita, questa proprietà è impostata sul nome completo del contesto.

**Generazione ed esecuzione di migrazioni**

Migrazioni Code First include due comandi principali.

* **Add-Migration** esegue lo scaffolding della migrazione successiva in base alle modifiche apportate al modello dalla creazione dell'ultima migrazione
* **Update-Database** applica tutte le migrazioni in sospeso al database

Viene ora eseguito lo scaffolding di una migrazione per la nuova proprietà Url aggiunta. Il comando **Add-Migration** consente di assegnare un nome alle migrazioni, ovvero **AddBlogUrl**.

* Eseguire il comando **Add-Migration AddBlogUrl** nella console di Gestione pacchetti
* Nella cartella **Migrazioni** viene inserita la nuova migrazione **AddBlogUrl**. Il nome file della migrazione è preceduto da un timestamp per facilitare l'ordinamento

C#Copia

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");

}

}

}

È ora possibile apportare modifiche o aggiunte alla migrazione. Usare **Update-Database** per applicare la migrazione al database.

* Eseguire il comando **Update-Database** nella console di Gestione pacchetti
* Migrazioni Code First esegue un confronto delle migrazioni della cartella **Migrazioni** con le migrazioni applicate al database. Viene individuata la migrazione **AddBlogUrl** da applicare e viene eseguita la migrazione.

Il database **MigrationsDemo.BlogContext** è ora aggiornato in modo da includere la colonna **Url** nella tabella **Blogs**.

**Personalizzazione delle migrazioni**

Nei passaggi precedenti è stata creata ed eseguita una migrazione senza apportare alcuna modifica. Viene ora descritto come modificare il codice generato per impostazione predefinita.

* È ora possibile apportare alcune modifiche al modello: aggiungere una nuova proprietà **Rating** alla classe **Blog**

C#Copia

public int Rating { get; set; }

* Aggiungere anche una nuova classe **Post**

C#Copia

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; }

}

* Si procede anche all'aggiunta delle raccolta **Posts** alla classe **Blog** per creare l'altro elemento della relazione tra **Blog** e **Post**

C#Copia

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

Viene usato il comando **Add-Migration** per consentire a Migrazioni Code First di eseguire automaticamente lo scaffolding nella migrazione. La migrazione viene denominata **AddPostClass**.

* Eseguire il comando **Add-Migration AddPostClass** nella console di Gestione pacchetti.

Sebbene Migrazioni Code First abbia eseguito lo scaffolding di queste modifiche, è possibile modificare ulteriori elementi:

1. Aggiungere innanzitutto un indice univoco alla colonna **Posts.Title** (aggiunta alle righe 22 e 29 del codice seguente).
2. Viene anche aggiunta una colonna **Blogs.Rating** non nullable. Se la tabella include dati esistenti, viene assegnato alla tabella il valore predefinito CLR del tipo di dati per la nuova colonna (poiché il tipo di dati di Rating è integer, il valore è **0**). Si supponga tuttavia di voler specificare un valore predefinito **3** in modo che le righe esistenti della tabella **Blogs** abbiano inizio con una classificazione ragionevole. (È possibile visualizzare il valore predefinito specificato nella riga 24 del codice riportato di seguito)

C#Copia

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");

}

}

}

La migrazione modificata è ora pronta. Usare **Update-Database** per aggiornare il database. Specificare questa volta il flag **-Verbose** per poter visualizzare il codice SQL eseguito da Migrazioni Code First.

* Eseguire il comando **Update-Database –Verbose** nella console di Gestione pacchetti.

**Spostamento di dati/SQL personalizzato**

Le operazioni sulle migrazioni eseguite finora non comportano alcuna modifica o spostamento dei dati. Di seguito viene descritto il caso in cui è necessario spostare alcuni dati. Sebbene non sia disponibile alcun supporto nativo per lo spostamento dei dati, è possibile eseguire alcuni comandi SQL arbitrari in qualsiasi posizione all'interno dello script.

* Aggiungere una proprietà **Post.Abstract** al modello. In seguito verrà prepopolato **Abstract** per i post esistenti usando un testo della parte iniziale della colonna **Content**.

C#Copia

public string Abstract { get; set; }

Viene usato il comando **Add-Migration** per consentire a Migrazioni Code First di eseguire automaticamente lo scaffolding nella migrazione.

* Eseguire il comando **Add-Migration AddPostAbstract** nella console di Gestione pacchetti.
* Sebbene la migrazione generata gestisca le modifiche dello schema, si vuole prepopolare anche la colonna **Abstract** usando i primi 100 caratteri del contenuto di ogni post. Per eseguire questa operazione, passare a SQL ed eseguire un'istruzione **UPDATE** dopo aver aggiunto la colonna. (L'aggiunta si trova nella riga 12 del codice riportato di seguito)

C#Copia

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");

}

}

}

La migrazione modificata è ora pronta. Usare **Update-Database** per aggiornare il database. Specificare il flag **-Verbose** per visualizzare il codice SQL eseguito nel database.

* Eseguire il comando **Update-Database –Verbose** nella console di Gestione pacchetti.

**Eseguire la migrazione a una versione specifica (incluso il downgrade)**

Sebbene fino ad ora sia sempre stato eseguito l'aggiornamento alla migrazione più recente, in alcuni casi può essere necessario eseguire l'aggiornamento o il downgrade a una migrazione specifica.

Si supponga di volere eseguire la migrazione del database allo stato in cui si trovava dopo l'esecuzione della migrazione **AddBlogUrl**. È possibile usare l'opzione **-TargetMigration** per eseguire il downgrade alla migrazione specifica.

* Eseguire il comando **Update-Database -TargetMigration: AddBlogUrl** nella console di Gestione pacchetti.

Il comando esegue lo script di downgrade per le migrazioni **AddBlogAbstract** e **AddPostClass**.

Se si vuole eseguire il rollback a un database vuoto, è possibile usare il comando **Update-Database –TargetMigration: $InitialDatabase**.

**Recupero di uno script SQL**

Dopo aver controllato le modifiche nel controllo del codice sorgente, se un altro sviluppatore vuole apportare le modifiche nel proprio computer è sufficiente che esegua la sincronizzazione. Dopo aver ottenuto le nuove migrazioni, sarà sufficiente eseguire il comando Update-Database per applicare le modifiche in locale. Tuttavia, se si vuole eseguire il push delle modifiche in un server di test, ed eventualmente in produzione, può essere necessario uno script SQL da inviare al DBA.

* Eseguire il comando **Update-Database** e specificare il flag **-Script** in modo che le modifiche vengano scritte in uno script anziché essere applicate. Vengono anche specificate una migrazione di origine e una migrazione di destinazione per cui generare lo script. Si vuole che lo script passi da un database vuoto (**$InitialDatabase**) alla versione più recente (migrazione **AddPostAbstract**). *Se non si specifica una migrazione di destinazione, Migrazioni userà la migrazione di più recente come destinazione. Se non si specifica una migrazione di origine, Migrazioni userà lo stato corrente del database.*
* Eseguire il comando **Update-Database -Script -SourceMigration: $InitialDatabase -TargetMigration: AddPostAbstract** nella console di Gestione pacchetti

Migrazioni Code First esegue la pipeline di migrazione scrivendo le modifiche in un file con estensione sql anziché applicarle. Dopo essere stato generato, lo script viene aperto automaticamente in Visual Studio dove è possibile visualizzarlo o salvarlo.

**Generazione di script idempotenti**

A partire da EF6, se si specifica **-SourceMigration $InitialDatabase**, viene generato uno script 'idempotente'. Gli script idempotenti possono eseguire l'aggiornamento del database da qualsiasi versione alla versione più recente o alla versione specificata con **-TargetMigration**. Lo script generato include la logica per controllare la tabella **\_\_MigrationsHistory** e applicare solo le modifiche non sono state applicate precedentemente.

**Aggiornamento automatico all'avvio dell'applicazione (Inizializzatore MigrateDatabaseToLatestVersion)**

Durante la distribuzione dell'applicazione è possibile scegliere di aggiornare automaticamente il database e applicare eventuali migrazioni in sospeso all'avvio dell'applicazione. A tale scopo, registrare l'inizializzatore di database **MigrateDatabaseToLatestVersion**. Un inizializzatore di database contiene solo la logica usata per assicurarsi che il database sia configurato correttamente. Questa logica viene eseguita la prima volta che il contesto viene usato all'interno del processo dell'applicazione (**AppDomain**).

È possibile aggiornare il file **Program.cs**, come illustrato di seguito, per impostare l'inizializzatore **MigrateDatabaseToLatestVersion** per BlogContext prima che venga usato il contesto (riga 14). Si noti che è anche necessario aggiungere un'istruzione using per lo spazio dei nomi **System.Data.Entity** (riga 5).

*Quando si crea un'istanza di questo inizializzatore è necessario specificare il tipo di contesto (****BlogContext****) e la configurazione delle migrazioni (****Configurazioni****). La configurazione delle migrazioni è la classe aggiunta alla cartella****Migrazioni****quando vengono attivate le migrazioni.*

C#Copia

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();

}

}

}

Ad ogni esecuzione, l'applicazione verificherà se il database di destinazione è aggiornato e, in caso negativo, applicherà tutte le migrazioni in sospeso.

**Migrazioni Code First automatico**

* 23/10/2016
* 5 minuti per la lettura

Le migrazioni automatiche consentono di usare Migrazioni Code First senza avere un file di codice nel progetto per ogni modifica apportata. Non tutte le modifiche possono essere applicate automaticamente. ad esempio, per rinominare le colonne è necessario utilizzare una migrazione basata su codice.

**Nota**

Questo articolo presuppone che l'utente sappia come usare Migrazioni Code First negli scenari di base. In caso contrario, sarà necessario leggere [**migrazioni Code First**](https://docs.microsoft.com/it-it/ef/ef6/modeling/code-first/migrations/) prima di continuare.

**Raccomandazione per gli ambienti Team**

È possibile alternare le migrazioni automatiche e basate su codice, ma ciò non è consigliato negli scenari di sviluppo del team. Se si fa parte di un team di sviluppatori che usano il controllo del codice sorgente, è consigliabile usare migrazioni puramente automatiche o migrazioni esclusivamente basate su codice. Considerate le limitazioni delle migrazioni automatiche, è consigliabile usare migrazioni basate sul codice in ambienti team.

**Creazione del modello iniziale e del database**

Per iniziare a usare le migrazioni sono necessari un progetto e un modello Code First. In questa procedura dettagliata verranno usati i modelli tradizionali **Blog** e **Post**.

