Workflows

Database First

- We design our tables
- EF generates domain classes

Code First

- We create our domain classes
- EF generates database tables

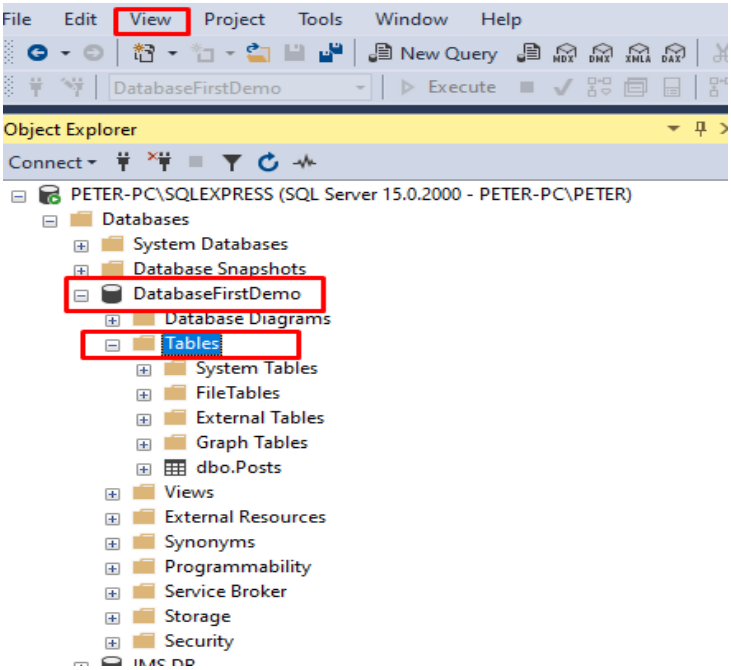
Model First

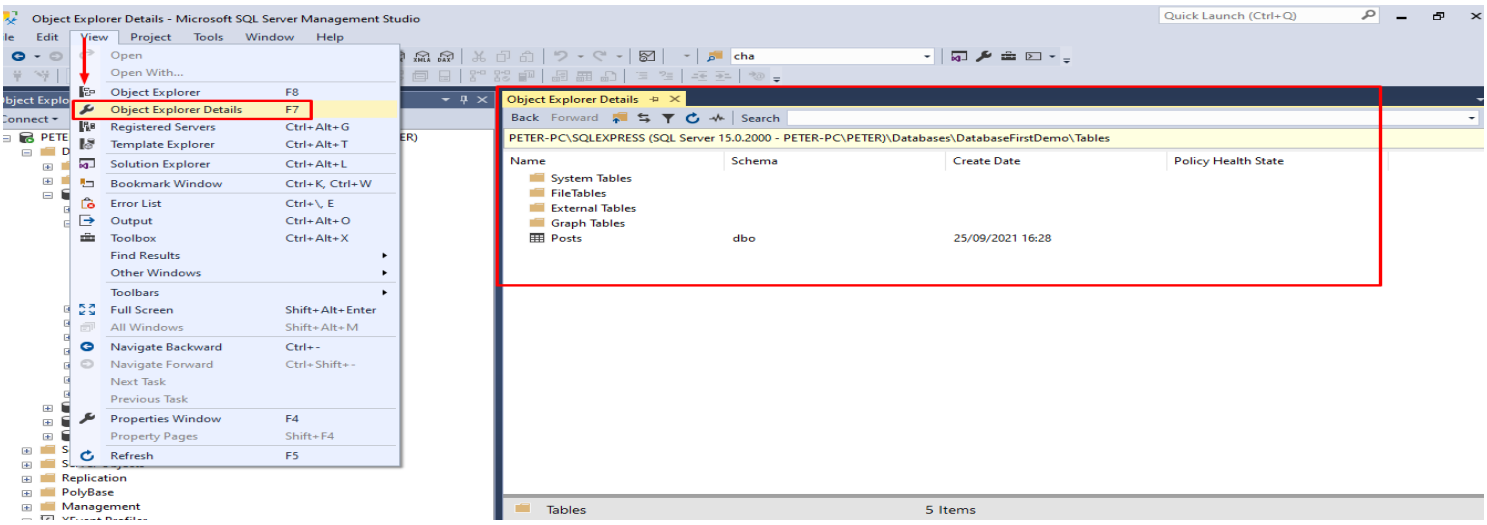
- We create a UML diagram
- EF generates domain classes and database

