https://medium.com/better-programming/beginners-guide-to-entity-framework-d862c9aaec4

Beginner's Guide to Entity Framework

How to connect your C# application to a database and perform basic CRUD operations

 Adam Snyder

Aug 28 ·

With some C# knowledge under your belt, you might be wondering: “How do I expand on what I know?”

Perhaps, you have an idea for your application that would benefit from having a database. Entity Framework is an excellent tool to have at your disposal. You will be able to perform CRUD operations via your application.

Not only will this broaden the capabilities of your application but also your own capabilities.

I’ll introduce you to what Entity Framework is, how to set it up in a Winforms application, and give you a solid foundation to continue learning more.

What is Entity Framework?

Entity Framework is an open-source ORM (Object Relational Mapper), supported by Microsoft, for .NET applications.

Basically, this is a tool that will make it easier to make the connections to your objects in your code to the information in your relational database.

Entity Framework will take care of creating the connection to your database, creating the objects you can use to manipulate the database, and give you methods to work with your data.

You will be able to manipulate objects, in your code, that represent data in your database and will allow you to save those changes to the database. Think of this as an interface, or window, to your database that you can add simple functionality to, for anyone to use through your application.

Code First and Database First

There are two different ways to approach the use of Entity Framework in your application. These different approaches have their own benefits and it's up to you to decide which one works best for your needs.

Code first

Code first focuses on creating a database that doesn't yet exist though the design of your classes. This allows the possibility if your classes' change to also change the configuration in your database.

This is a popular approach as it allows for more control through the code instead of restrictions placed by your database.

Database first

Database first focuses on working with an existing database. You create your database first and design your tables/relationship.

This approach will generate the classes for you that align with the structure you created within the database. This allows you to modify a database manually and then update the classes that correlate automatically.

Getting Started

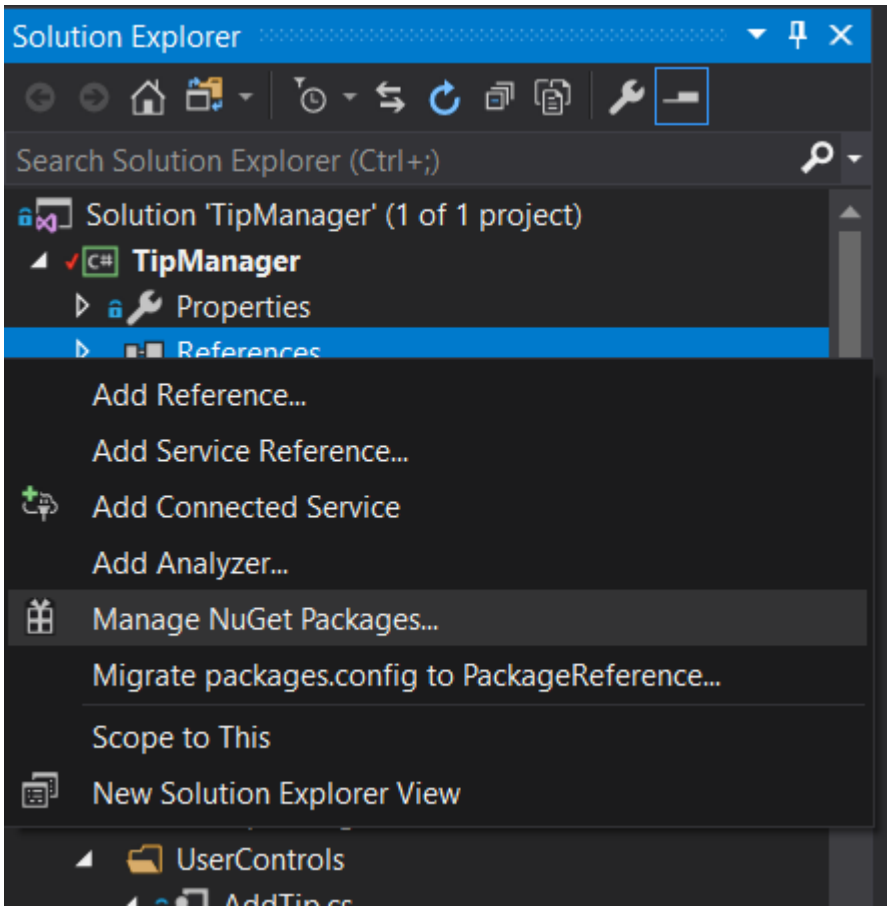
For this example, I'm going to use EF6 (Entity Framework 6) in C# Winforms, incorporating the database-first approach.