* Creare una nuova applicazione console **MigrationsAutomaticDemo**
* Aggiungere la versione più recente del pacchetto NuGet **EntityFramework** al progetto
  + **Strumenti –> Library Package Manager (Gestione pacchetti librerie) –> Console di Gestione pacchetti**
  + Eseguire il comando **Install-Package EntityFramework**
* Aggiungere un file **Model.cs** con il codice riportato di seguito. Questo codice definisce una singola classe **Blog** che rappresenta il modello di dominio e una classe **BlogContext** che rappresenta il contesto Code First di Entity Framework

C#Copia

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; }

}

}

* È ora possibile usare il modello per eseguire l'accesso ai dati. Aggiornare il file **Program.cs** con il codice riportato di seguito.

C#Copia

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...");

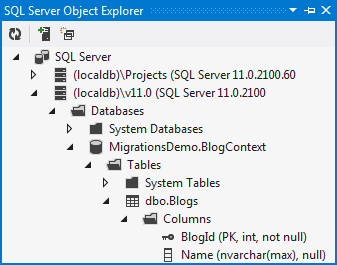
Console.ReadKey();

}

}

}

* Eseguire l'applicazione e si noterà che viene creato un database **MigrationsAutomaticCodeDemo. BlogContext** .



**Abilitare le migrazioni**

È ora possibile apportare ulteriori modifiche al modello.

* Introdurre una proprietà Url nella classe Blog.

C#Copia

public string Url { get; set; }

Se si esegue di nuovo l'applicazione, viene generata un'eccezione InvalidOperationException con il messaggio *The model backing the 'BlogContext' context has changed since the database was created. Valutare la possibilità di usare migrazioni Code First per aggiornare il database (* [*http://go.microsoft.com/fwlink/?LinkId=238269*](https://go.microsoft.com/fwlink/?LinkId=238269)*).*

Come suggerisce l'eccezione, è ora possibile iniziare a usare Migrazioni Code First. Poiché si vogliono usare le migrazioni automatiche, si specifica l'opzione **-EnableAutomaticMigrations** .

* Eseguire il comando **Enable-Migrations-EnableAutomaticMigrations** nella console di gestione pacchetti. questo comando ha aggiunto una cartella **Migrations** al progetto. Questa nuova cartella contiene un file:
* **La classe Configuration.** Questa classe consente di configurare il comportamento di Migrazioni per il contesto. In questa procedura dettagliata verrà usata solo la configurazione predefinita. *Poiché è presente un solo contesto Code First nel progetto, Enable-Migrations ha specificato automaticamente il tipo di contesto a cui si applica questa configurazione.*

**Prima migrazione automatica**

Migrazioni Code First include due comandi principali.

* **Add-Migration** esegue lo scaffolding della migrazione successiva in base alle modifiche apportate al modello dalla creazione dell'ultima migrazione
* **Update-Database** applica tutte le migrazioni in sospeso al database

Si eviterà di usare l'aggiunta della migrazione (a meno che non sia effettivamente necessario) e si concentrerà su come consentire a Migrazioni Code First di calcolare e applicare automaticamente le modifiche. Usare **Update-database** per ottenere migrazioni Code First per eseguire il push delle modifiche al modello (la nuova proprietà **Blog. ur**l) nel database.

* Eseguire il comando **Update-database** nella console di gestione pacchetti.

Il database **MigrationsAutomaticDemo. BlogContext** è ora aggiornato per includere la colonna **URL** nella tabella **Blogs** .

**Seconda migrazione automatica**

Apportare un'altra modifica e consentire a Migrazioni Code First di eseguire automaticamente il push delle modifiche apportate al database.

* Aggiungere anche una nuova classe **Post**

C#Copia

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; }

}

* Si procede anche all'aggiunta delle raccolta **Posts** alla classe **Blog** per creare l'altro elemento della relazione tra **Blog** e **Post**

C#Copia

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

A questo punto, usare **Update-database** per rendere aggiornato il database. Specificare questa volta il flag **-Verbose** per poter visualizzare il codice SQL eseguito da Migrazioni Code First.

* Eseguire il comando **Update-Database –Verbose** nella console di Gestione pacchetti.

**Aggiunta di una migrazione basata su codice**

Si osservi ora che è possibile usare una migrazione basata sul codice per.

* Aggiungere una proprietà **rating** alla classe **Blog**

C#Copia

public int Rating { get; set; }

È possibile eseguire **Update-database** per effettuare il push di queste modifiche nel database. Tuttavia, viene aggiunta una colonna di **Blog. rating** che non ammette i valori null. se nella tabella sono presenti dati esistenti, verrà assegnato il valore predefinito CLR del tipo di dati per la nuova colonna (rating è Integer, in modo che sia **0**). Si supponga tuttavia di voler specificare un valore predefinito **3** in modo che le righe esistenti della tabella **Blogs** abbiano inizio con una classificazione ragionevole. Usare il comando Add-Migration per scrivere questa modifica in una migrazione basata sul codice in modo che sia possibile modificarla. Il comando **Add-Migration** consente di assegnare un nome a queste migrazioni, quindi è sufficiente chiamare la nostra **AddBlogRating**.

* Eseguire il comando **Add-Migration AddBlogRating** nella console di gestione pacchetti.
* Nella cartella **migrazioni** è ora disponibile una nuova migrazione di **AddBlogRating** . Il nome del file di migrazione è preceduto da un timestamp per facilitare l'ordinamento. Modificare il codice generato per specificare il valore predefinito 3 per Blog. rating (riga 10 nel codice riportato di seguito)

*La migrazione dispone anche di un file code-behind che acquisisce alcuni metadati. Questi metadati consentiranno Migrazioni Code First di replicare le migrazioni automatiche eseguite prima di questa migrazione basata sul codice. Questo è importante se un altro sviluppatore vuole eseguire le migrazioni o quando è il momento di distribuire l'applicazione.*

C#Copia

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");

}

}

}

La migrazione modificata è ora pronta. Usare **Update-Database** per aggiornare il database.

* Eseguire il comando **Update-database** nella console di gestione pacchetti.

**Torna a migrazioni automatiche**

A questo punto è possibile tornare alla migrazione automatica per le modifiche più semplici. Migrazioni Code First si occuperà di eseguire le migrazioni automatiche e basate sul codice nell'ordine corretto, in base ai metadati archiviati nel file code-behind per ogni migrazione basata sul codice.

* Aggiungere una proprietà post. Abstract al modello

C#Copia

public string Abstract { get; set; }

A questo punto è possibile usare **Update-database** per ottenere migrazioni Code First per eseguire il push di questa modifica nel database usando una migrazione automatica.

* Eseguire il comando **Update-database** nella console di gestione pacchetti.

**Riepilogo**

In questa procedura dettagliata è stato illustrato come utilizzare le migrazioni automatiche per eseguire il push delle modifiche del modello nel database. È stato anche illustrato come eseguire il patibolo ed eseguire migrazioni basate su codice tra le migrazioni automatiche quando è necessario un maggiore controllo.

**Migrazioni Code First con un database esistente**

* 23/10/2016
* 7 minuti per la lettura

**EF 4.3 e versioni successive** : le funzionalità, le API e così via descritte in questa pagina sono state introdotte in Entity Framework 4,1. Se si usa una versione precedente, le informazioni qui riportate, o parte di esse, non sono applicabili.

Questo articolo illustra l'uso di Migrazioni Code First con un database esistente, uno che non è stato creato da Entity Framework.

Questo articolo presuppone che l'utente sappia come usare Migrazioni Code First negli scenari di base. In caso contrario, sarà necessario leggere [**migrazioni Code First**](https://docs.microsoft.com/it-it/ef/ef6/modeling/code-first/migrations/) prima di continuare.

**Screencast**

Se si preferisce guardare uno screencast anziché leggere questo articolo, i due video seguenti riguardano lo stesso contenuto di questo articolo.

**Video uno: "migrazioni-dietro le quinte"**

[Questo screencast](https://channel9.msdn.com/blogs/ef/migrations-under-the-hood) illustra il modo in cui le migrazioni tracciano e usano le informazioni sul modello per rilevare le modifiche al modello.

**Video due: "migrazioni-database esistenti"**

Basandosi sui concetti del video precedente, [questo screencast](https://channel9.msdn.com/blogs/ef/migrations-existing-databases) illustra come abilitare e usare le migrazioni con un database esistente.

**Passaggio 1: creare un modello**

Il primo passaggio consiste nel creare un modello di Code First destinato al database esistente. L'argomento [Code First a un database esistente](https://docs.microsoft.com/it-it/ef/ef6/modeling/code-first/workflows/existing-database) fornisce istruzioni dettagliate su come eseguire questa operazione.

**Nota**

Prima di apportare modifiche al modello che richiederebbero modifiche allo schema del database, è importante attenersi ai passaggi rimanenti di questo argomento. Per la procedura seguente è necessario che il modello sia sincronizzato con lo schema del database.

**Passaggio 2: abilitare le migrazioni**

Il passaggio successivo consiste nell'abilitare le migrazioni. È possibile eseguire questa operazione eseguendo il comando **Enable-Migrations** nella console di gestione pacchetti.

Questo comando crea una cartella nella soluzione denominata migrazioni e inserisce una singola classe al suo interno chiamata configurazione. La classe di configurazione consente di configurare le migrazioni per l'applicazione. per altre informazioni, vedere l'argomento [migrazioni Code First](https://docs.microsoft.com/it-it/ef/ef6/modeling/code-first/migrations/) .

**Passaggio 3: aggiungere una migrazione iniziale**

Al termine della creazione e dell'applicazione delle migrazioni al database locale, è possibile che si desideri applicare tali modifiche anche ad altri database. È ad esempio possibile che il database locale sia un database di prova e che sia necessario applicare le modifiche anche a un database di produzione e/o ad altri database di prova degli sviluppatori. Per questo passaggio è possibile scegliere tra due opzioni, a seconda che lo schema di altri database sia vuoto o che attualmente corrisponda allo schema del database locale.

* **Opzione uno: utilizzare lo schema esistente come punto di partenza.** È consigliabile utilizzare questo approccio quando gli altri database a cui verranno applicate le migrazioni in futuro avranno lo stesso schema del database locale. Ad esempio, è possibile usare questo se il database di test locale corrisponde attualmente al V1 del database di produzione e in seguito si applicano queste migrazioni per aggiornare il database di produzione alla versione V2.
* **Opzione due: usare il database vuoto come punto di partenza.** È consigliabile utilizzare questo approccio quando altri database a cui verranno applicate le migrazioni in futuro sono vuoti (o non sono ancora presenti). Ad esempio, è possibile usarlo se si è iniziato a sviluppare l'applicazione usando un database di test, ma senza usare le migrazioni e in un secondo momento si vuole creare un database di produzione da zero.