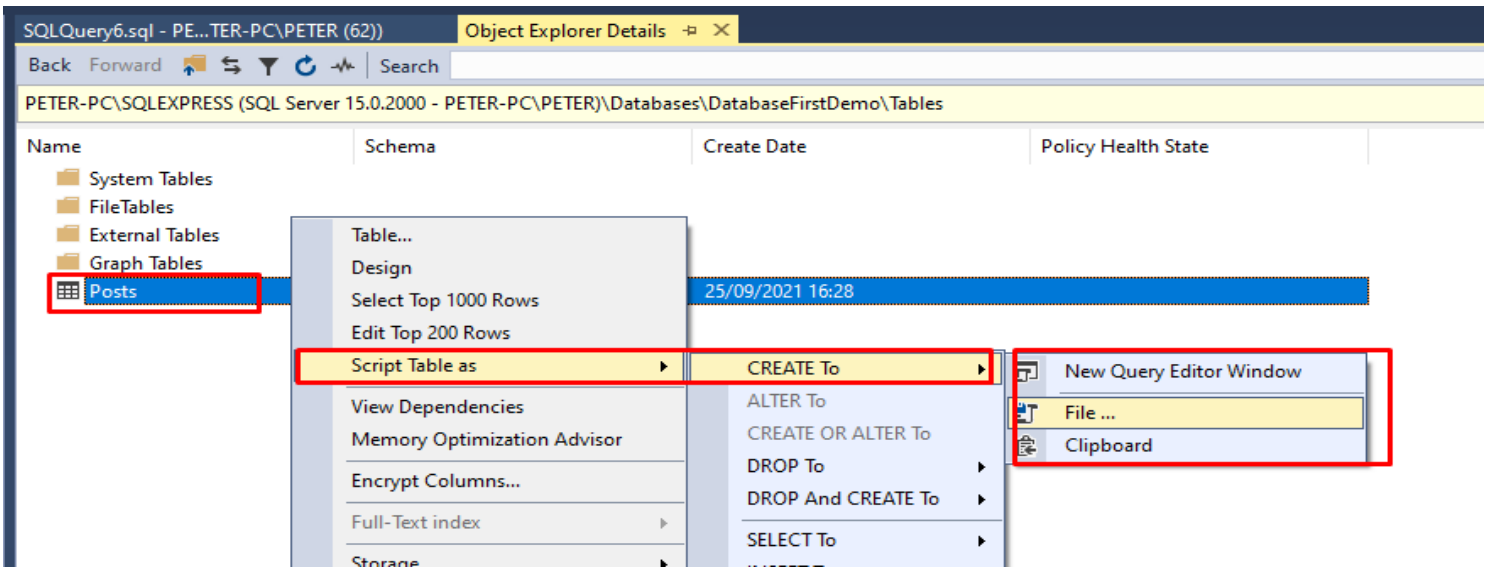
Copy DB

Create scripts for DB objects

Run scripts in a different server

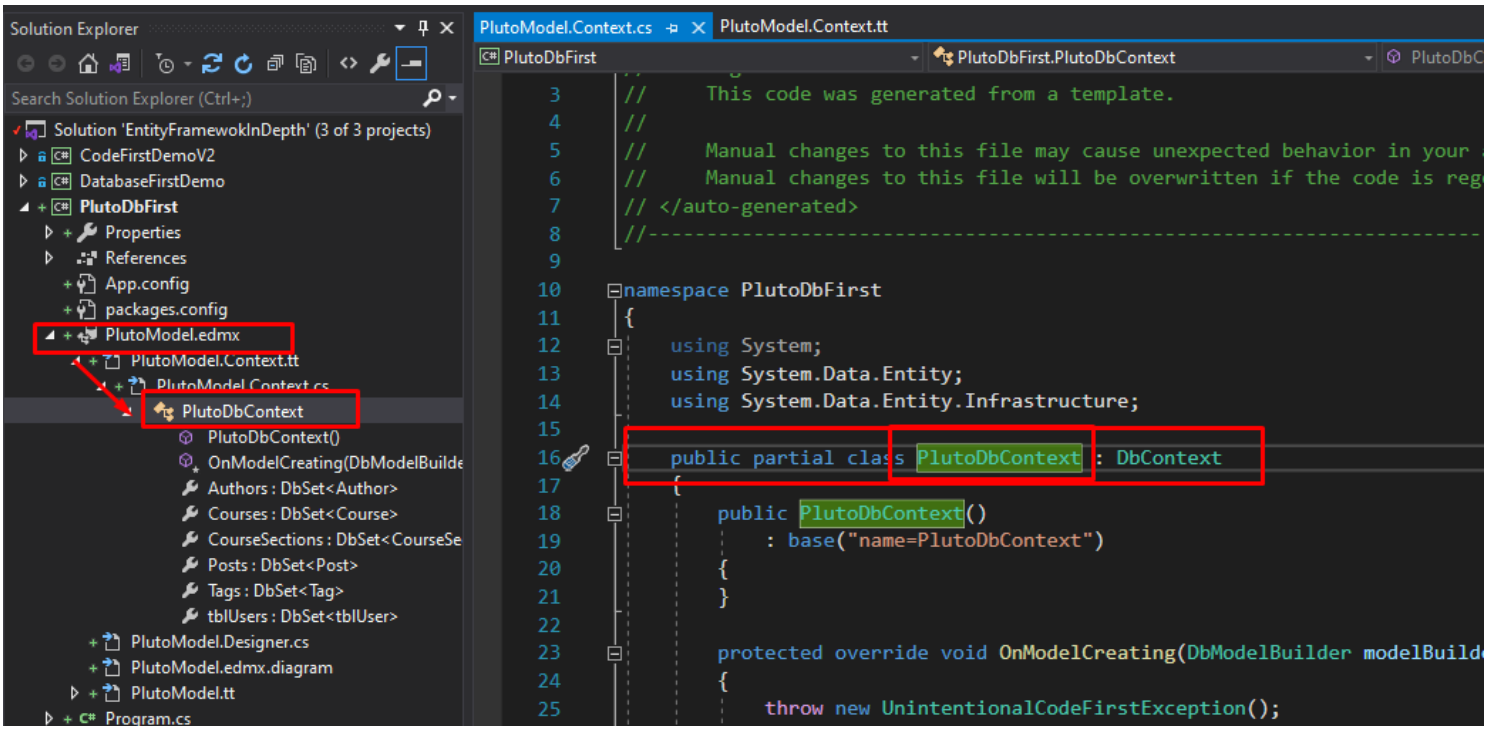


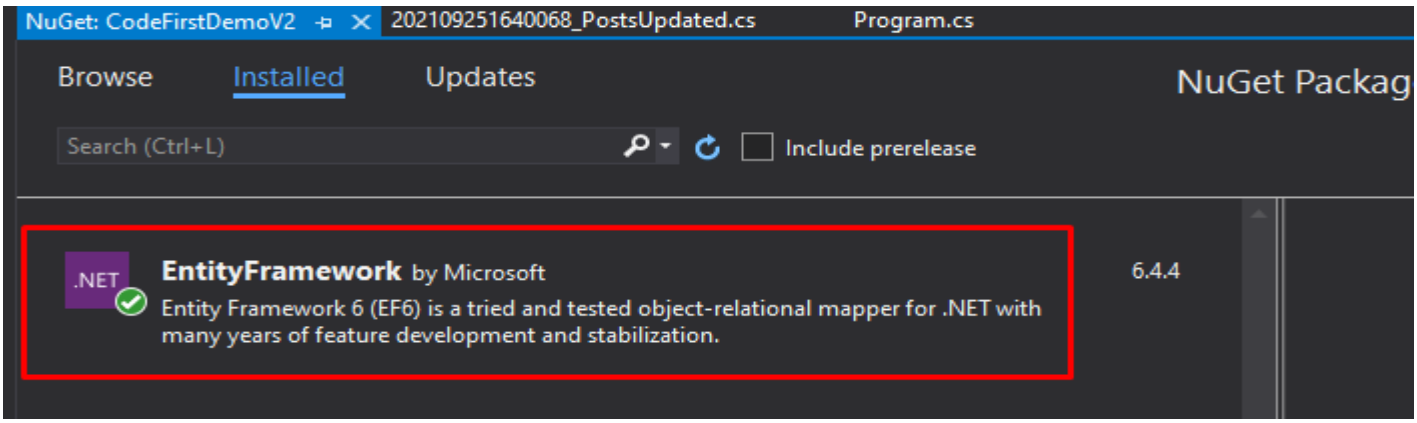


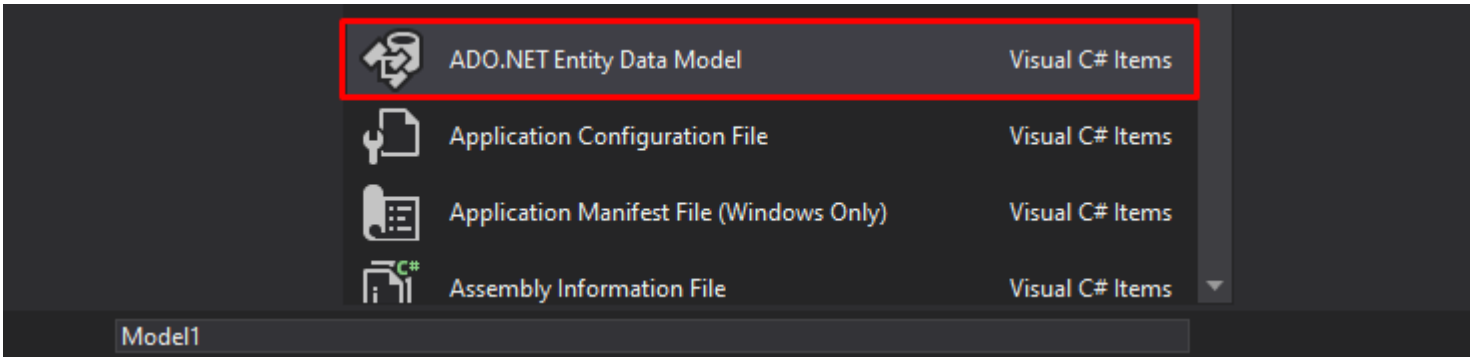


Building model - Database first

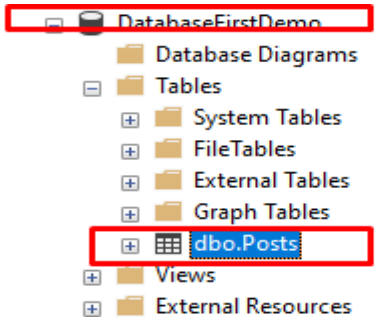
DB name is in the .Context.cs file
Create SQL DB, including table
Add package : EntityFramework by Microsoft
Add “ADO.NET Entity Data Model”
Initialize context and use it to change DB
Verify naming, plurals and singles.








```
var context = new DatabaseFirstDemoEntities();
var post = new Post()
{
    Body = "Body",
    DatePublished = DateTime.Now,
    Title = "Title",
    PostID = 2
};
context.Posts.Add(post);
context.SaveChanges();
```



<u>CodeFirst</u>

Add package : EntityFramework by Microsoft

Create a model.