I'll also use a database created with SSMS (SQL Server Management Studio) and Microsoft SQL Server 2017.

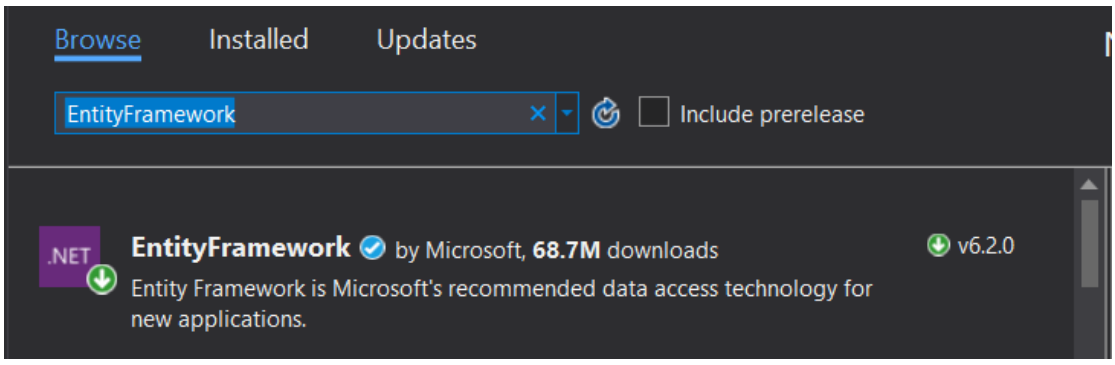
For more information on these topics, you can read this guide. This video, *Creating a Database with Tables and Relationships(MS SQL),* can help you set up a database if you need a bit of a refresher. I'm not going to dive too deep into any of that information but I wanted to provide it, just in case.

Once you have a database created, go into your Winforms application and follow these steps:

- Right-click on *References* in the *Solution Explorer* and select *Manage NuGet Packages…*



- Search for *EntityFramework* by Microsoft, click on it, and click *Install*.

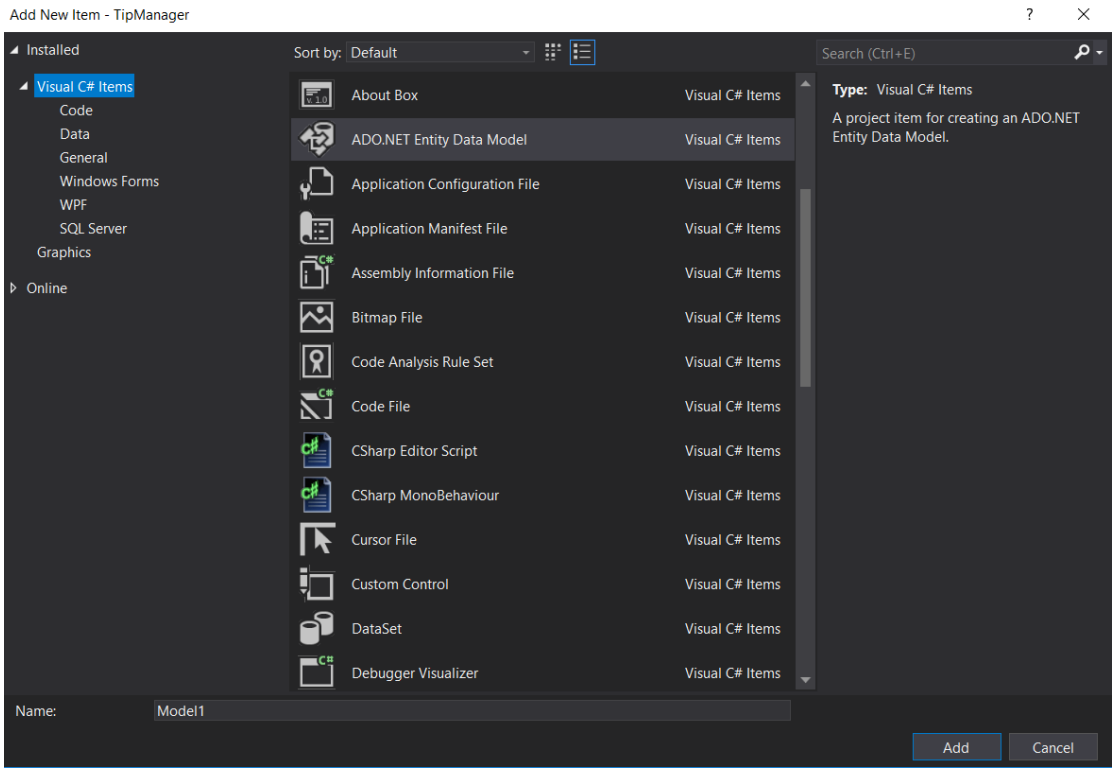


- Now you should have Entity Framework installed and you can verify it by clicking the drop-down icon in *References* — you should see *EntityFramework* listed.