**Opzione uno: utilizzare lo schema esistente come punto di partenza**

Migrazioni Code First utilizza uno snapshot del modello archiviato nella migrazione più recente per rilevare le modifiche apportate al modello (è possibile trovare informazioni dettagliate su questo in [migrazioni Code First in ambienti Team](https://docs.microsoft.com/it-it/ef/ef6/modeling/code-first/migrations/teams)). Poiché si presuppone che i database abbiano già lo schema del modello corrente, verrà generata una migrazione vuota (no-op) con il modello corrente come snapshot.

1. Eseguire il comando **Add-Migration InitialCreate – IgnoreChanges** nella console di gestione pacchetti. In questo modo viene creata una migrazione vuota con il modello corrente come snapshot.
2. Eseguire il comando **Update-database** nella console di gestione pacchetti. Verrà applicata la migrazione InitialCreate al database. Poiché la migrazione effettiva non contiene alcuna modifica, viene semplicemente aggiunta una riga alla \_ \_ tabella MigrationsHistory che indica che la migrazione è già stata applicata.

**Opzione due: usare un database vuoto come punto di partenza**

In questo scenario sono necessarie migrazioni per poter creare l'intero database da zero, incluse le tabelle già presenti nel database locale. Verrà generata una migrazione InitialCreate che include la logica per la creazione dello schema esistente. Si renderà quindi il database esistente simile a questa migrazione è già stata applicata.

1. Eseguire il comando **Add-Migration InitialCreate** nella console di gestione pacchetti. In questo modo viene creata una migrazione per creare lo schema esistente.
2. Impostare come commento tutto il codice nel metodo up della migrazione appena creata. Questo consentirà di applicare la migrazione al database locale senza provare a ricreare tutte le tabelle e così via.
3. Eseguire il comando **Update-database** nella console di gestione pacchetti. Verrà applicata la migrazione InitialCreate al database. Poiché la migrazione effettiva non contiene alcuna modifica (perché è stata impostata temporaneamente come commento), viene semplicemente aggiunta una riga alla \_ \_ tabella MigrationsHistory che indica che la migrazione è già stata applicata.
4. Annulla il commento del codice nel metodo up. Ciò significa che quando questa migrazione viene applicata ai database futuri, lo schema già esistente nel database locale verrà creato dalle migrazioni.

**Aspetti da tenere presenti**

Quando si usano le migrazioni su un database esistente, è necessario tenere presenti alcuni aspetti.

**I nomi predefiniti/calcolati potrebbero non corrispondere allo schema esistente**

Le migrazioni specificano in modo esplicito i nomi delle colonne e delle tabelle durante l'impalcatura di una migrazione. Tuttavia, esistono altri oggetti di database per cui la migrazione calcola un nome predefinito durante l'applicazione delle migrazioni. Sono inclusi gli indici e i vincoli FOREIGN KEY. Quando la destinazione è uno schema esistente, i nomi calcolati potrebbero non corrispondere a quello effettivamente esistente nel database.

Di seguito sono riportati alcuni esempi di quando è necessario tenere presente quanto segue:

**Se è stato usato il valore ' opzione uno: USA schema esistente come punto di partenza ' dal passaggio 3:**

* Se le modifiche future apportate al modello richiedono la modifica o l'eliminazione di uno degli oggetti di database denominati in modo diverso, sarà necessario modificare la migrazione con impalcatura per specificare il nome corretto. Le API Migrations hanno un parametro nome facoltativo che consente di eseguire questa operazione. Ad esempio, lo schema esistente può includere una tabella post con una colonna chiave esterna BlogId con un indice denominato IndexFk \_ BlogId. Tuttavia, per le migrazioni predefinite si prevede che questo indice sia denominato IX \_ BlogId. Se si modifica il modello che comporta l'eliminazione di questo indice, sarà necessario modificare la chiamata DropIndex con impalcatura per specificare il nome del BlogId IndexFk \_ .

**Se è stato usato "opzione due: USA database vuoto come punto di partenza" dal passaggio 3:**

* Il tentativo di eseguire il metodo down della migrazione iniziale (ovvero il ripristino di un database vuoto) sul database locale potrebbe non riuscire perché le migrazioni tenterà di eliminare gli indici e i vincoli di chiave esterna utilizzando i nomi non corretti. Questo influirà solo sul database locale, perché gli altri database verranno creati da zero utilizzando il metodo up della migrazione iniziale. Se si desidera eseguire il downgrade del database locale esistente a uno stato vuoto, è più semplice eseguire questa operazione manualmente, eliminando il database o eliminando tutte le tabelle. Al termine di questo downgrade iniziale, tutti gli oggetti di database verranno ricreati con i nomi predefiniti, pertanto questo problema non si presenta di nuovo.
* Se le modifiche future apportate al modello richiedono la modifica o l'eliminazione di uno degli oggetti di database denominati in modo diverso, questo non funzionerà con il database locale esistente, perché i nomi non corrispondono ai valori predefiniti. Tuttavia, funzionerà con i database creati ' da zerò perché utilizzeranno i nomi predefiniti scelti dalle migrazioni. Queste modifiche possono essere apportate manualmente nel database locale esistente oppure è consigliabile che le migrazioni ricreino il database da zero, come in altri computer.
* I database creati utilizzando il metodo up della migrazione iniziale possono variare leggermente rispetto al database locale, poiché verranno utilizzati i nomi predefiniti calcolati per gli indici e i vincoli di chiave esterna. È anche possibile che si verifichino indici aggiuntivi poiché le migrazioni creeranno indici per le colonne chiave esterne per impostazione predefinita. questa situazione potrebbe non essere stata eseguita nel database locale originale.

**Non tutti gli oggetti di database sono rappresentati nel modello**

Gli oggetti di database che non fanno parte del modello non verranno gestiti dalle migrazioni. Questo può includere viste, stored procedure, autorizzazioni, tabelle che non fanno parte del modello, indici aggiuntivi e così via.

Di seguito sono riportati alcuni esempi di quando è necessario tenere presente quanto segue:

* Indipendentemente dall'opzione scelta nel passaggio 3, se le modifiche future apportate al modello richiedono la modifica o l'eliminazione di queste migrazioni di oggetti aggiuntivi non sapranno apportare tali modifiche. Se, ad esempio, si rilascia una colonna con un indice aggiuntivo, le migrazioni non noteranno di eliminare l'indice. Sarà necessario aggiungerlo manualmente alla migrazione con impalcature.
* Se è stata usata l'opzione "due: USA database vuoto come punto di partenza", questi oggetti aggiuntivi non verranno creati dal metodo up della migrazione iniziale. Se lo si desidera, è possibile modificare i metodi up e down per gestire questi oggetti aggiuntivi. Per gli oggetti che non sono supportati in modo nativo nell'API Migrations, ad esempio views, è possibile usare il metodo [SQL](https://msdn.microsoft.com/library/system.data.entity.migrations.dbmigration.sql.aspx) per eseguire SQL non elaborato per crearli o eliminarli.

**Personalizzazione della tabella di cronologia delle migrazioni**

* 23/10/2016
* 3 minuti per la lettura

**Solo EF6 e versioni successive**: funzionalità, API e altri argomenti discussi in questa pagina sono stati introdotti in Entity Framework 6. Se si usa una versione precedente, le informazioni qui riportate, o parte di esse, non sono applicabili.

**Nota**

Questo articolo presuppone che l'utente sappia come usare Migrazioni Code First negli scenari di base. In caso contrario, sarà necessario leggere [**migrazioni Code First**](https://docs.microsoft.com/it-it/ef/ef6/modeling/code-first/migrations/) prima di continuare.

**Che cos'è la tabella di cronologia delle migrazioni?**

Tabella di cronologia migrazioni è una tabella utilizzata da Migrazioni Code First per archiviare i dettagli sulle migrazioni applicate al database. Per impostazione predefinita, il nome della tabella nel database è \_ \_ MigrationHistory e viene creato quando si applica la prima migrazione al database. In Entity Framework 5 questa tabella è una tabella di sistema se l'applicazione utilizza il database di Microsoft SQL Server. Questa operazione è stata modificata in Entity Framework 6, ma la tabella di cronologia delle migrazioni non è più contrassegnata come tabella di sistema.

**Perché personalizzare la tabella di cronologia delle migrazioni?**

La tabella di cronologia delle migrazioni dovrebbe essere utilizzata esclusivamente da Migrazioni Code First e modificarla manualmente può suddividere le migrazioni. Tuttavia, a volte la configurazione predefinita non è adatta e la tabella deve essere personalizzata, ad esempio:

* Per abilitare un provider di migrazionidi terze parti, è necessario modificare i nomi e/o i facet delle colonne
* Si desidera modificare il nome della tabella
* È necessario usare uno schema non predefinito per la \_ \_ tabella MigrationHistory
* È necessario archiviare dati aggiuntivi per una determinata versione del contesto e pertanto è necessario aggiungere una colonna aggiuntiva alla tabella.

**Parole di precauzione**

Modificare la tabella di cronologia della migrazione è potente, ma è necessario prestare attenzione a non esagerare. EF Runtime attualmente non controlla se la tabella di cronologia delle migrazioni personalizzate è compatibile con il Runtime. In caso contrario, l'applicazione potrebbe interrompersi in fase di esecuzione o comportarsi in modi imprevedibili. Questa operazione è ancora più importante se si usano più contesti per database, nel qual caso più contesti possono usare la stessa tabella di cronologia della migrazione per archiviare le informazioni sulle migrazioni.

**Come personalizzare la tabella di cronologia delle migrazioni**

Prima di iniziare, è necessario tenere presente che è possibile personalizzare la tabella di cronologia delle migrazioni solo prima di applicare la prima migrazione. A questo punto, al codice.

In primo luogo, sarà necessario creare una classe derivata dalla classe System. Data. Entity. Migrations. History. HistoryContext. La classe HistoryContext è derivata dalla classe DbContext, quindi la configurazione della tabella di cronologia delle migrazioni è molto simile alla configurazione dei modelli EF con l'API Fluent. È sufficiente eseguire l'override del metodo OnModelCreating e usare l'API Fluent per configurare la tabella.

**Nota**

In genere, quando si configurano i modelli EF non è necessario chiamare base. OnModelCreating () dal metodo OnModelCreating sottoposto a override poiché DbContext. OnModelCreating () ha un corpo vuoto. Questa situazione non si verifica quando si configura la tabella di cronologia delle migrazioni. In questo caso, la prima operazione da eseguire nell'override di OnModelCreating () consiste nell'eseguire effettivamente la chiamata di base. OnModelCreating (). Verrà configurata la tabella di cronologia delle migrazioni con la modalità predefinita, che verrà quindi modificata nel metodo di override.