Add a class of type DbContext. Add DbSet<Table> property, for each table.

Add connectionString to App.config file

Open NuGet package manager window.

Run commands :

enable-migrations - run once for a project

add-migration <migration name> - migration store the changes to the DB

Update-DataBase - to update all not updated migrations to DB

```
public partial class Post
{
    public int PostID { get; set; }
    public System.DateTime DatePublished { get; set; }
    public string Title { get; set; }
    public string Body { get; set; }
    public string UserEmail { get; set; }
}

public class BlogDBContext : DbContext
{
    public DbSet<Post> Posts { get; set; }
}
```

```
<connectionStrings>
  <add name="BlogDbContext"
      connectionString="data source=PETER-PC\SQLEXPRESS;initial catalog=CodeFirstDemo;integrated security=SSPI"
      providerName="System.Data.SqlClient"
    />
</connectionStrings>
```

Analyze | Tools | Extensions | Window | Help | Search (Ctrl+Q) | EntityFramewokInDepth

- Get Tools and Features...
- Connect to Database...
- Connect to Server...
- Add SQL Server...
- SQL Server
- Data Lake
- Code Snippets Manager... Ctrl+K, Ctrl+B
- Choose Toolbox Items...
- NuGet Package Manager
  - Package Manager Console
  - Manage NuGet Packages for Solution...
  - Package Manager Settings
- Create GUID
- Error Lookup
- Spy++
- External Tools...
- Command Line
- Import and Export Settings...
- Customize...
- Options...

```
Package Manager Console
Package source: All                    Default project: CodeFirstDemoV2
PM> enable-migrations
Checking if the context targets an existing database...
PM> add-migration CreatePosts
Scaffolding migration 'CreatePosts'.
The Designer Code for this migration file includes a snapshot of your current Code First model. This snapshot is used
migration. If you make additional changes to your model that you want to include in this migration, then you can re-sc
PM> Update-Database
Specify the '-Verbose' flag to view the SQL statements being applied to the target database.
Applying explicit migrations: [202109251638095_CreatePosts].
Applying explicit migration: 202109251638095_CreatePosts.
Running Seed method.
PM> add-migration PostsUpdated
Scaffolding migration 'PostsUpdated'.
The Designer Code for this migration file includes a snapshot of your current Code First model. This snapshot is used
migration. If you make additional changes to your model that you want to include in this migration, then you can re-sc
PM> update-database
Specify the '-Verbose' flag to view the SQL statements being applied to the target database.
Applying explicit migrations: [202109251640068_PostsUpdated].
Applying explicit migration: 202109251640068_PostsUpdated.
Running Seed method.
PM>
```

Search Solution Explorer (Ctrl+;)

- Solution 'EntityFramewokInDepth' (2 of 2 projects)
  - CodeFirstDemoV2
    - Properties
    - References
    - Migrations
      - 202109251638095_CreatePosts.cs
      - 202109251640068_PostsUpdated.cs
      - Configuration.cs
    - App.config
    - packages.config
    - Program.cs
      - Post
      - BlogDBContext
      - Program
  - DatabaseFirstDemo

```
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace CodeFirstDemoV2
{
    public partial class Post
    {
        public int PostID { get; set; }
        public System.DateTime DatePublished { get; set; }
        public string Title { get; set; }
        public string Body { get; set; }
        public string UserEmail { get; set; }
    }

    public class BlogDBContext : DbContext
    {
        public DbSet<Post> Posts { get; set; }
    }

    class Program
    {
        static void Main(string[] args)
```

DB changes - Dadabase First

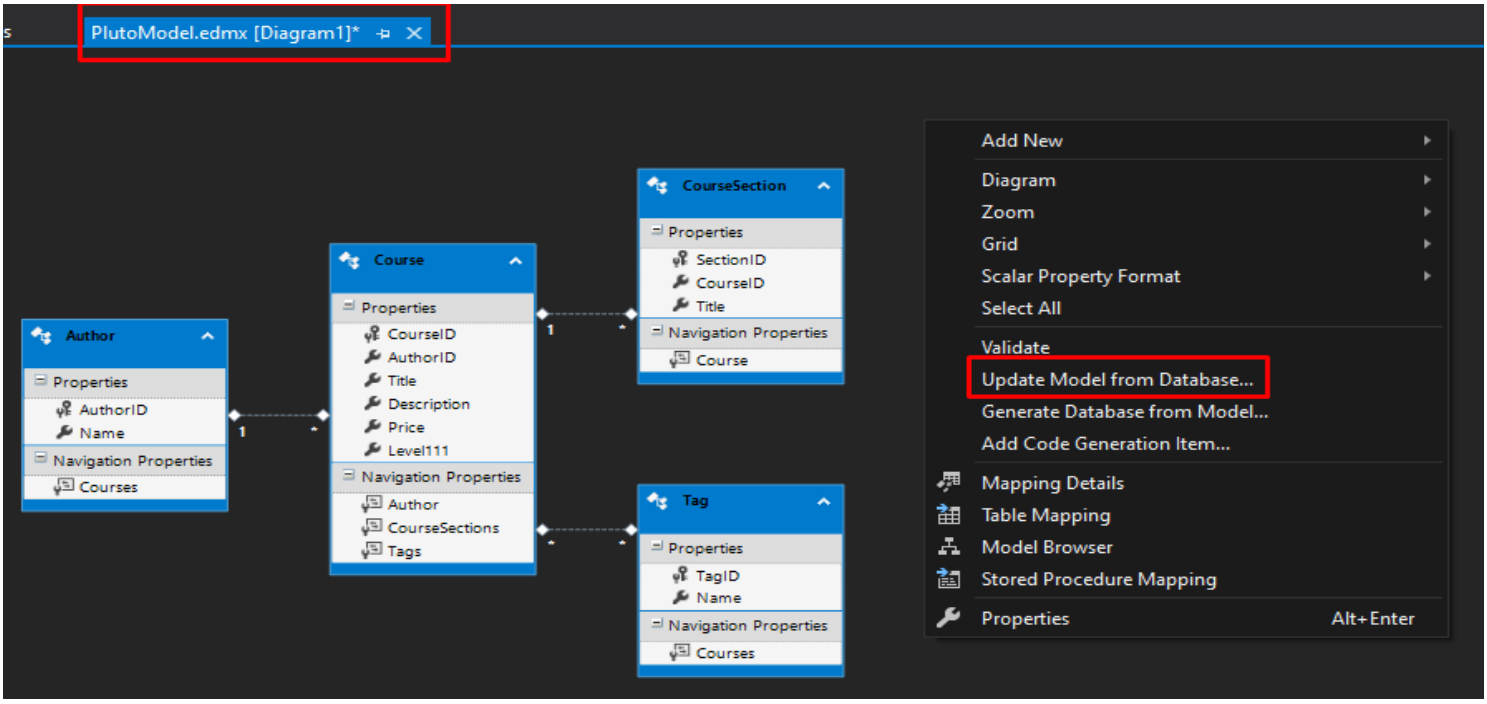
Change the DB via SQL Management Studio(or other why)

In the .edmx file, right mouse button, on an empty place on the editor, opens a menu. Choose “Update Model from Database..”.