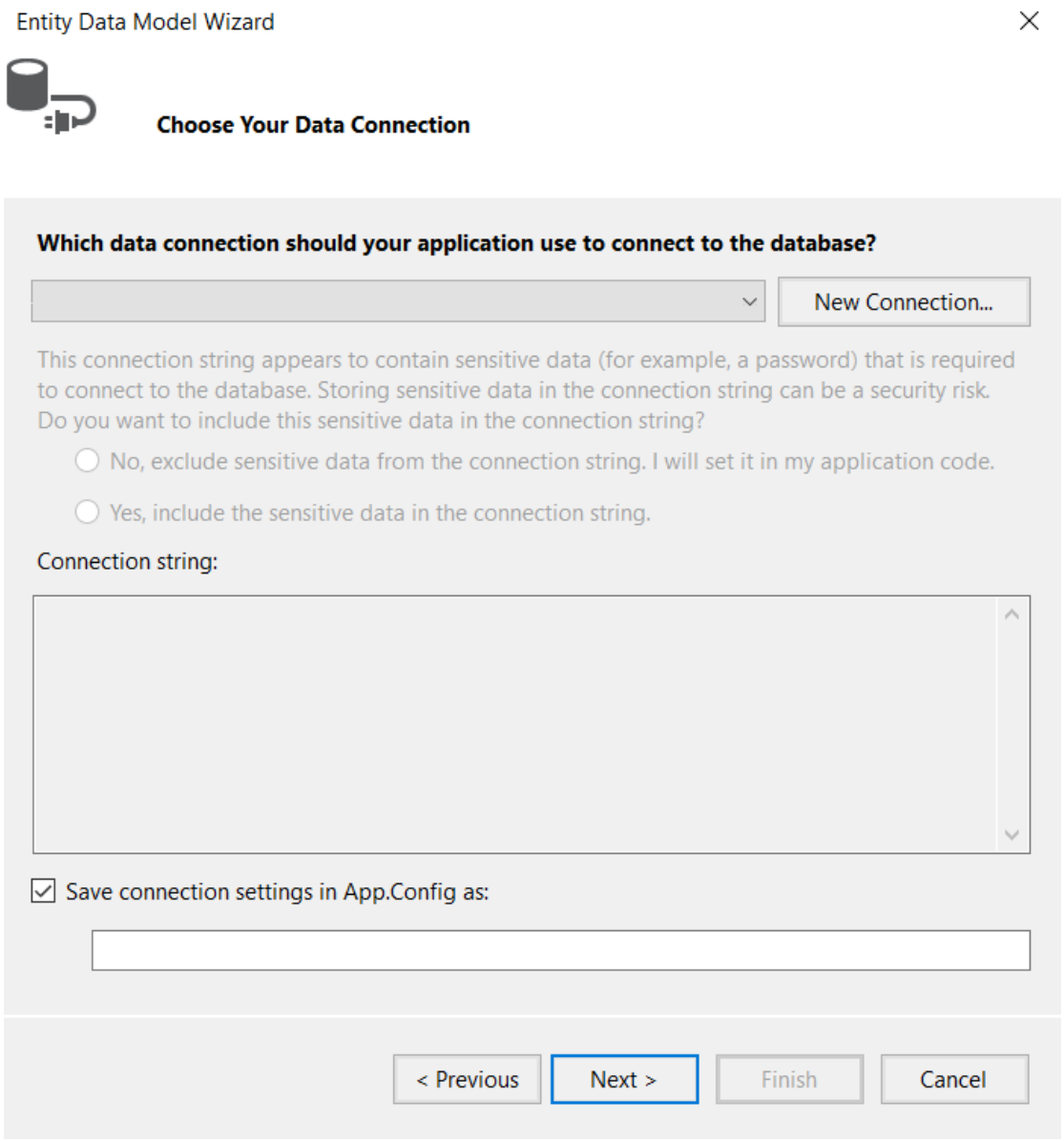
Creating an Entity Data Model

Now that you have your database all set up and you have EF6 installed in your project, you need to set up the connection for your project to the database and let Entity Framework create the Entity Data Models (classes) that correlate to your database.