Si noti che si vuole rinominare la tabella di cronologia delle migrazioni e inserirla in uno schema personalizzato denominato "admin". Inoltre, l'amministratore di database desidera rinominare la colonna MigrationId in ID migrazione \_ . È possibile ottenere questo risultato creando la classe seguente derivata da HistoryContext:

C#Copia

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");

}

}

}

Quando il HistoryContext personalizzato è pronto, è necessario fare in modo che EF lo renda conto mediante la registrazione tramite la [configurazione basata su codice](https://msdn.com/data/jj680699):

C#Copia

using System.Data.Entity;

namespace CustomizableMigrationsHistoryTableSample

{

public class ModelConfiguration : DbConfiguration

{

public ModelConfiguration()

{

this.SetHistoryContext("System.Data.SqlClient",

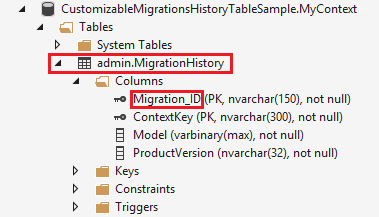
(connection, defaultSchema) => new MyHistoryContext(connection, defaultSchema));

}

}

}

Questo è molto simile. A questo punto è possibile passare alla console di gestione pacchetti, abilitare-Migrations, Add-Migration e infine Update-database. Ciò comporta l'aggiunta al database di una tabella di cronologia delle migrazioni configurata in base ai dettagli specificati nella classe derivata HistoryContext.



**Utilizzo di migrate.exe**

* 23/10/2016
* 4 minuti per la lettura

Migrazioni Code First può essere utilizzato per aggiornare un database dall'interno di Visual Studio, ma può essere eseguito anche tramite lo strumento da riga di comando migrate.exe. Questa pagina fornirà una rapida panoramica su come usare migrate.exe per eseguire le migrazioni in un database.

**Nota**

Questo articolo presuppone che l'utente sappia come usare Migrazioni Code First negli scenari di base. In caso contrario, sarà necessario leggere [**migrazioni Code First**](https://docs.microsoft.com/it-it/ef/ef6/modeling/code-first/migrations/) prima di continuare.

**Copia migrate.exe**

Quando si installa Entity Framework usando NuGet migrate.exe si troverà all'interno della cartella Tools del pacchetto scaricato. Nella < cartella del progetto > \ pacchetti \ EntityFramework. < strumenti della versione > \

Una volta migrate.exe, è necessario copiarlo nel percorso dell'assembly che contiene le migrazioni.

Se l'applicazione è destinata a .NET 4 e non a 4,5, sarà necessario copiare anche il **Redirect.config** nel percorso e rinominarlo **migrate.exe.config**. In questo modo migrate.exe ottiene i reindirizzamenti di binding corretti per poter individuare l'assembly del Entity Framework.

| **COPIA MIGRATE.EXE** | |
| --- | --- |
| **.NET 4.5** | **.NET 4,0** |
| File .NET 4,5 | File .NET 4,0 |

**Nota**

migrate.exe non supporta gli assembly x64.

Dopo aver spostato migrate.exe nella cartella corretta, sarà possibile usarlo per eseguire le migrazioni sul database. Tutte le utilità sono progettate per eseguire le migrazioni. Non è possibile generare migrazioni o creare uno script SQL.

**Vedere le opzioni**

ConsoleCopia

Migrate.exe /?

Sopra verrà visualizzata la pagina della Guida associata a questa utilità, si noti che è necessario che il EntityFramework.dll sia nello stesso percorso in cui si esegue migrate.exe per poter funzionare.

**Eseguire la migrazione alla migrazione più recente**

ConsoleCopia

Migrate.exe MyMvcApplication.dll /startupConfigurationFile="..\\web.config"

Quando si esegue migrate.exe l'unico parametro obbligatorio è l'assembly, ovvero l'assembly contenente le migrazioni che si sta tentando di eseguire, ma verranno utilizzate tutte le impostazioni basate sulla convenzione se non si specifica il file di configurazione.

**Eseguire la migrazione a una migrazione specifica**

ConsoleCopia

Migrate.exe MyApp.exe /startupConfigurationFile="MyApp.exe.config" /targetMigration="AddTitle"

Se si desidera eseguire le migrazioni fino a una migrazione specifica, è possibile specificare il nome della migrazione. Questa operazione consente di eseguire tutte le migrazioni precedenti, se necessario, fino alla migrazione specificata.

**Specifica directory di lavoro**

ConsoleCopia

Migrate.exe MyApp.exe /startupConfigurationFile="MyApp.exe.config" /startupDirectory="c:\\MyApp"

Se l'assembly ha dipendenze o legge i file relativi alla directory di lavoro, sarà necessario impostare startupDirectory.

**Specificare la configurazione della migrazione da usare**

ConsoleCopia

Migrate.exe MyAssembly CustomConfig /startupConfigurationFile="..\\web.config"

Se sono presenti più classi di configurazione della migrazione, le classi che ereditano da DbMigrationConfiguration, è necessario specificare che deve essere usato per questa esecuzione. Questa impostazione viene specificata fornendo il secondo parametro facoltativo senza un'opzione come sopra.

**Specificare la stringa di connessione**

ConsoleCopia

Migrate.exe BlogDemo.dll /connectionString="Data Source=localhost;Initial Catalog=BlogDemo;Integrated Security=SSPI" /connectionProviderName="System.Data.SqlClient"

Se si desidera specificare una stringa di connessione dalla riga di comando, è necessario fornire anche il nome del provider. Se non si specifica il nome del provider, verrà generata un'eccezione.

**Problemi comuni**

| **PROBLEMI COMUNI** | |
| --- | --- |
| **Messaggio di errore** | **Soluzione** |
| Eccezione non gestita: System. IO. FileLoadException: Impossibile caricare il file o l'assembly ' EntityFramework, Version = 5.0.0.0, Culture = neutral, PublicKeyToken = b77a5c561934e089' o una delle relative dipendenze. La definizione del manifesto dell'assembly individuato non corrisponde al riferimento all'assembly. (Eccezione da HRESULT: 0x80131040) | Ciò significa in genere che si esegue un'applicazione .NET 4 senza il file di Redirect.config. È necessario copiare il Redirect.config nello stesso percorso migrate.exe e rinominarlo in migrate.exe.config. |
| Eccezione non gestita: System. IO. FileLoadException: Impossibile caricare il file o l'assembly ' EntityFramework, Version = 4.4.0.0, Culture = neutral, PublicKeyToken = b77a5c561934e089' o una delle relative dipendenze. La definizione del manifesto dell'assembly individuato non corrisponde al riferimento all'assembly. (Eccezione da HRESULT: 0x80131040) | Questa eccezione indica che è in esecuzione un'applicazione .NET 4,5 con la Redirect.config copiata nel percorso migrate.exe. Se l'app è .NET 4,5, non è necessario avere il file di configurazione con i reindirizzamenti all'interno di. Eliminare il file di migrate.exe.config. |
| ERRORE: non è possibile aggiornare il database in modo che corrisponda al modello corrente perché sono presenti modifiche in sospeso e la migrazione automatica è disabilitata. Scrivere le modifiche al modello in sospeso in una migrazione basata su codice o abilitare la migrazione automatica. Impostare DbMigrationsConfiguration. AutomaticMigrationsEnabled su true per abilitare la migrazione automatica. | Questo errore si verifica se si esegue la migrazione quando non è stata creata una migrazione per gestire le modifiche apportate al modello e il database non corrisponde al modello. L'aggiunta di una proprietà a una classe di modello e l'esecuzione migrate.exe senza creare una migrazione per aggiornare il database è un esempio. |
| ERRORE: il tipo non è stato risolto per il membro ' System. Data. Entity. Migrations. Design. ToolingFacade + UpdateRunner, EntityFramework, Version = 5.0.0.0, Culture = neutral, PublicKeyToken = b77a5c561934e089'. | Questo errore può essere causato dalla specifica di una directory di avvio non corretta. Deve essere il percorso di migrate.exe |
| Eccezione non gestita: System. NullReferenceException: riferimento all'oggetto non impostato su un'istanza di un oggetto.    in System. Data. Entity. Migrations. Console. Program. Main (String [] args) | Questo problema può essere causato dalla mancata specificazione di un parametro obbligatorio per uno scenario in uso. Ad esempio specificando una stringa di connessione senza specificare il nome del provider. |
| ERRORE: nell'assembly ' ClassLibrary1' è stato trovato più di un tipo di configurazione delle migrazioni. Specificare il nome di quello da usare. | Come indica l'errore, nell'assembly specificato sono presenti più classi di configurazione. È necessario usare l'opzione/configurationType per specificare quale usare. |
| ERRORE: Impossibile caricare il file o l'assembly ' < AssemblyName > ' o una delle relative dipendenze. Il nome o la codebase dell'assembly specificato non è valido. (Eccezione da HRESULT: 0x80131047) | Questa situazione può essere causata dalla specifica di un nome di assembly in modo errato o senza |
| ERRORE: Impossibile caricare il file o l'assembly ' < AssemblyName > ' o una delle relative dipendenze. Tentativo di caricare un programma con un formato non corretto. | Questo errore si verifica se si tenta di eseguire migrate.exe su un'applicazione x64. EF 5,0 e versioni precedenti funzioneranno solo su x86. |

[**https://docs.microsoft.com/it-it/aspnet/mvc/overview/getting-started/getting-started-with-ef-using-mvc/creating-a-more-complex-data-model-for-an-asp-net-mvc-application**](https://docs.microsoft.com/it-it/aspnet/mvc/overview/getting-started/getting-started-with-ef-using-mvc/creating-a-more-complex-data-model-for-an-asp-net-mvc-application)

**Esercitazione: usare migrazioni di EF in un'app MVC ASP.NET e distribuirle in Azure**

* 16/01/2019
* 19 minuti per la lettura

Fino a questo punto, l'applicazione Web di esempio Contoso University è stata eseguita localmente in IIS Express nel computer di sviluppo. Per rendere disponibile un'applicazione reale che altri utenti usino su Internet, è necessario distribuirla a un provider di hosting Web. In questa esercitazione si abilitano le migrazioni Code First e si distribuisce l'applicazione nel cloud in Azure:

* Abilitare Migrazioni Code First. La funzionalità migrazioni consente di modificare il modello di dati e di distribuire le modifiche nell'ambiente di produzione aggiornando lo schema del database senza dover eliminare e ricreare il database.
* Eseguire la distribuzione in Azure. Questo passaggio è facoltativo. è possibile continuare con le esercitazioni rimanenti senza aver distribuito il progetto.