Added tables and cols will be validated. Name changes, data types changes, deletion of tables and cols will require manual mapping.

See “Error List” tab for all mapping errors.

Save the .edmx file to make changes to model - **MUST** to implement changes.



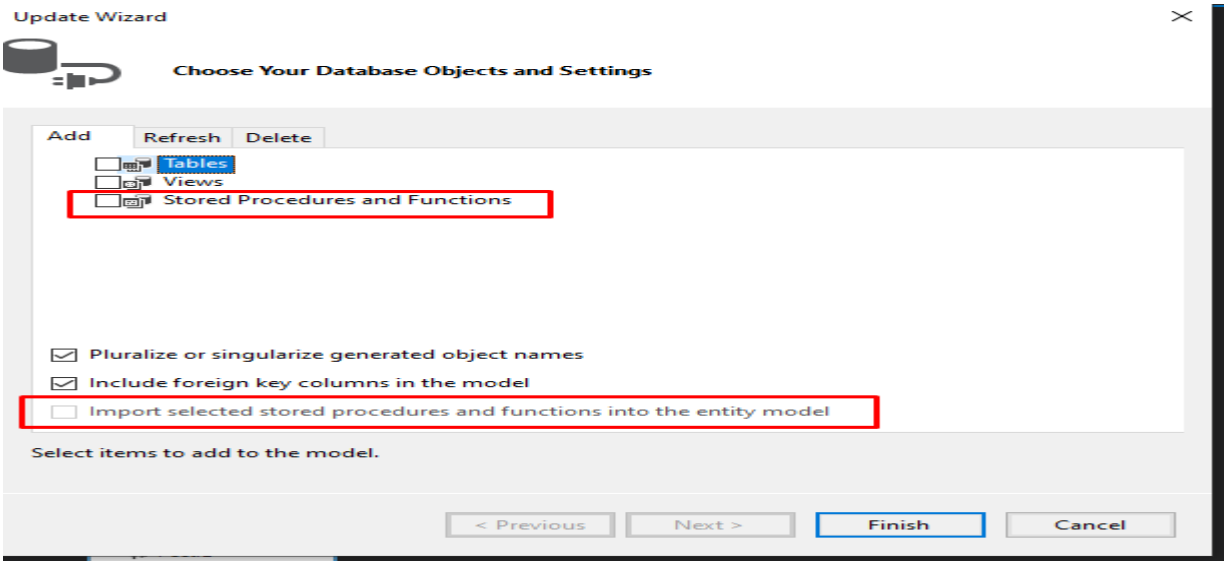
Add Stored Procedures

Add From the Update Wizard

Checkout box “Import selected…” sow function will be created in the conceptual model and not stay only in the storage model.

Save .edmx file to apply changes.

Execute stored procedure via Context.cs file.



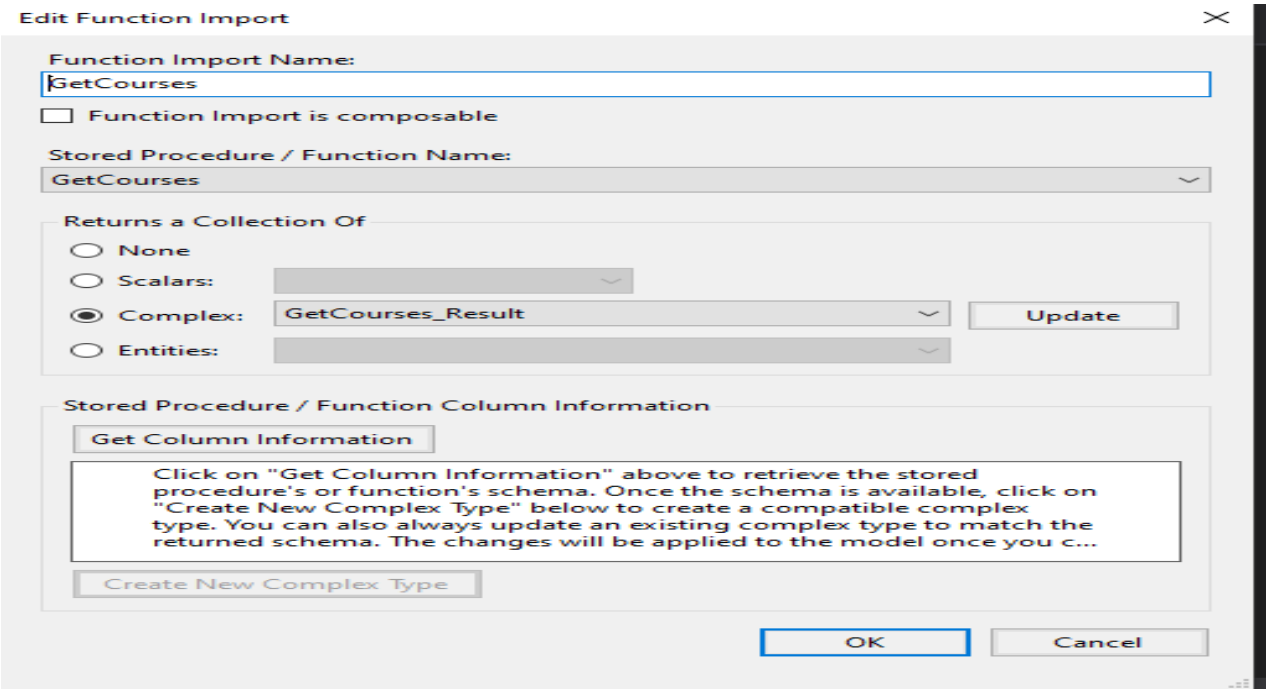
```
Program.cs    PlutoModel.edmx [Diagram1]    PlutoModel.Context.cs
PlutoDbFirst                                GetCourses()

45
46  [DbFunction("PlutoDbContext", "funcGetAuthorCourses")]
47  public virtual IQueryable<funcGetAuthorCourses_Result> funcGetAuthorCourses(Nullable<int> authorID)
48  {
49      var authorIDParameter = authorID.HasValue ?
50          new ObjectParameter("AuthorID", authorID) :
51          new ObjectParameter("AuthorID", typeof(int));
52
53      return ((IObjectContextAdapter)this).ObjectContext.CreateQuery<funcGetAuthorCourses_Result>("[Pl
54  }
55
56  public virtual ObjectResult<GetCourses_Result> GetCourses()
57  {
58      return ((IObjectContextAdapter)this).ObjectContext.ExecuteFunction<GetCourses_Result>("GetCourse
59  }
60
```

```
class Program
{
    static void Main(string[] args)
    {
        var contextPluto = new PlutoDbContext();
        using(var courses= contextPluto.GetCourses())
        {
            foreach(var course in courses)
            {
                Console.WriteLine(course.Title);
            }
        }
        Console.ReadLine();
    }
```