- Right-click on your project in the *Solution Explorer* and select *Add* → *New Item*. Then, you will want to find and select *ADO.NET Entity Data Model.*



- Select *EF Designer from Database*.
- Click on *New Connection…*

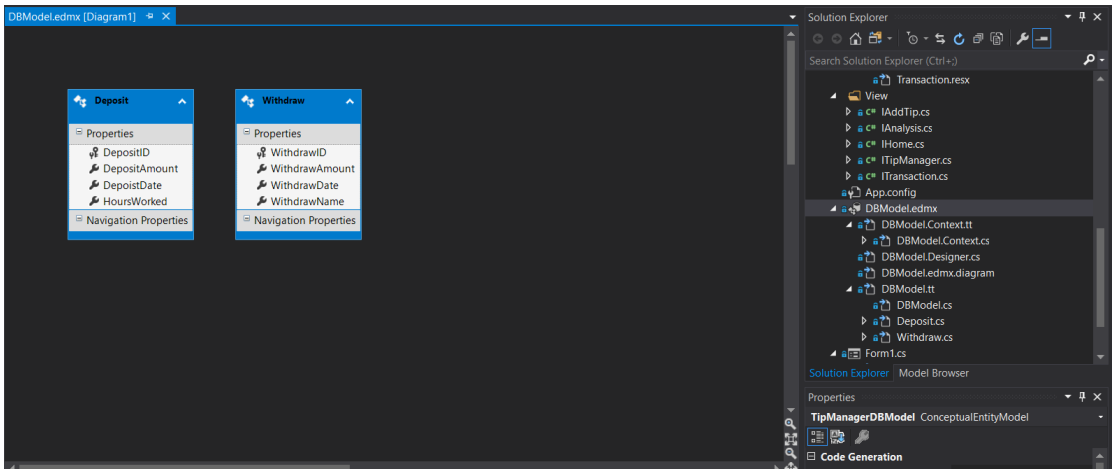


- In the *Properties* pop-up, provide the server name (this can be found in SSMS) and enter the name of your database. Press *OK*.
- Make sure to check the checkbox *Save connection settings in App.Config as:* and give it an appropriate name, such as *MyProjectDBEntities*, and press *Next*.
- Displayed now will be Tables, Views, Stored Procedures, and Functions. Select everything you want, keep the default checkboxes selected, and click *Finish*.

You've now successfully linked your database to your project and created entity data models to use in relation to your database.

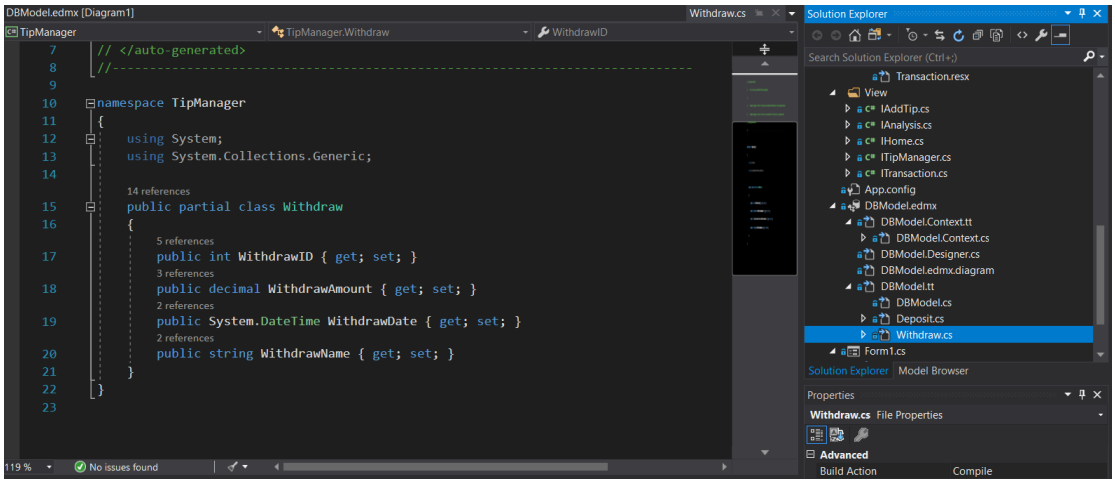
In your Solution Explorer, you will see an .edmx file. If you click on that file you will open the EDM (Entity Data Model) designer.