Si consiglia di utilizzare un processo di integrazione continua con il controllo del codice sorgente per la distribuzione, ma in questa esercitazione non vengono trattati tali argomenti. Per altre informazioni, vedere i capitoli [controllo del codice sorgente](https://docs.microsoft.com/it-it/aspnet/aspnet/overview/developing-apps-with-windows-azure/building-real-world-cloud-apps-with-windows-azure/source-control) e [integrazione continua](https://docs.microsoft.com/it-it/aspnet/aspnet/overview/developing-apps-with-windows-azure/building-real-world-cloud-apps-with-windows-azure/continuous-integration-and-continuous-delivery) della [creazione di app Cloud reali con Azure](https://docs.microsoft.com/it-it/aspnet/aspnet/overview/developing-apps-with-windows-azure/building-real-world-cloud-apps-with-windows-azure/introduction).

In questa esercitazione:

* Abilita migrazioni di Code First
* Distribuire l'app in Azure (facoltativo)

**Prerequisiti**

* [Resilienza della connessione e intercettazione dei comandi](https://docs.microsoft.com/it-it/aspnet/mvc/overview/getting-started/getting-started-with-ef-using-mvc/connection-resiliency-and-command-interception-with-the-entity-framework-in-an-asp-net-mvc-application)

**Abilita migrazioni di Code First**

Quando si sviluppa una nuova applicazione, il modello di dati cambia di frequente e, a ogni cambiamento, non è più sincronizzato con il database. Il Entity Framework è stato configurato per eliminare e ricreare automaticamente il database ogni volta che si modifica il modello di dati. Quando si aggiungono, rimuovono o si modificano le classi di entità o si modifica la DbContext classe, alla successiva esecuzione dell'applicazione viene eliminato automaticamente il database esistente, ne viene creato uno nuovo che corrisponde al modello e ne viene eseguito il seeding con dati di test.

Questo metodo che consiste nel mantenere il database sincronizzato con il modello di dati funziona bene fino a quando non si distribuisce l'applicazione nell'ambiente di produzione. Quando l'applicazione è in esecuzione nell'ambiente di produzione, in genere archivia i dati che si desidera mantenere e non si desidera perdere tutto ogni volta che viene apportata una modifica, ad esempio l'aggiunta di una nuova colonna. La funzionalità [migrazioni Code First](https://msdn.microsoft.com/data/jj591621) risolve questo problema abilitando Code First per aggiornare lo schema del database anziché eliminare e ricreare il database. In questa esercitazione l'applicazione verrà distribuita e verrà preparata per consentire le migrazioni.

1. Disabilitare l'inizializzatore configurato in precedenza impostando come commento o eliminando l' contexts elemento aggiunto al file di Web.config dell'applicazione.

XMLCopia

<entityFramework>

<!--<contexts>

<context type="ContosoUniversity.DAL.SchoolContext, ContosoUniversity">

<databaseInitializer type="ContosoUniversity.DAL.SchoolInitializer, ContosoUniversity" />

</context>

</contexts>-->

<defaultConnectionFactory type="System.Data.Entity.Infrastructure.LocalDbConnectionFactory, EntityFramework">

<parameters>

<parameter value="v11.0" />

</parameters>

</defaultConnectionFactory>

<providers>

<provider invariantName="System.Data.SqlClient" type="System.Data.Entity.SqlServer.SqlProviderServices, EntityFramework.SqlServer" />

</providers>

</entityFramework>

1. Inoltre, nel file di *Web.config* dell'applicazione, modificare il nome del database nella stringa di connessione in ContosoUniversity2.

XMLCopia

<connectionStrings>

<add name="SchoolContext" connectionString="Data Source=(LocalDb)\MSSQLLocalDB;Initial Catalog=ContosoUniversity2;Integrated Security=SSPI;" providerName="System.Data.SqlClient" />

</connectionStrings>

Questa modifica consente di configurare il progetto in modo che la prima migrazione crei un nuovo database. Questa operazione non è necessaria, ma in seguito verrà illustrato il motivo per cui è consigliabile.

1. Nel menu **Strumenti** selezionare **Gestione pacchetti NuGet** > **Console di Gestione pacchetti**.
2. Al PM> prompt immettere i comandi seguenti:

textCopia

enable-migrations

add-migration InitialCreate

Il enable-migrations comando crea una cartella *Migrations* nel progetto ContosoUniversity e inserisce in tale cartella un file *Configuration.cs* che è possibile modificare per configurare le migrazioni.

Se il passaggio precedente non è stato superato e viene indicato di modificare il nome del database, le migrazioni troveranno il database esistente ed eseguiranno automaticamente il add-migration comando. Questo significa semplicemente che non verrà eseguito un test del codice delle migrazioni prima di distribuire il database. In un secondo momento, quando si esegue il update-database comando, non si verificherà nulla perché il database esiste già.

Aprire il file *ContosoUniversity\Migrations\Configuration.cs* . Analogamente alla classe dell'inizializzatore illustrata in precedenza, la Configuration classe include un Seed metodo.

C#Copia

internal sealed class Configuration : DbMigrationsConfiguration<ContosoUniversity.DAL.SchoolContext>

{

public Configuration()

{

AutomaticMigrationsEnabled = false;

}

protected override void Seed(ContosoUniversity.DAL.SchoolContext context)

{

// This method will be called after migrating to the latest version.

// You can use the DbSet<T>.AddOrUpdate() helper extension method

// to avoid creating duplicate seed data. E.g.

//

// context.People.AddOrUpdate(

// p => p.FullName,

// new Person { FullName = "Andrew Peters" },

// new Person { FullName = "Brice Lambson" },

// new Person { FullName = "Rowan Miller" }

// );

//

}

}

Lo scopo del metodo [Seed](https://msdn.microsoft.com/library/hh829453(v=vs.103).aspx) è quello di consentire l'inserimento o l'aggiornamento dei dati di test dopo che Code First crea o aggiorna il database. Il metodo viene chiamato quando viene creato il database e ogni volta che lo schema del database viene aggiornato dopo la modifica di un modello di dati.

**Configurare il metodo Seed**

Quando si elimina e si ricrea il database per ogni modifica del modello di dati, si utilizza il metodo della classe inizializzatore Seed per inserire i dati di test, poiché dopo la modifica di ogni modello il database viene eliminato e tutti i dati di test vengono persi. Con Migrazioni Code First, i dati di test vengono conservati dopo le modifiche apportate al database, quindi l'inclusione dei dati di test nel metodo [Seed](https://msdn.microsoft.com/library/hh829453(v=vs.103).aspx) non è in genere necessaria. Infatti, non si vuole che il Seed metodo inserisca dati di test se si usano le migrazioni per distribuire il database in produzione, perché il Seed metodo verrà eseguito in produzione. In tal caso, si desidera che il Seed metodo inserisca nel database solo i dati necessari nell'ambiente di produzione. È ad esempio possibile che il database includa i nomi di reparto effettivi nella Department tabella quando l'applicazione diventa disponibile nell'ambiente di produzione.

Per questa esercitazione si useranno le migrazioni per la distribuzione, ma il Seed metodo inserirà i dati di test in modo da semplificare la verifica del funzionamento della funzionalità dell'applicazione senza dover inserire manualmente una grande quantità di dati.

1. Sostituire il contenuto del file *Configuration.cs* con il codice seguente, che carica i dati di test nel nuovo database.

C#Copia

namespace ContosoUniversity.Migrations

{

using ContosoUniversity.Models;

using System;

using System.Collections.Generic;

using System.Data.Entity;

using System.Data.Entity.Migrations;

using System.Linq;

internal sealed class Configuration : DbMigrationsConfiguration<ContosoUniversity.DAL.SchoolContext>

{

public Configuration()

{

AutomaticMigrationsEnabled = false;

}

protected override void Seed(ContosoUniversity.DAL.SchoolContext context)

{

var students = new List<Student>

{

new Student { FirstMidName = "Carson", LastName = "Alexander",

EnrollmentDate = DateTime.Parse("2010-09-01") },

new Student { FirstMidName = "Meredith", LastName = "Alonso",

EnrollmentDate = DateTime.Parse("2012-09-01") },

new Student { FirstMidName = "Arturo", LastName = "Anand",

EnrollmentDate = DateTime.Parse("2013-09-01") },

new Student { FirstMidName = "Gytis", LastName = "Barzdukas",

EnrollmentDate = DateTime.Parse("2012-09-01") },

new Student { FirstMidName = "Yan", LastName = "Li",

EnrollmentDate = DateTime.Parse("2012-09-01") },

new Student { FirstMidName = "Peggy", LastName = "Justice",

EnrollmentDate = DateTime.Parse("2011-09-01") },

new Student { FirstMidName = "Laura", LastName = "Norman",

EnrollmentDate = DateTime.Parse("2013-09-01") },

new Student { FirstMidName = "Nino", LastName = "Olivetto",

EnrollmentDate = DateTime.Parse("2005-08-11") }

};

students.ForEach(s => context.Students.AddOrUpdate(p => p.LastName, s));

context.SaveChanges();

var courses = new List<Course>

{

new Course {CourseID = 1050, Title = "Chemistry", Credits = 3, },

new Course {CourseID = 4022, Title = "Microeconomics", Credits = 3, },

new Course {CourseID = 4041, Title = "Macroeconomics", Credits = 3, },

new Course {CourseID = 1045, Title = "Calculus", Credits = 4, },

new Course {CourseID = 3141, Title = "Trigonometry", Credits = 4, },

new Course {CourseID = 2021, Title = "Composition", Credits = 3, },

new Course {CourseID = 2042, Title = "Literature", Credits = 4, }

};

courses.ForEach(s => context.Courses.AddOrUpdate(p => p.Title, s));

context.SaveChanges();

var enrollments = new List<Enrollment>

{

new Enrollment {

StudentID = students.Single(s => s.LastName == "Alexander").ID,

CourseID = courses.Single(c => c.Title == "Chemistry" ).CourseID,

Grade = Grade.A

},

new Enrollment {

StudentID = students.Single(s => s.LastName == "Alexander").ID,

CourseID = courses.Single(c => c.Title == "Microeconomics" ).CourseID,

Grade = Grade.C

},

new Enrollment {

StudentID = students.Single(s => s.LastName == "Alexander").ID,

CourseID = courses.Single(c => c.Title == "Macroeconomics" ).CourseID,

Grade = Grade.B

},

new Enrollment {

StudentID = students.Single(s => s.LastName == "Alonso").ID,

CourseID = courses.Single(c => c.Title == "Calculus" ).CourseID,

Grade = Grade.B

},

new Enrollment {

StudentID = students.Single(s => s.LastName == "Alonso").ID,

CourseID = courses.Single(c => c.Title == "Trigonometry" ).CourseID,

Grade = Grade.B

},

new Enrollment {

StudentID = students.Single(s => s.LastName == "Alonso").ID,

CourseID = courses.Single(c => c.Title == "Composition" ).CourseID,

Grade = Grade.B

},

new Enrollment {

StudentID = students.Single(s => s.LastName == "Anand").ID,

CourseID = courses.Single(c => c.Title == "Chemistry" ).CourseID

},

new Enrollment {

StudentID = students.Single(s => s.LastName == "Anand").ID,

CourseID = courses.Single(c => c.Title == "Microeconomics").CourseID,

Grade = Grade.B

},

new Enrollment {

StudentID = students.Single(s => s.LastName == "Barzdukas").ID,

CourseID = courses.Single(c => c.Title == "Chemistry").CourseID,

Grade = Grade.B

},

new Enrollment {

StudentID = students.Single(s => s.LastName == "Li").ID,

CourseID = courses.Single(c => c.Title == "Composition").CourseID,

Grade = Grade.B

},

new Enrollment {

StudentID = students.Single(s => s.LastName == "Justice").ID,

CourseID = courses.Single(c => c.Title == "Literature").CourseID,

Grade = Grade.B

}

};

foreach (Enrollment e in enrollments)