<u>Functions Editing</u>

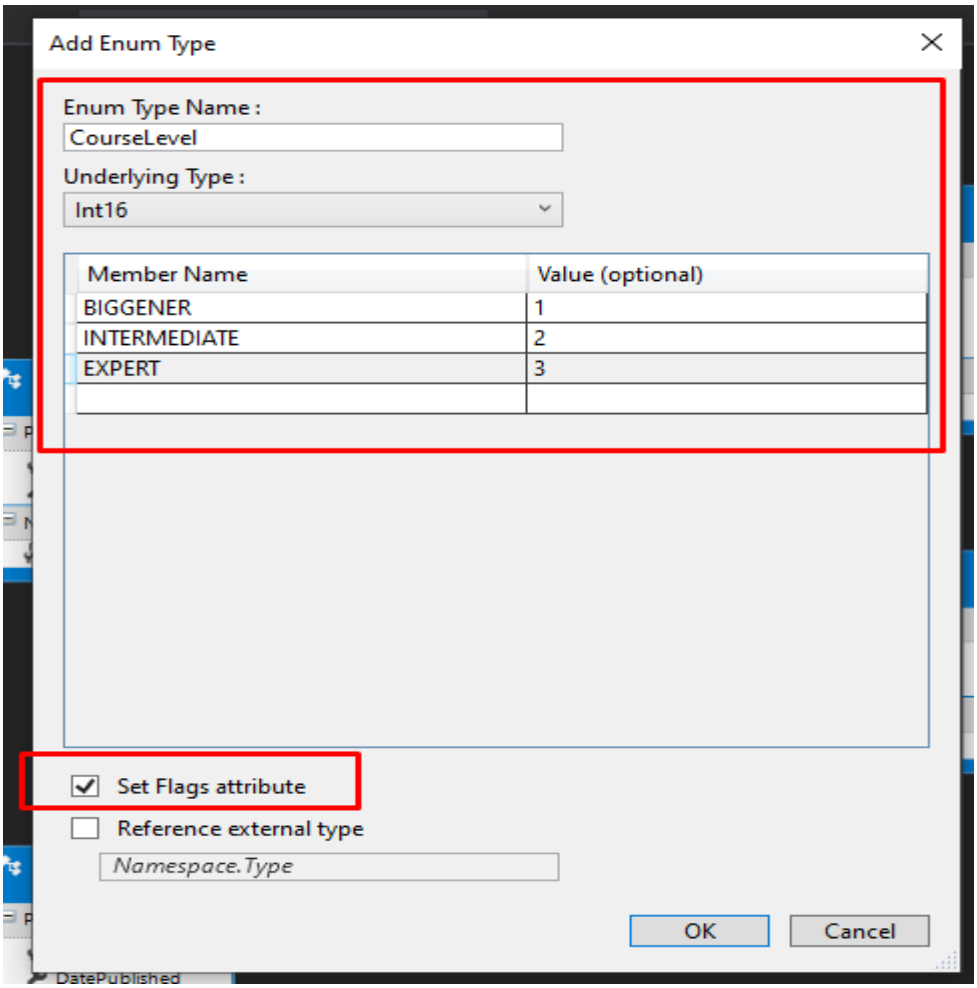
Edit function via “Edit Function Import” tab.



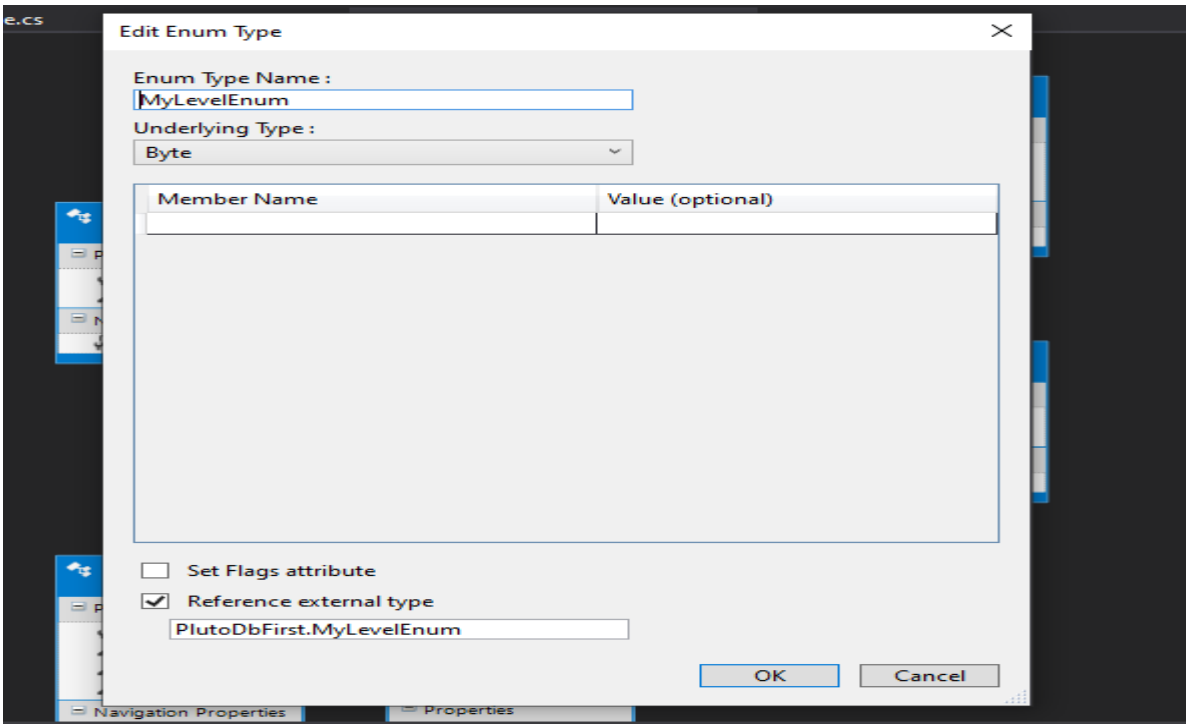
Enums

Create enum model

Choose “Set Flags attribute” to enable bitwise operations



Reference enum from code. The name of the enum in model have to identical to the name in code. In “reference…” check box enter fully qualified name.



Database changes - Code First

Create new db CodeFirst:

CodeFirst

Add package : EntityFramework by Microsoft

Create a model.

Initialize an object of type DbContext. Add DbSet<Table> property, for each table.

Add connectionString to App.config file

Open NuGet package manager window.