This will display all the entities for your selected tables from your database and their relationships. (Note: There are no relationships between the tables in the example below.)



Also, if you expand the .edmx file, you will see a file named something similar to Model.tt.

Expand the .tt file and it will reveal the classes created from the tables you imported. These classes are what you will use as a model when manipulating your database.



DBContext

You have your database set up to your project and you have created your Entity Data Model, now it's time to learn how to work with your model and Entity Framework.

To perform your basic CRUD operations (Create, Read, Update, and Delete), you will need to work with the DBContext class.

The DBContext class in association with your model will allow you to write and run queries, track any changes made to these objects, save changes you've made to these objects, and reflect those changes within the database, and bind objects to UI controls.

DBContext is generated for you and can be found under DBModel.Context.tt in your Solution Explorer. You will notice code similar to this:

```
public virtual DbSet<Deposit> Deposits { get; set; }
public virtual DbSet<Withdraw> Withdraws { get; set; }
```

The DBSet class derives from DBContext and represents an entity set that is used for CRUD operations. They are added as properties within DBContext and are mapped to your database tables.

CRUD Operations

Now, you should have a working understanding of Entity Framework.

It's now time to learn how to work with DBContext and the modules you've created to perform basic CRUD operations. It's common practice to create a new instance of your context by use of using, as you will see below.

This ensures all the resources that the context controls will be disposed of at the end of the block.

The examples below are from my own personal project. They are by no means perfect and are still in need of refactoring but I believe it would be good to show some real-world examples, as opposed to just random examples made for this purpose.

Create（新增）

This is a simple method to create a new deposit within the database. The methods accept a Deposit object, created from the Entity Data Model made earlier as an argument.

```
public void AddNewDeposit(Deposit deposit)
{
        using (var context = new TipManagerDBEntities())
        {
                context.Deposits.Add(deposit);
                context.SaveChanges();
        }
}
```

Inside the method, a new DBContext, inside a using block, is created.

Then, the method context.Deposits.Add() is called and the Deposit object is passed into the parameter. Finally, for the changes to take effect, context.SaveChanges() must be called.

Read（查詢、讀取）

This method returns the entire sum of all deposits made.

```
public double GetTotalMade()
{
    using (var context = new TipManagerDBEntities())
    {
        double sum = (double)context.Deposits.Sum(d =>
d.DepositAmount);
        return sum;
    }
}
```

Once again, a new DBContext in a using block is created. Inside the block, a variable to hold the sum is made and the method context.Deposits.Sum() is called and saved in that variable.

Inside of the method parameters, a lambda is used to go through each DepositAmount and add them to the sum. Finally, the method returns the sum for the client.

Update

This method accepts a Deposit object that is known to exist already within the database.

```
public void UpdateDeposit(Deposit deposit)
{
    using (var context = new TipManagerDBEntities())
    {
        context.Entry(deposit).State = EntityState.Modified;
        context.SaveChanges();
    }
}
```

By setting EntityState.Modified to context.Entry(deposit).State, you are essentially attaching the deposit to the context and setting its state to modified.

Then, you call context.SaveChanges() to finalize the changes.

Delete

This method accepts a Deposit object then deletes it from the database. Inside of the using block, it saves the entry of the object, which provides tracking details and operations for the entity, to the variable entry.

```
public void DeleteDeposit(Deposit deposit)
{
    using (var context = new TipManagerDBEntities())
    {
        var entry = context.Entry(deposit);

        if (entry.State == EntityState.Detached)
        {
            context.Deposits.Attach(deposit);
        }

        context.Deposits.Remove(deposit);
        context.SaveChanges();
    }
}
```

Then, it checks to make sure the state is attached first. If not, it will attach the entity. It then calls context.Deposits.Remove() while passing in the object into the parameters. Finally, the changes are saved.

If you would like to see a list of the methods available to you while using DBContext, you can check them out in the Microsoft docs.

Conclusion

You are now all set to incorporate Entity Framework into your next project.

Although this is a beginners guide, you now know how to install the framework, connect to your local database, use basic CRUD operations, and you've informed yourself of the core concepts relating to Entity Framework.

There is a lot more to learn about this subject but, at the very least, you've gotten your feet wet with the basics and can dive into this extremely useful tool.

Adam Snyder

https://medium.com/better-programming/beginners-guide-to-entity-framework-d862c9aaec4