{

var enrollmentInDataBase = context.Enrollments.Where(

s =>

s.Student.ID == e.StudentID &&

s.Course.CourseID == e.CourseID).SingleOrDefault();

if (enrollmentInDataBase == null)

{

context.Enrollments.Add(e);

}

}

context.SaveChanges();

}

}

}

Il metodo [Seed](https://msdn.microsoft.com/library/hh829453(v=vs.103).aspx) accetta l'oggetto contesto di database come parametro di input e il codice nel metodo utilizza tale oggetto per aggiungere nuove entità al database. Per ogni tipo di entità, il codice crea una raccolta di nuove entità, le aggiunge alla proprietà [DbSet](https://msdn.microsoft.com/library/system.data.entity.dbset(v=vs.103).aspx) appropriata e quindi Salva le modifiche nel database. Non è necessario chiamare il metodo [SaveChanges](https://msdn.microsoft.com/library/system.data.entity.dbcontext.savechanges(v=VS.103).aspx) dopo ogni gruppo di entità, come avviene qui, ma ciò consente di individuare l'origine di un problema se si verifica un'eccezione durante la scrittura del codice nel database.

Alcune delle istruzioni che inseriscono dati utilizzano il metodo [AddOrUpdate](https://msdn.microsoft.com/library/system.data.entity.migrations.idbsetextensions.addorupdate(v=vs.103).aspx) per eseguire un'operazione "upsert". Poiché il Seed metodo viene eseguito ogni volta che si esegue il update-database comando, in genere dopo ogni migrazione, non è possibile inserire semplicemente i dati, perché le righe che si sta tentando di aggiungere saranno già presenti dopo la prima migrazione che crea il database. L'operazione "upsert" impedisce gli errori che si verificano se si tenta di inserire una riga già esistente, ma viene ***eseguito l'override*** di eventuali modifiche apportate ai dati durante il test dell'applicazione. Con i dati di test in alcune tabelle potrebbe non essere necessario eseguire questa operazione: in alcuni casi, quando si modificano i dati durante il test si desidera che le modifiche rimangano dopo gli aggiornamenti del database. In tal caso si desidera eseguire un'operazione di inserimento condizionale: inserire una riga solo se non esiste già. Il metodo Seed usa entrambi gli approcci.

Il primo parametro passato al metodo [AddOrUpdate](https://msdn.microsoft.com/library/system.data.entity.migrations.idbsetextensions.addorupdate(v=vs.103).aspx) specifica la proprietà da utilizzare per verificare se una riga esiste già. Per i dati degli studenti di test da fornire, LastName è possibile usare la proprietà per questo scopo, perché ogni cognome nell'elenco è univoco:

C#Copia

context.Students.AddOrUpdate(p => p.LastName, s)

Questo codice presuppone che i cognomi siano univoci. Se si aggiunge manualmente uno studente con un cognome duplicato, alla successiva esecuzione di una migrazione si otterrà l'eccezione seguente:

**La sequenza contiene più di un elemento**

Per informazioni su come gestire dati ridondanti, ad esempio due studenti denominati "Alexander Carson", vedere il post di Blog relativo al [seeding e al debug Entity Framework (EF)](https://blogs.msdn.com/b/rickandy/archive/2013/02/12/seeding-and-debugging-entity-framework-ef-dbs.aspx) nel Blog di Rick Anderson. Per ulteriori informazioni sul AddOrUpdate metodo, vedere la pagina relativa alla [cura del metodo EF 4,3 AddOrUpdate](http://thedatafarm.com/blog/data-access/take-care-with-ef-4-3-addorupdate-method/) nel Blog di Julie Lerman.

Il codice che crea le Enrollment entità presuppone che il ID valore sia presente nelle entità della students raccolta, anche se tale proprietà non è stata impostata nel codice che crea la raccolta.

C#Copia

new Enrollment {

StudentID = students.Single(s => s.LastName == "Alexander").ID,

CourseID = courses.Single(c => c.Title == "Chemistry" ).CourseID,

Grade = Grade.A

},

È possibile usare la ID proprietà qui perché il ID valore è impostato quando si chiama SaveChanges per la students raccolta. EF ottiene automaticamente il valore della chiave primaria quando inserisce un'entità nel database e aggiorna la ID proprietà dell'entità in memoria.

Il codice che aggiunge ogni Enrollment entità al Enrollments set di entità non usa il AddOrUpdate metodo. Verifica se un'entità esiste già e inserisce l'entità se non esiste. Questo approccio consente di mantenere le modifiche apportate a un livello di registrazione usando l'interfaccia utente dell'applicazione. Il codice esegue il ciclo di ogni membro dell' Enrollment [elenco](https://msdn.microsoft.com/library/6sh2ey19.aspx) e se la registrazione non viene trovata nel database, aggiunge la registrazione al database. La prima volta che si aggiorna il database, il database sarà vuoto, quindi verrà aggiunta ogni registrazione.

C#Copia

foreach (Enrollment e in enrollments)

{

var enrollmentInDataBase = context.Enrollments.Where(

s => s.Student.ID == e.Student.ID &&

s.Course.CourseID == e.Course.CourseID).SingleOrDefault();

if (enrollmentInDataBase == null)

{

context.Enrollments.Add(e);

}

}

1. Compilare il progetto.

**Eseguire la prima migrazione**

Quando si esegue il add-migration comando, le migrazioni generano il codice che crea il database da zero. Questo codice si trova anche nella cartella *migrazioni* , nel file denominato \* < timestamp > \_ InitialCreate.cs\*. Il Up metodo della InitialCreate classe crea le tabelle di database che corrispondono ai set di entità del modello di dati e il Down Metodo li elimina.

C#Copia

public partial class InitialCreate : DbMigration

{

public override void Up()

{

CreateTable(

"dbo.Course",

c => new

{

CourseID = c.Int(nullable: false),

Title = c.String(),

Credits = c.Int(nullable: false),

})

.PrimaryKey(t => t.CourseID);

CreateTable(

"dbo.Enrollment",

c => new

{

EnrollmentID = c.Int(nullable: false, identity: true),

CourseID = c.Int(nullable: false),

StudentID = c.Int(nullable: false),

Grade = c.Int(),

})

.PrimaryKey(t => t.EnrollmentID)

.ForeignKey("dbo.Course", t => t.CourseID, cascadeDelete: true)

.ForeignKey("dbo.Student", t => t.StudentID, cascadeDelete: true)

.Index(t => t.CourseID)

.Index(t => t.StudentID);

CreateTable(

"dbo.Student",

c => new

{

ID = c.Int(nullable: false, identity: true),

LastName = c.String(),

FirstMidName = c.String(),

EnrollmentDate = c.DateTime(nullable: false),

})

.PrimaryKey(t => t.ID);

}

public override void Down()

{

DropForeignKey("dbo.Enrollment", "StudentID", "dbo.Student");

DropForeignKey("dbo.Enrollment", "CourseID", "dbo.Course");

DropIndex("dbo.Enrollment", new[] { "StudentID" });

DropIndex("dbo.Enrollment", new[] { "CourseID" });

DropTable("dbo.Student");

DropTable("dbo.Enrollment");

DropTable("dbo.Course");

}

}

Le migrazioni chiamano il metodo Up per implementare le modifiche al modello di dati per una migrazione. Quando si immette un comando per annullare l'aggiornamento, le migrazioni chiamano il metodo Down.

Si tratta della migrazione iniziale creata al momento dell'immissione del add-migration InitialCreate comando. Il parametro ( InitialCreate nell'esempio) viene usato per il nome del file e può essere quello desiderato. in genere si sceglie una parola o una frase che riepiloga le operazioni eseguite nella migrazione. Ad esempio, è possibile denominare un AddDepartmentTable di migrazione successivo " " .

Se la migrazione iniziale è stata creata quando il database esisteva già, il codice di creazione del database viene generato ma non è necessario eseguirlo perché il database corrisponde già al modello di dati. Quando si distribuisce l'app in un altro ambiente in cui il database non esiste ancora, questo codice verrà eseguito per creare il database, è consigliabile quindi testarlo prima. Per questo motivo è stato modificato il nome del database nella stringa di connessione in precedenza — , in modo che le migrazioni possano crearne una nuova da zero.

1. Nella finestra **Console di Gestione pacchetti** immettere il comando seguente:

update-database

Il update-database comando esegue il Up metodo per creare il database, quindi esegue il Seed metodo per popolare il database. Lo stesso processo verrà eseguito automaticamente in produzione dopo la distribuzione dell'applicazione, come si vedrà nella sezione seguente.

1. Utilizzare **Esplora server** per esaminare il database come è stato fatto nella prima esercitazione ed eseguire l'applicazione per verificare che tutto funzioni ancora come prima.

**Distribuisci in Azure**

Fino a questo punto l'applicazione è stata eseguita localmente in IIS Express nel computer di sviluppo. Per renderlo disponibile per l'uso da parte di altri utenti su Internet, è necessario distribuirlo a un provider di hosting Web. In questa sezione dell'esercitazione verrà distribuita in Azure. Questa sezione è facoltativa. è possibile ignorare questo passaggio e continuare con l'esercitazione seguente oppure è possibile adattare le istruzioni riportate in questa sezione per un provider di hosting diverso a scelta.

**Usare migrazioni di Code First per distribuire il database**

Per distribuire il database, si utilizzerà Migrazioni Code First. Quando si crea il profilo di pubblicazione usato per configurare le impostazioni per la distribuzione da Visual Studio, selezionare una casella di controllo con l'etichetta **Aggiorna database**. Questa impostazione fa in modo che il processo di distribuzione configuri automaticamente il file dell'applicazione *Web.config* nel server di destinazione in modo che Code First utilizzi la MigrateDatabaseToLatestVersion classe inizializzatore.