Run commands :

enable-migrations - run once for a project

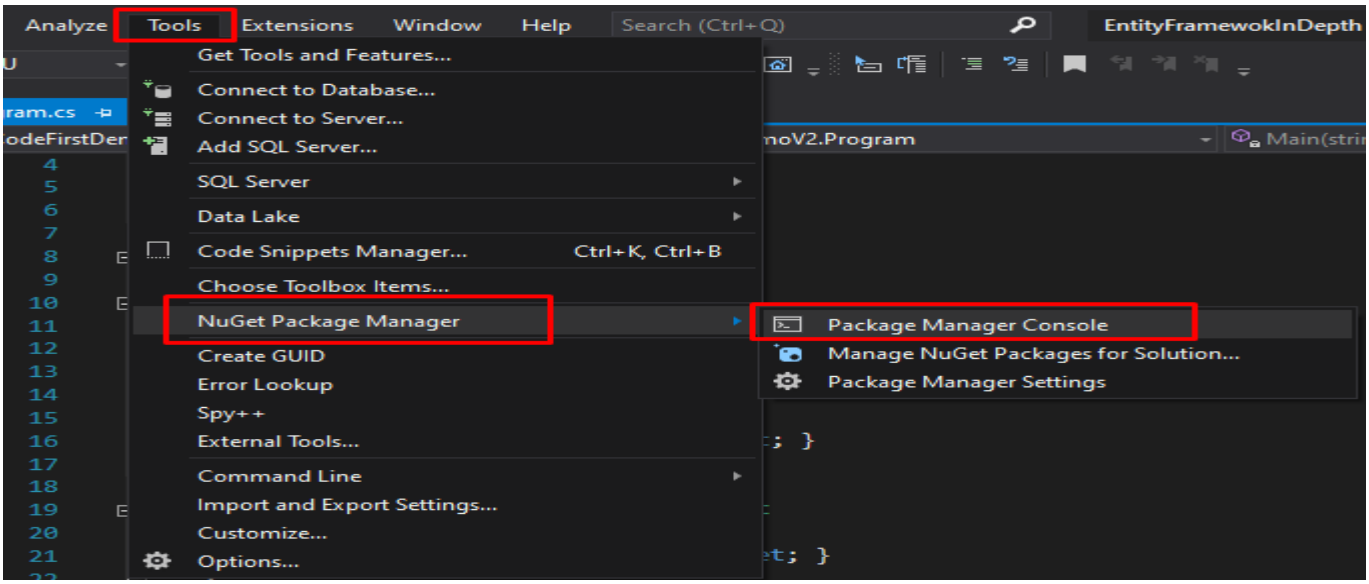
add-migration <migration name> - migration store the changes to the DB

Update-DataBase - to update all not updated migrations to DB

```
public partial class Post
{
    public int PostID { get; set; }
    public System.DateTime DatePublished { get; set; }
    public string Title { get; set; }
    public string Body { get; set; }
    public string UserEmail { get; set; }
}

public class BlogDBContext : DbContext
{
    public DbSet<Post> Posts { get; set; }
}
```

```
<connectionStrings>
  <add name="BlogDbContext"
       connectionString="data source=PETER-PC\SQLEXPRESS;initial catalog=CodeFirstDemo;integrated security=SSPI"
       providerName="System.Data.SqlClient"
    />
</connectionStrings>
```



```
Package Manager Console
Package source: All        Default project: CodeFirstDemoV2
PM> enable-migrations
Checking if the context targets an existing database...
PM> add-migration CreatePosts
Scaffolding migration 'CreatePosts'.
The Designer Code for this migration file includes a snapshot of your current Code First model. This snapshot is used
migration. If you make additional changes to your model that you want to include in this migration, then you can re-sc
PM> Update-Database
Specify the '-Verbose' flag to view the SQL statements being applied to the target database.
Applying explicit migrations: [202109251638095_CreatePosts].
Applying explicit migration: 202109251638095_CreatePosts.
Running Seed method.
PM> add-migration PostsUpdated
Scaffolding migration 'PostsUpdated'.
The Designer Code for this migration file includes a snapshot of your current Code First model. This snapshot is used
migration. If you make additional changes to your model that you want to include in this migration, then you can re-sc
PM> update-database
Specify the '-Verbose' flag to view the SQL statements being applied to the target database.
Applying explicit migrations: [202109251640068_PostsUpdated].
Applying explicit migration: 202109251640068_PostsUpdated.
Running Seed method.
PM>
```

```
Search Solution Explorer (Ctrl+;)
Solution 'EntityFramewokInDepth' (2 of 2 projects)
  CodeFirstDemoV2
    Properties
    References
    Migrations
      202109251638095_CreatePosts.cs
      202109251640068_PostsUpdated.cs
      Configuration.cs
    App.config
    packages.config
    Program.cs
      Post
      BlogDBContext
      Program
  DatabaseFirstDemo
```

```
4   using System.Linq;
5   using System.Text;
6   using System.Threading.Tasks;
7
8   namespace CodeFirstDemoV2
9   {
10      public partial class Post
11      {
12          public int PostID { get; set; }
13          public System.DateTime DatePublished { get; set; }
14          public string Title { get; set; }
15          public string Body { get; set; }
16          public string UserEmail { get; set; }
17      }
18
19      public class BlogDBContext : DbContext
20      {
21          public DbSet<Post> Posts { get; set; }
22      }
23
24      class Program
25      {
26          static void Main(string[] args)
```

<u>Update existing DB CodeFirst</u>

<u>Process:</u> Create model from existing db. Enter changes to the model. Create migration and use the to update the db.

