Configurazione EF: Code-First

Configurazione del progetto per utilizzare un database con l'approccio Code-First.

Progetto di Informatica classe 5^{a}

Anno 2013/2014

Ambiente: .NET 4.5/C# 5.0

Entity Framework: 6.0

Indice generale

1 Introduzione.....3

1.1 Installazione EF 6.0.....3

1.2 Namespace.....3

1.3 Domain model utilizzato.....4

2 Dal domain model al database.....5

2.1 Modifica del modello: impostazione predefinita.....6

2.1.1 Modificare la “strategia” di creazione del database.....6

2.1.2 Utilizzare “DropCreateDatabaseIf...” come strategia predefinita.....6

3 Dal domain model al database – file attached.....7

3.1 Uso dell'alias “DataDirectory”.....7

3.2 Personalizzare il nome della stringa di connessione.....8

4 Accesso ad un database preesistente.....9

4.1 Accesso predefinito (CreateDatabaseIfNotExist).....9

4.2 Inizializzazione mediante “CreateDatabaseAlways”.....9

4.3 Inizializzazione mediante “CreateDatabaseIfModelChanges”.....9

4.4 Conclusioni.....9

5 “Migrazione” del modello verso un database esistente.....10

5.1 Abilitare la migrazione.....10

5.2 Aggiungere una migrazione vuota.....10

5.3 Migrazione manuale.....11

5.4 Migrazione automatica.....11

5.5 Problemi legati alla migrazione.....11

6 Note personali.....13

1 Introduzione

Questo tutorial affronta alcune problematiche legate alla configurazione di applicazioni che usano Entity Framework 6.0 + Code-First, sia con database SQL Server che SQLite.

Per quanto riguarda SQL Server, si presuppone l'utilizzo di LocalDB, l'istanza di SQL Server installata automaticamente con Visual Studio.

Code-First e SQL Server Express

Se sul sistema è installato SQL Server Express sarà questo ad essere utilizzato per ospitare i database creati.

Nel tutorial si continuerà a fare riferimento a LocalDB, poiché non necessita di alcuna installazione.

Si consiglia comunque di installare SQL Server Management Studio Express:

"http://www.microsoft.com/it-it/download/details.aspx?id=29062"

Saranno affrontati i seguenti scenari:

1. dal domain model al database
2. dal domain model al database; file di database *attached*.
3. accesso ad un database esistente.
4. modifica del domain model, considerando i vari scenari

1.1 Installazione EF 6.0

Aprire il menù di scelta rapida sulla *solution* (o sul progetto) nel Solution Explorer e cliccare sulla voce: "**Menage NuGet Packages for Solution...**".

Sulla successiva finestra cliccare su "**Entity Framework**" e installare.

1.2 Namespace

Le funzionalità di EF sono suddivise su vari namespace, tra i quali:

1. **System.Data.Entity**: unico necessario per le applicazioni C-F più semplici.
2. **System.ComponentModel.DataAnnotations**: definisce l'annotazione **[Key]**
3. **System.ComponentModel.DataAnnotations.Schema**: definisce le annotazioni **[Table]**, **[Column]** e altre, utilizzate per configurare il mapping tra *domain model* e datatabse.
4. **System.Data.Entity.ModelConfiguration**: definisce la "**Code First Fluent API**", utilizzabile per configurare il mapping tra *domain model* e database.

1.3 Domain model utilizzato

Nei vari scenari sarà utilizzato prevalentemente il seguente modello:

```
using System.Data.Entity;

namespace EFCodeFirstConfig
{
    public class Library: DbContext
    {
        public DbSet<Book> Books { get; set; }
    }

    public class Book
    {
        public int BookId { get; set; }
        public string Title { get; set; }
    }
}
```

2 Dal domain model al database

In questo scenario si lascia a EF la responsabilità di creare il database alla prima esecuzione del programma.