Visual Studio non esegue alcuna operazione con il database durante il processo di distribuzione durante la copia del progetto nel server di destinazione. Quando si esegue l'applicazione distribuita e si accede al database per la prima volta dopo la distribuzione, Code First controlla se il database corrisponde al modello di dati. In caso di mancata corrispondenza, Code First crea automaticamente il database (se non esiste già) o aggiorna lo schema del database alla versione più recente (se esiste un database ma non corrisponde al modello). Se l'applicazione implementa un metodo Migrations Seed , il metodo viene eseguito dopo la creazione del database o l'aggiornamento dello schema.

Il metodo Migrations Seed inserisce i dati di test. Se si esegue la distribuzione in un ambiente di produzione, è necessario modificare il Seed metodo in modo che inserisca solo i dati che si desidera inserire nel database di produzione. Nel modello di dati corrente, ad esempio, potrebbe essere necessario disporre di corsi reali ma studenti fittizi nel database di sviluppo. È possibile scrivere un Seed metodo per il caricamento in fase di sviluppo e quindi impostare come commento gli studenti fittizi prima della distribuzione nell'ambiente di produzione. In alternativa, è possibile scrivere un Seed metodo per caricare solo i corsi e immettere manualmente gli studenti fittizi nel database di test usando l'interfaccia utente dell'applicazione.

**Ottenere un account Azure**

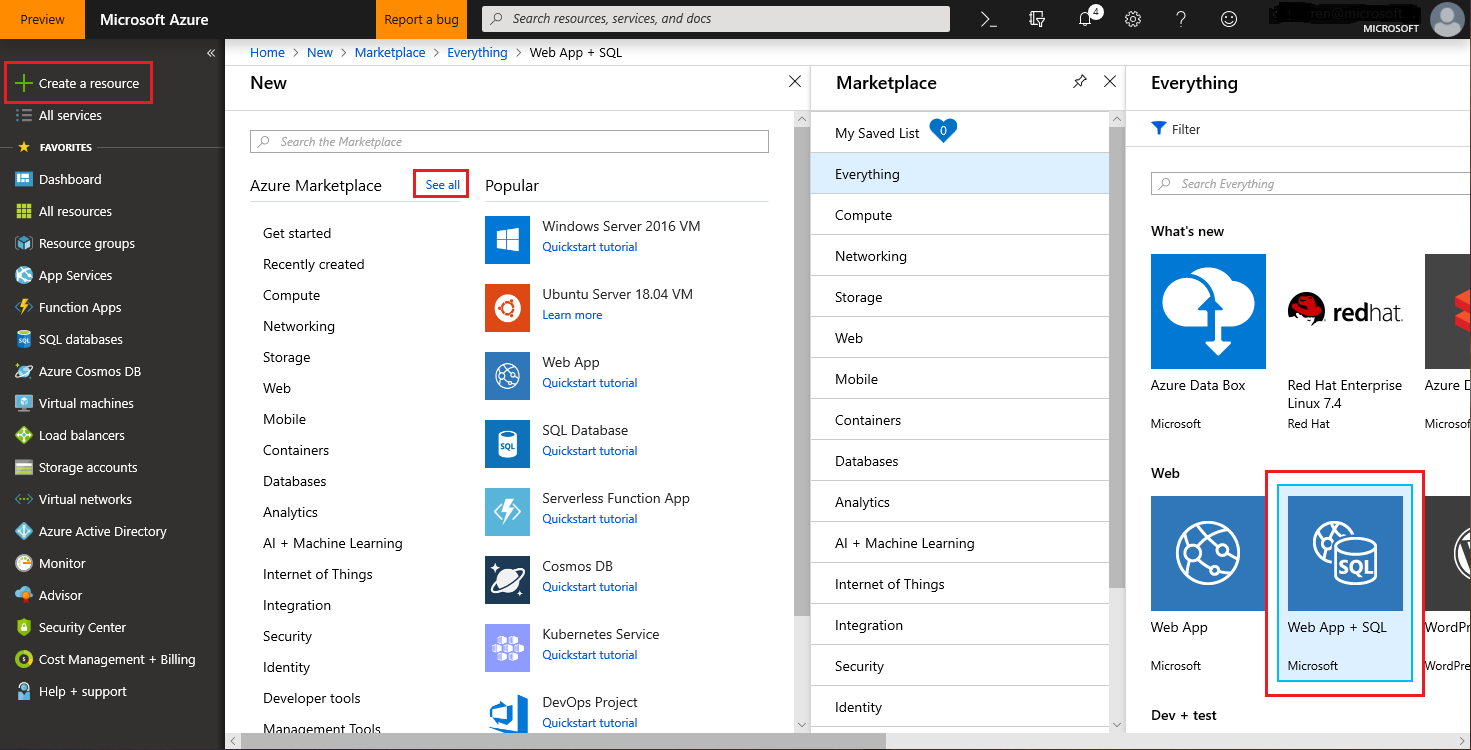
È necessario un account Azure. Se non si dispone già di una sottoscrizione di Visual Studio, è possibile [attivare i vantaggi della sottoscrizione](https://azure.microsoft.com/pricing/member-offers/credit-for-visual-studio-subscribers/). In caso contrario, è possibile creare un account di valutazione gratuito in pochi minuti. Per informazioni dettagliate, vedere la pagina relativa alla [versione di valutazione gratuita di Azure](https://azure.microsoft.com/free/dotnet/).

**Creare un sito Web e un database SQL in Azure**

L'app Web in Azure verrà eseguita in un ambiente di hosting condiviso, il che significa che viene eseguito in macchine virtuali (VM) condivise con altri client di Azure. Un ambiente di hosting condiviso è un punto di partenza economicamente conveniente per iniziare a utilizzare il cloud. In seguito, in caso di incremento del traffico Web, è possibile scalare l'applicazione in modo da soddisfare le nuove esigenze tramite l'esecuzione su macchine virtuali dedicate. Per altre informazioni sulle opzioni di prezzo per il servizio app Azure, vedere [prezzi del servizio app](https://azure.microsoft.com/pricing/details/app-service/).

Il database verrà distribuito nel database SQL di Azure. Il database SQL è un servizio di database relazionale basato sul cloud che si basa su tecnologie SQL Server. Gli strumenti e le applicazioni che funzionano con SQL Server funzionano anche con il database SQL.

1. Nel [portale di gestione di Azure](https://portal.azure.com/)scegliere **Crea una risorsa** nella scheda a sinistra, quindi fare clic su **Visualizza tutto** nel **nuovo** riquadro *(o pannello*) per visualizzare tutte le risorse disponibili. Scegliere **app Web e SQL** nella sezione **Web** del pannello **tutto** . Infine, scegliere **Crea**.



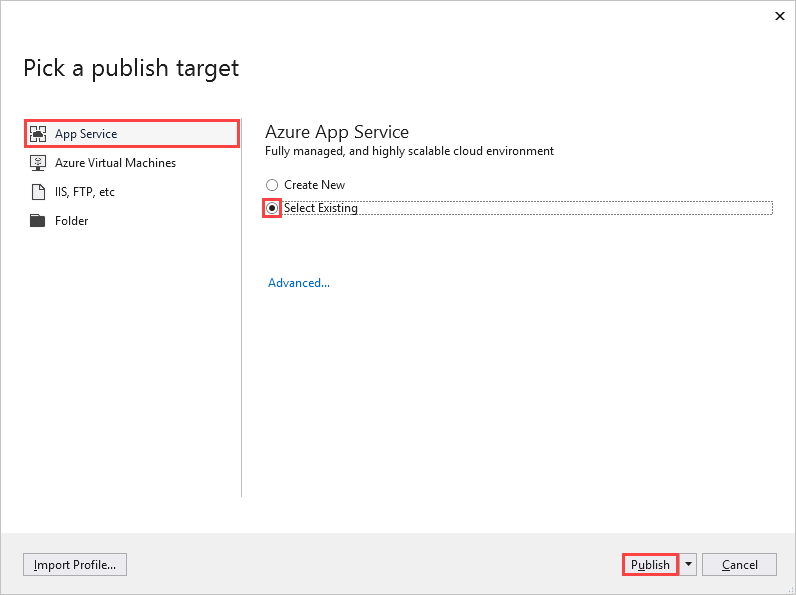
Viene aperto il modulo per creare una nuova **app Web** e una nuova risorsa SQL.

1. Immettere una stringa nella casella **nome app** da usare come URL univoco per l'applicazione. L'URL completo sarà costituito da quanto immesso qui e dal dominio predefinito di app Azure Services (azurewebsites.net). Se il **nome dell'app** è già stato fatto, la procedura guidata invia una notifica con un messaggio di colore rosso. *il nome dell'app non è disponibile* . Se il **nome dell'app** è disponibile, viene visualizzato un segno di spunta verde.
2. Nella casella **sottoscrizione** scegliere la sottoscrizione di Azure in cui si vuole inserire il **servizio app** .
3. Nella casella di testo **gruppo di risorse** scegliere un gruppo di risorse o crearne uno nuovo. Questa impostazione specifica quali data center verrà eseguito il sito Web. Per altre informazioni sui gruppi di risorse, vedere [gruppi di risorse](https://docs.microsoft.com/it-it/azure/azure-resource-manager/resource-group-overview" \l "resource-groups).
4. Creare un nuovo **piano di servizio app** facendo clic sulla *sezione servizio app*, **Crea nuovo**e compila **piano di servizio app** (può essere lo stesso nome del servizio app), **posizione**e piano **tariffario** (è disponibile un'opzione gratuita).
5. Fare clic su **database SQL**, quindi scegliere **Crea un nuovo database** o seleziona un database esistente.
6. Nella casella **nome** immettere un nome per il database.
7. Fare clic sulla casella **server di destinazione** , quindi selezionare **Crea un nuovo server**. In alternativa, se in precedenza è stato creato un server, è possibile selezionare il server dall'elenco dei server disponibili.
8. Scegliere la sezione piano **tariffario** , scegliere *gratuito*. Se sono necessarie risorse aggiuntive, il database può essere ridimensionato in qualsiasi momento. Per altre informazioni sui prezzi di Azure SQL, vedere [prezzi del database SQL di Azure](https://azure.microsoft.com/pricing/details/sql-database/managed/).
9. Modificare le [regole di confronto](https://docs.microsoft.com/it-it/sql/relational-databases/collations/collation-and-unicode-support) in base alle esigenze.
10. Immettere un **nome utente amministratore SQL** amministratore e una **password amministratore SQL**.
    * Se è stato selezionato **nuovo server di database SQL**, definire un nuovo nome e una nuova password da utilizzare in un secondo momento quando si accede al database.
    * Se è stato selezionato un server creato in precedenza, immettere le credenziali per tale server.
11. La raccolta di dati di telemetria può essere abilitata per il servizio app usando Application Insights. Con una configurazione minima, Application Insights raccoglie informazioni importanti su eventi, eccezioni, dipendenze, richieste e tracce. Per altre informazioni su Application Insights, vedere [monitoraggio di Azure](https://azure.microsoft.com/services/monitor/).
12. Fare clic su **Crea** nella parte inferiore per indicare che l'operazione è terminata.