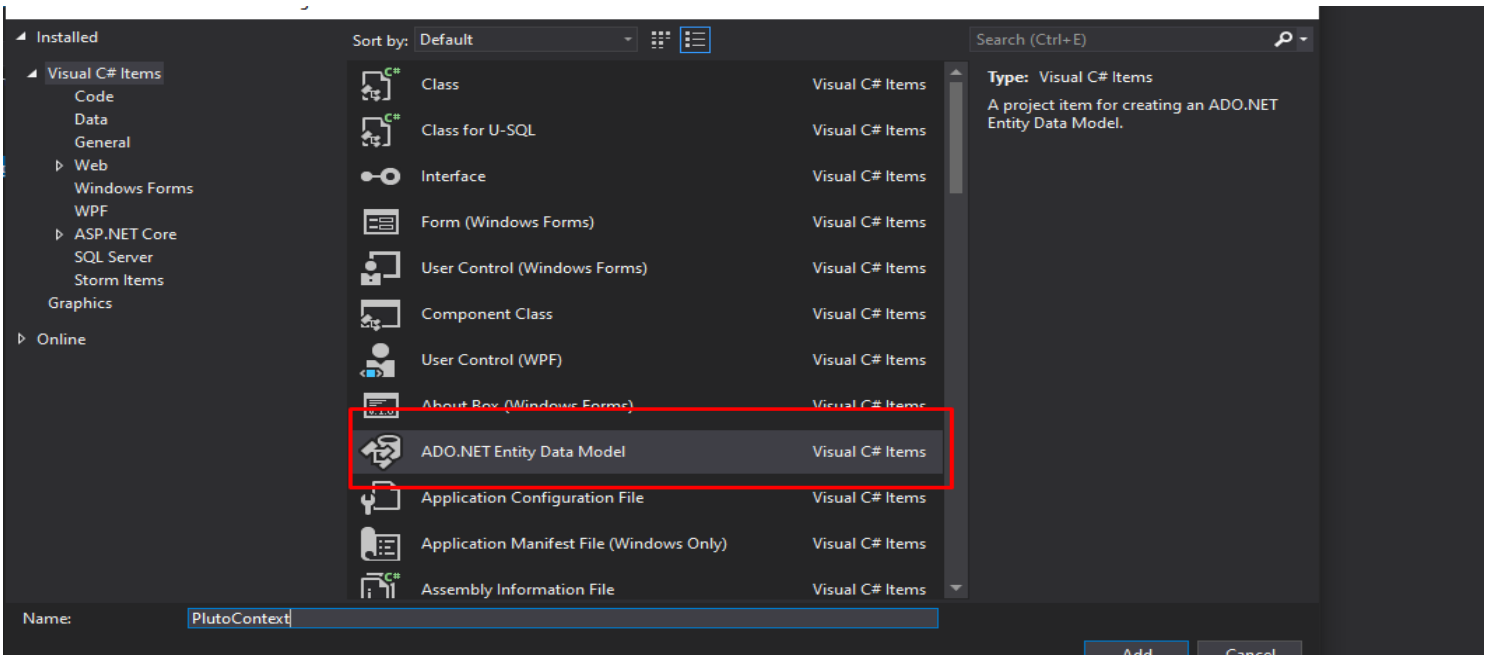
Add new ADO.NET Entity Data model

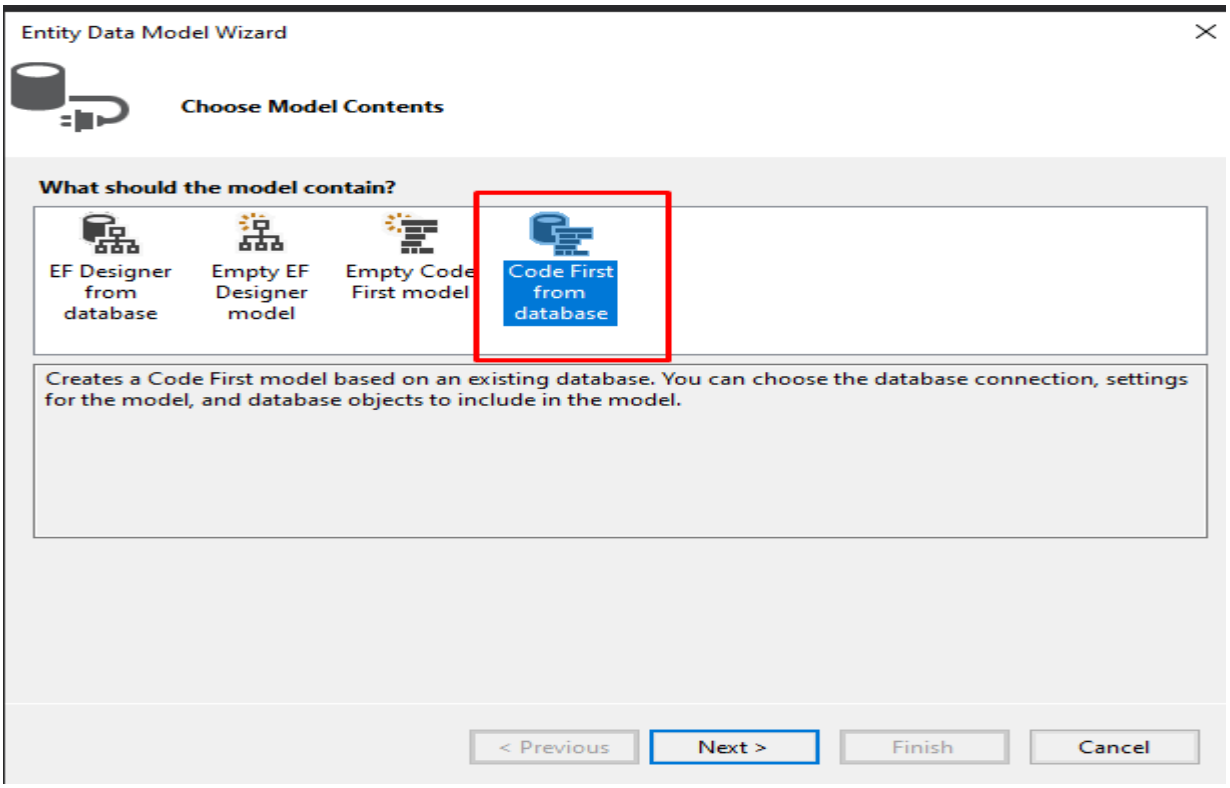
Choose code first from database

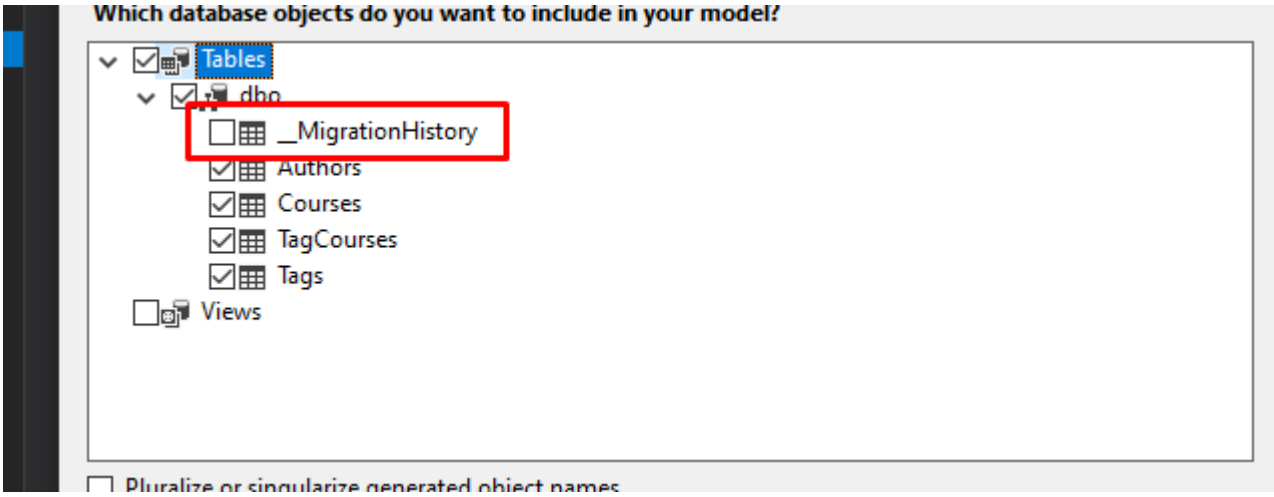
Choose tables, dot import the migrations table to the model