Per impostazione predefinita, EF:

1. crea un database su LocalDB e colloca il file nella cartella "**C:\users\<utente>**".
2. Denomina il database e il file secondo il pattern:

   <namespace>.<classe context>

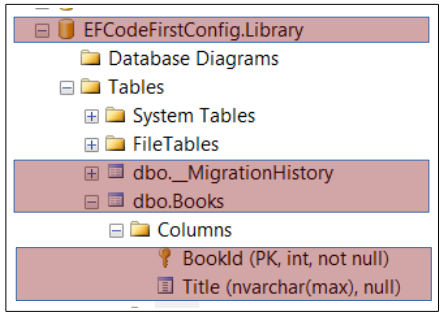
Ad esempio, l'esecuzione del seguente codice:

```
static void Main(string[] args)
{
    TestNewBook("Il gioco di Ender");
}

static void TestNewBook(string bookTitle)
{
    using (Library lib = new Library())
    {
        Book b = new Book { Title = bookTitle };
        lib.Books.Add(b);
        lib.SaveChanges();
    }
}
```

Crea i seguenti file, collocati in "c:\users\<utente>":

E crea un nuovo database in LocalDB:



Vi sono alcune considerazioni da fare:

1. E' stata creata una tabella di nome "Books". Il nome corrisponde al nome pluralizzato della classe Book.
2. La tabella ha una PK di nome BookId (di tipo identity)

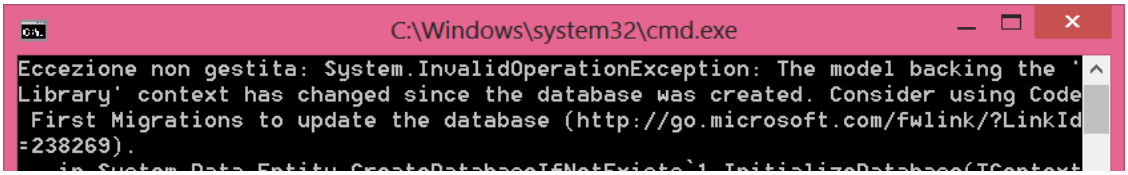
3. E' stata creata una tabella di nome "**__MigrationHistory**". Questa contiene i metadati che descrivono il modello e sarà utilizzata nelle successive esecuzioni per stabilire se il modello è cambiato rispetto allo schema del database.

2.1 Modifica del modello: impostazione predefinita

Una volta che il database è stato creato, C-F non ammette nessuna modifica al modello. Ad esempio, l'inserimento del campo "Author" in Book:

```
public class Book
{
    public int BookId { get; set; }
    public string Title { get; set; }
    public string Author { get; set; }
}
```

e l'esecuzione del programma produce il seguente errore:



2.1.1 Modificare la "strategia" di creazione del database

Se vogliamo che il database sia sincronizzato automaticamente con il modello, un'opzione è l'esecuzione del metodo **SetInitializer()** della classe statica **Database**.

Questo consente d'impostare la strategia di creazione del database. Quella predefinita è "**CreateDatabaseIfNotExist**". Per consentire l'aggiornamento del database occorre modificarla in: "**DropCreateDatabaseIfModelChanges**".

Il metodo deve essere eseguito prima che EF acceda al database. Ad esempio, la chiamata può essere collocata nel costruttore della classe context:

```
public class Library: DbContext
{
    public Library()
    {
        Database.SetInitializer(new DropCreateDatabaseIfModelChanges<Library>());
    }
    public DbSet<Book> Books { get; set; }
}
```

Nota bene: il metodo verifica il cambiamento del modello e, in caso affermativo, *elimina e ricrea l'intero database*. In sostanza: tutti i dati sono perduti.

2.1.2 Utilizzare "DropCreateDatabaseIf..." come strategia predefinita

E' possibile impostare questa strategia fin dall'inizio. Alla prima esecuzione, il database viene creato. Successivamente viene modificato in accordo con i cambiamenti del modello.

3 Dal domain model al database – file *attached*

Questo scenario implica la creazione di un file di database collocato nel percorso desiderato.

A questo scopo è necessario indicare la stringa di connessione che referenzia il file di database da creare e lo si può fare in **App.Config**.

Nel file App.Config si crea (se non esiste) la sezione <connectionString> e si aggiunge una nuova stringa. Ad esempio:

```
<connectionStrings>
    <add name="EFCodeFirstConfig.Library"
         connectionString="Data Source=(LocalDB)\v11.0;
                           AttachDbFilename=C:\...\Library.mdf;
                           Initial Catalog=Library"
                           Integrated Security=true"
         providerName="System.Data.SqlClient" />
</connectionStrings>
```

I parametri evidenziati sono:

1. Il nome della stringa di connessione. Se coincide con il nome della classe context (comprensivo di namespace), EF utilizzerà automaticamente la stringa di connessione per accedere al database.
2. Il percorso del file di database che si desidera creare.
3. Il nome del database. Il parametro è opzionale; in caso non venga specificato, LocalDB userà il nome del file.