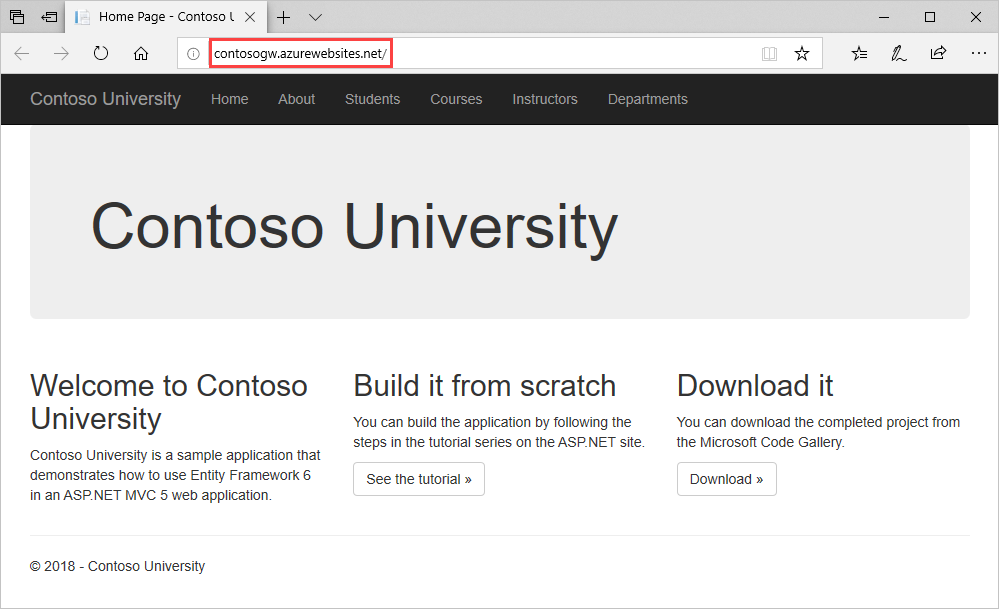
Il portale di gestione torna alla pagina dashboard e l'area **notifiche** nella parte superiore della pagina indica che è in corso la creazione del sito. Dopo un periodo di tempo (in genere meno di un minuto), esiste una notifica che la distribuzione è riuscita. Nella barra di spostamento a sinistra, il nuovo servizio app viene visualizzato nella sezione **Servizi app** e il nuovo database SQL viene visualizzato nella sezione **database SQL** .

**Distribuire l'app in Azure**

1. In Visual Studio fare clic con il pulsante destro del mouse sul progetto in **Esplora soluzioni** e scegliere **Pubblica** dal menu di scelta rapida.
2. Nella pagina selezionare **una destinazione di pubblicazione** scegliere **servizio app** , quindi **selezionare esistente**e quindi scegliere **pubblica**.

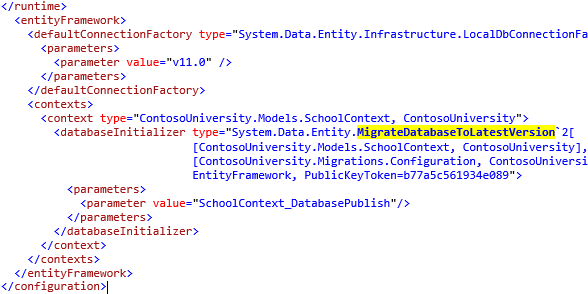


1. Se in precedenza non è stata aggiunta la sottoscrizione di Azure in Visual Studio, eseguire i passaggi sullo schermo. Questa procedura consente a Visual Studio di connettersi alla sottoscrizione di Azure in modo che l'elenco di **Servizi app** includa il sito Web.
2. Nella pagina **servizio app** selezionare la **sottoscrizione** a cui è stato aggiunto il servizio app. In **Visualizza**selezionare **gruppo di risorse**. Espandere il gruppo di risorse a cui è stato aggiunto il servizio app e quindi selezionare il servizio app. Scegliere **OK** per pubblicare l'app.
3. Nella finestra **Output** vengono indicate le azioni di distribuzione effettuate e viene segnalato il corretto completamento della distribuzione.
4. Al completamento della distribuzione, il browser predefinito si apre automaticamente all'URL del sito Web distribuito.

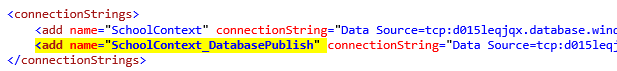


L'app è ora in esecuzione nel cloud.

A questo punto, il database *schoolContext* è stato creato nel database SQL di Azure perché è stata selezionata l'opzione **Esegui migrazioni Code First (esecuzione all'avvio dell'app)**. Il file *Web.config* nel sito Web distribuito è stato modificato in modo che l'inizializzatore [MigrateDatabaseToLatestVersion](https://msdn.microsoft.com/library/hh829476(v=vs.103).aspx) venga eseguito la prima volta che il codice legge o scrive i dati nel database, che si è verificato quando è stata selezionata la scheda **students (studenti** ):



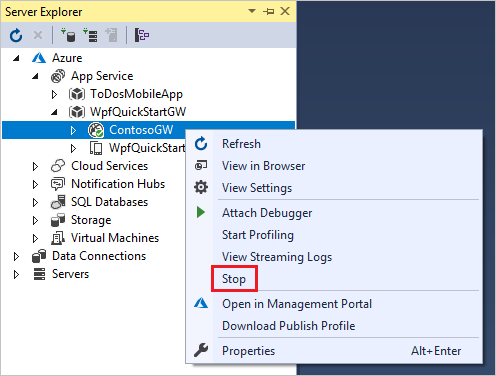
Il processo di distribuzione ha creato anche una nuova stringa di connessione *(schoolContext \_ DatabasePublish*) per migrazioni Code First da usare per l'aggiornamento dello schema del database e il seeding del database.



È possibile trovare la versione distribuita del file di Web.config nel computer in uso *ContosoUniversity\obj\Release\Package\PackageTmp\Web.config*. È possibile accedere al file di *Web.config* distribuito utilizzando FTP. Per istruzioni, vedere [distribuzione Web ASP.NET con Visual Studio: distribuzione di un aggiornamento del codice](https://docs.microsoft.com/it-it/aspnet/web-forms/overview/deployment/visual-studio-web-deployment/deploying-a-code-update). Seguire le istruzioni che iniziano con "per usare uno strumento FTP, sono necessari tre elementi: l'URL FTP, il nome utente e la password".

**Nota**

L'app Web non implementa la sicurezza, pertanto chiunque trovi l'URL può modificare i dati. Per istruzioni su come proteggere il sito Web, vedere [**distribuire un'app ASP.NET MVC sicura con appartenenza, OAuth e database SQL in Azure**](https://docs.microsoft.com/it-it/aspnet/core/security/authorization/secure-data). È possibile impedire ad altri utenti di usare il sito arrestando il servizio usando il portale di gestione o **Esplora server** di Azure in Visual Studio.



**Scenari di migrazioni avanzate**

Se si distribuisce un database eseguendo automaticamente le migrazioni come illustrato in questa esercitazione e si distribuisce in un sito Web in esecuzione su più server, è possibile ottenere più server che tentano di eseguire le migrazioni nello stesso momento. Poiché le migrazioni sono atomiche, se due server tentano di eseguire la stessa migrazione, ne verrà eseguita una e l'altra avrà esito negativo (presupponendo che le operazioni non possano essere eseguite due volte). In questo scenario, se si desidera evitare questi problemi, è possibile chiamare le migrazioni manualmente e configurare il proprio codice in modo che venga eseguito una sola volta. Per ulteriori informazioni, vedere [esecuzione e creazione di script delle migrazioni dal codice](http://romiller.com/2012/02/09/running-scripting-migrations-from-code/) nel Blog di Rowan Miller e [Migrate.exe](https://docs.microsoft.com/it-it/ef/ef6/modeling/code-first/migrations/migrate-exe) (per l'esecuzione di migrazioni dalla riga di comando).

Per informazioni su altri scenari di migrazione, vedere la [serie di screencast sulle migrazioni](https://blogs.msdn.com/b/adonet/archive/2014/03/12/migrations-screencast-series.aspx).

**Aggiorna migrazione specifica**

update-database -target MigrationName

Il update-database -target MigrationName comando esegue la migrazione di destinazione.

**Ignora le modifiche alla migrazione del database**

Add-migration MigrationName -ignoreChanges

ignoreChangesCrea una migrazione vuota con il modello corrente come snapshot.

**Inizializzatori di Code First**

Nella sezione relativa alla distribuzione è stato visualizzato l'inizializzatore [MigrateDatabaseToLatestVersion](https://msdn.microsoft.com/library/hh829476(v=vs.103).aspx) usato. Code First fornisce anche altri inizializzatori, tra cui [CreateDatabaseIfNotExists](https://msdn.microsoft.com/library/gg679221(v=vs.103).aspx) (impostazione predefinita), [DropCreateDatabaseIfModelChanges](https://msdn.microsoft.com/library/gg679604(v=VS.103).aspx) (usato in precedenza) e [DropCreateDatabaseAlways](https://msdn.microsoft.com/library/gg679506(v=VS.103).aspx). L' DropCreateAlways inizializzatore può essere utile per configurare le condizioni per gli unit test. È anche possibile scrivere inizializzatori personalizzati ed è possibile chiamare un inizializzatore in modo esplicito se non si vuole attendere che l'applicazione legga o scriva nel database.

Per ulteriori informazioni sugli inizializzatori, vedere [Understanding Database Initializers in Entity Framework Code First](http://www.codeguru.com/csharp/article.php/c19999/Understanding-Database-Initializers-in-Entity-Framework-Code-First.htm) e il capitolo 6 del libro [Programming Entity Framework: Code First](http://shop.oreilly.com/product/0636920022220.do) di Julie Lerman e Rowan Miller.

**Ottenere il codice**

[Scaricare il progetto completato](https://webpifeed.blob.core.windows.net/webpifeed/Partners/ASP.NET MVC Application Using Entity Framework Code First.zip)

**Risorse aggiuntive**

Collegamenti ad altre risorse Entity Framework sono disponibili in [ASP.NET Data Access-risorse consigliate](https://docs.microsoft.com/it-it/aspnet/whitepapers/aspnet-data-access-content-map).

**Passaggi successivi**

In questa esercitazione:

* Migrazioni Code First abilitate
* L'app è stata distribuita in Azure (facoltativo)

Passare all'articolo successivo per informazioni su come creare un modello di dati più complesso per un'applicazione MVC ASP.NET.