3.1 Uso dell'alias “DataDirectory”

Per il file è possibile utilizzare un percorso relativo mediante l'alias “|DataDirectory|”. Ad esempio:

```
<connectionStrings>
    <add name="EFCodeFirstConfig.Library"
         connectionString="Data Source=(LocalDB)\v11.0;
                           AttachDbFilename=|DataDirectory|\Library.mdf;
                           Initial Catalog=Library"
                           Integrated Security=true"
         providerName="System.Data.SqlClient" />
  </connectionStrings>
```

Nelle applicazioni Web, all'avvio del programma DataDirectory viene sostituito con la cartella “App_Data”. Nelle Applicazioni Console e Windows, “DataDirectory” non è preimpostato; è possibile farlo manualmente eseguendo:

```
AppDomain.CurrentDomain.SetData("DataDirectory",
AppDomain.CurrentDomain.BaseDirectory);
```

che imposta DataDirectory sulla cartella contenente l'eseguibile.

3.2 Personalizzare il nome della stringa di connessione

Il nome dato alla stringa di connessione in App.Config non deve per forza rispettare il pattern <namespace>.<classe context>. Ad esempio:

```
<connectionStrings>
    <add name="Library"
         connectionString="..." providerName="System.Data.SqlClient" />
  </connectionStrings>
```

In questo caso, però, è necessario indicare alla classe context il nome della connessione. Un modo per parlo è chiamare il costruttore della classe base specificando il nome dato alla connessione.

```
public class Library: DbContext
{
    public Library(): base ("Library")
    {}
    public DbSet<Book> Books { get; set; }
}
```

4 Accesso ad un database preesistente

Per database preesistente si intende un database creato a design time (con Visual Studio o altro software), non contenente i metadati che descrivono il *domain model*.

In questo caso, la strategia di utilizzo del database cambia rispetto agli scenari precedenti.

4.1 Impostare la stringa di connessione

L'accesso ad un database esistente richiede di impostare una stringa di connessione. Questa può indicare solo il nome del database, solo il file di database (database *attached*), oppure entrambi. Alcuni esempi:

```
connectionString="Data Source=(LocalDB)\v11.0; Initial Catalog=Library;
                  Integrated Security=true"
```

```
connectionString="Data Source=(LocalDB)\v11.0; AttachDbFilename=F:\Data\Library.mdf;
                  Integrated Security=true"
```

```
connectionString="Data Source=(LocalDB)\v11.0; Initial Catalog=Library;
                  AttachDbFilename=|DataDirectory|\Library.mdf;
                  Integrated Security=true"
```

4.2 Accesso predefinito (CreateDatabaseIfNotExist)

Rappresenta la strategia di default e implica il mapping del *domain model* con lo schema del database. E' responsabilità del programmatore garantire che i due modelli siano sincronizzati.

Ad esempio, se la tabella Books definisce le colonne BookId e Title, ma la classe Book aggiunge a queste la colonna Author, l'inserimento di un nuovo libro determina un errore del tipo "*Invalid column Author*".

4.2.1 Evitare la creazione accidentale del database

Nell'approccio predefinito esiste un problema potenziale: se, per errore, la stringa di connessione fa riferimento ad un database inesistente, EF ne crea uno compatibile con il *domain model*. Ovviamente non è ciò che si vuole.

Per evitarlo è opportuno impostare una strategia di inizializzazione nulla. Ad esempio:

```
public class Library: DbContext
{
    public Library()
    {
        Database.SetInitializer<Library>(null);
    }
    public DbSet<Book> Books { get; set; }
}
```

In questo modo, se la stringa di connessione non referenzia un database esistente, viene sollevata un'eccezione.

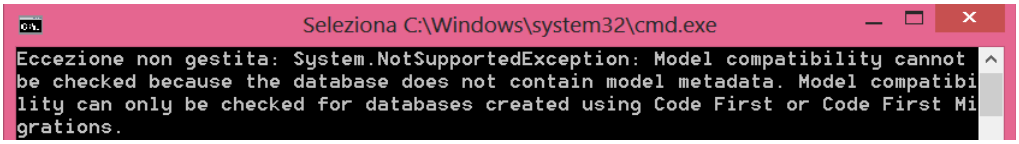
4.3 Inizializzazione mediante “CreateDatabaseAlways”

Il database viene eliminato e ricreato (purché non sia attualmente in uso). Ovviamente non si tratta di una strategia sensata in questo scenario.

4.4 Inizializzazione mediante “CreateDatabaseIfModelChanges”

Questa opzione non è utilizzabile, poiché EF non può confrontare il *domain model* e lo schema del database, non esistendo in quest'ultimo la tabella **__MigrationHistory**.

Viene prodotto l'errore:



4.5 Conclusioni

In questo scenario la strategia più ovvia è la prima.

Il problema sorge se si desidera accedere ad un database preesistente, ma si vuole comunque sfruttare la funzionalità di EF Code-First di sincronizzare lo schema del database con i cambiamenti al *domain model*.

5 “Migrazione” del modello verso un database esistente

Ricapitoliamo: abbiamo un database preesistente, contenente già le tabelle (ed eventualmente dei dati), ma privo della tabella “__MigrationHistory”, che descrive il *domain model*.

In questa situazione, possiamo utilizzare il database con C-F, purché il *domain model* coincida con il suo schema. Ciò che desideriamo è poter modificare il *domain model* e far sì che EF aggiorni lo schema del database; esattamente come avviene quando è EF a crearlo.

In questo caso è necessario sfruttare la funzionalità di “Migrazione” offerta da EF.

5.1 Abilitare la migrazione

Innanzitutto è necessario aprire il “**Package Manager Console**” dal menù “**Tools | Library Package Manager**”

Dopodiché si abilità la funzionalità di migrazione con il comando “**enable-migrations**”.

Package Manager Console

```
PM>enable-migrations
```

VS crea la cartella “Migrations”, nella quale colloca la classe Configuration:

```
namespace EFCodeFirstConfig.Migrations
{
...

    class Configuration: DbMigrationsConfiguration<EFCodeFirstConfig.Library>
    {
        public Configuration()
        {
            AutomaticMigrationsEnabled = true;  // il valore di default è false
        }

        protected override void Seed(EFCodeFirstConfig.Library context)
        {
        }
    }
}
```

Per abilitare la migrazione automatica (eseguita all'avvio del programma) è necessario assegnare true alla variabile **AutomaticMigrationsEnabled**.

Questa procedura sarebbe (quasi) sufficiente se il database fosse vuoto, ma non se contiene già delle tabelle, che è lo scenario considerato.

5.2 Aggiungere una migrazione vuota

Prima di procedere è necessario aggiungere il codice e i metadati richiesti per eseguire la prima migrazione: l'obiettivo è la creazione della tabella “__MigrationHistory”, con la descrizione del modello esistente.

Sulla PMC eseguire il comando "**Add-Migration InitialMigration -IgnoreChanges**"

Package Manager Console

```
PM>Add-Migration InitialMigration -IgnoreChanges
```

VS aggiunge delle classi e delle risorse alla cartella Migrations.

5.3 Migrazione manuale

Le informazioni necessarie per la migrazione sono pronte; è possibile eseguirla manualmente dalla PMC con il comando "**update-database**":

Package Manager Console

```
PM>update-database
```

5.4 Migrazione automatica

E' possibile configurare la classe context in modo che la migrazione venga eseguita automaticamente all'avvio del programma.

Si usa il metodo Initializer della classe Database, specificando come strategia la classe "**MigrateDatabaseToLatestVersion<,>**":

```
public class Library: DbContext
{
    public  Library()
    {
        Database.SetInitializer(new MigrateDatabaseToLatestVersion<Library,
                                                    Migrations.Configuration>());
    }
    public DbSet<Book> Books { get; set; }
}
```

Nota bene: **MigrateDatabaseToLatestVersion<,>** richiede come tipi la classe context e la classe Configuration (comprensiva di namespace).

5.5 Problemi legati alla migrazione

La migrazione verso un database esistente pone due problemi fondamentali:

1. La modifica indesiderata della struttura del database.
2. La perdita indesiderata dei dati.

Il punto 1, nello scenario che stiamo considerando è quello più problematico. Infatti, nell'eseguire la migrazione, EF sincronizza lo schema del database con il *domain model*. Se, ad esempio, la classe Book non definisce tutti i campi definiti dalla tabella Books, EF tenta di eliminare le colonne "di troppo" rispetto al modello. (E fallisce.)

In sostanza, perché la migrazione funzioni è necessario che il modello sia una copia (o un sovrainsieme) dello schema del database.

6 Note personali

(vedi link: “http://msdn.microsoft.com/en-us/data/jj591621.aspx”)