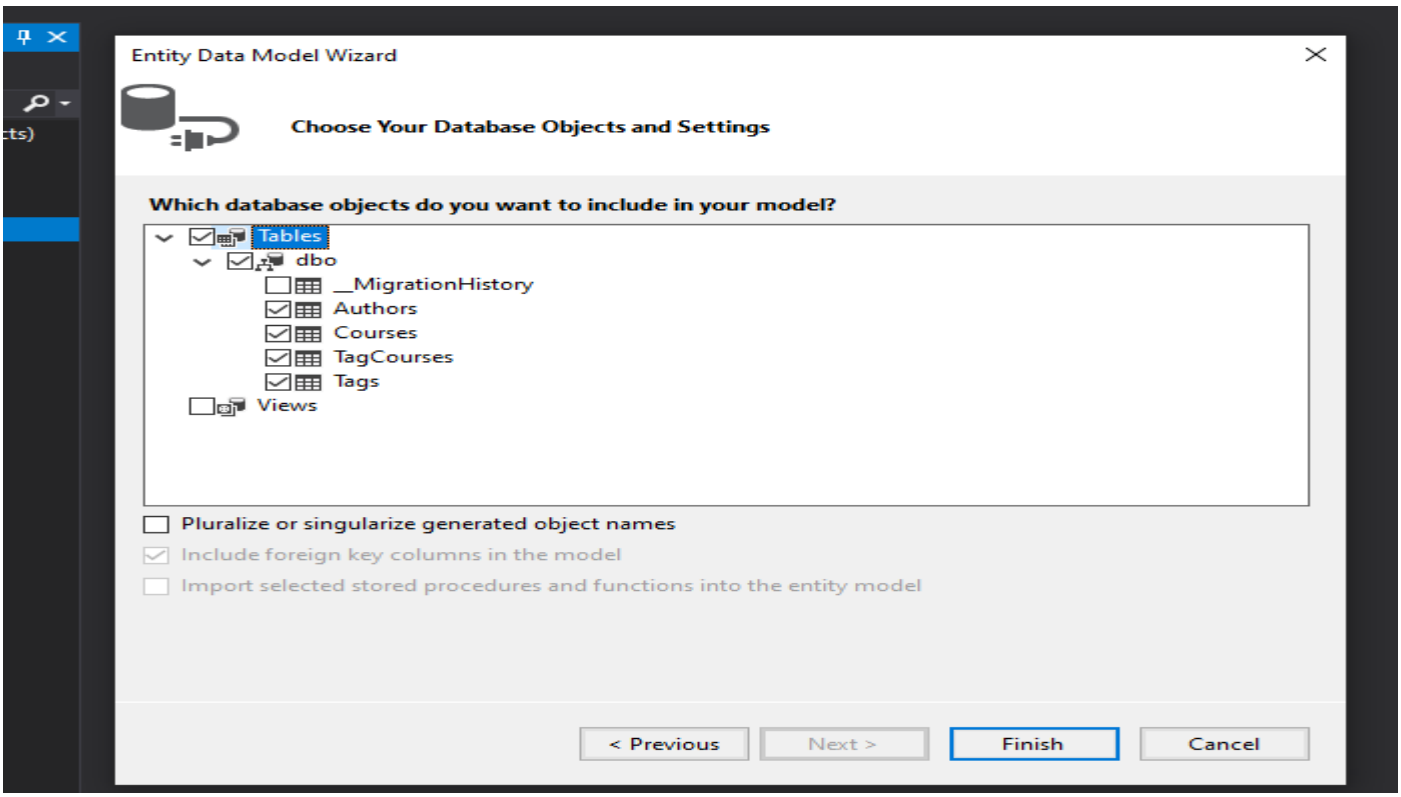
Create empty migration, to start track of migrations changes, and update it to the database.

Run “add-migration InitialModel -IgnoreChanges -Force”.- IgnoreChanges - to create empty migration, -Force - rewrite existing migration.


Installed
Visual C# Items
Code
Data
General
Web
Windows Forms
WPF
ASP.NET Core
SQL Server
Storm Items
Graphics
Online
Sort by: Default
Search (Ctrl+E)
Class
Class for U-SQL
Interface
Form (Windows Forms)
User Control (Windows Forms)
Component Class
User Control (WPF)
ADO.NET Entity Data Model
Application Configuration File
Application Manifest File (Windows Only)
Assembly Information File
Visual C# Items
Type: Visual C# Items
A project item for creating an ADO.NET Entity Data Model.
Name:
PlutoContext


Entity Data Model Wizard
Choose Model Contents
What should the model contain?
EF Designer from database
Empty EF Designer model
Empty Code First model
Code First from database
Creates a Code First model based on an existing database. You can choose the database connection, settings for the model, and database objects to include in the model.
< Previous
Next >
Finish
Cancel


Which database objects do you want to include in your model?
Tables
dbo
__MigrationHistory
Authors
Courses
TagCourses
Tags
Views
Pluralize or singularize generated object names




```
The Designer Code for this migration file includes a snapshot of your current Co
calculate the changes to your model when you scaffold the next migration. If you
want to include in this migration, then you can re-scaffold it by running 'Add-M
PM> add-migration InitialModel -IgnoreChanges -Force
A version of Entity Framework older than 6.3 is also installed. The newer tools
 for the older version.
Re-scaffolding migration 'InitialModel'.
PM> add-migration InitialModel -IgnoreChanges
A version of Entity Framework older than 6.3 is also installed. The newer tools
 for the older version.
Scaffolding migration 'InitialModel'.
The Designer Code for this migration file includes a snapshot of your current Co
calculate the changes to your model when you scaffold the next migration. If you
want to include in this migration, then you can re-scaffold it by running 'Add-M
PM> update data-base
```

Migrations

Add new class

Create a new class. Define Id/<className>Id properties. EF uses this two names as primary/id keys by default.
Introduce the class to EF by adding it to the DbContext,add a DbSet<ClassName> property.
Add data to the category table, by using Sql("query") command.
Sql("query") - add query to migration. The query will be executed on the database.
Create migration and update db

```
public class Category
{
    public int Id { get; set; }
    //public int CategoryId { get; set; }
    public string Name { get; set; }
}
}
```

Modify an existing class

Add property - add a property to class, add migration, update db.

Change property name

Change the property name in the model

Create migration, modify it to save the data in the column and to downgrade the migration.

```
        AddColumn("dbo.Courses", "Name", c => c.String(nullable: false));
        Sql("Update Courses SET Name=Title");
        DropColumn("dbo.Courses", "Title");

        //2
        //RenameColumn("dbo.Courses", "Title", "Name");
    }

    public override void Down()
    {
        //1
        AddColumn("dbo.Courses", "Title", c => c.String(nullable: false));
        Sql("Update Courses SET Title=Name");
        DropColumn("dbo.Courses", "Name");

        //2
        //RenameColumn("dbo.Courses", "Name", "Title");
    }
```

Delete Column - update model(delete property), add migration. Update db.

Delete class

Delete the properties of a class type in other classes. Delete in the “Delete Column” scenario.

Delete the class from the model. Delete DbSet<ClassName> from the .DbContext file.

Add migration. Update migration to save copies of the delete data and restore migration(if needed).

Update database.

```
    public override void Up()
    {
        //Sql("CREATE TABLE...");
        CreateTable(
            "dbo._Categories",
            c => new
            {
                Id = c.Int(nullable: false, identity: true),
                Name = c.String(),
            })
            .PrimaryKey(t => t.Id);
        Sql("INSERT INTO _Categories (Name) SELECT Name FROM Categories");
        DropTable("dbo.Categories");
    }

    public override void Down()
    {
        CreateTable(
            "dbo.Categories",
            c => new
                {
                    Id = c.Int(nullable: false, identity: true),
                    Name = c.String(),
                })
            .PrimaryKey(t => t.Id);
        Sql("INSERT INTO Categories (Name) SELECT Name FROM _Categories");
        DropTable("dbo._Categories");
    }
```

Migration - Recovering from mistake - Don't delete the migration with the mistake. Create new migration with the fix.

Downgrading db

run update-database -TargetMigration:TargetMigrationName

to restore the last state of the db run update-database

If the migration are defined sow the data is not stored in upgrades or downgrades data will be lost.

Seeding db

Seeding the db, use the Seed function in the migration configuration file

```
Package Manager Console    Configuration.cs
PlutoCodeFirstExestingDb    PlutoCodeFirstExestingDb.Migrations.Configuration    Seed(PlutoContext context)
7   using System.Linq;
8
9   internal sealed class Configuration : DbMigrationsConfiguration<PlutoCodeFirstExestingDb.PlutoContext>
10  {
11      public Configuration()
12      {
13          AutomaticMigrationsEnabled = false;
14      }
15
16      protected override void Seed(PlutoCodeFirstExestingDb.PlutoContext context)
17      {
18          context.Authors.AddOrUpdate(a => a.Name,
19              new Author()
20              {
21                  Name = "Peter",
22                  Courses = new Collection<Course>()
23                  {
24                      new Course()
25                      {
26                          Name="special course",
27                          Description="very very special course"
28                      }
29                  }
30              });
31      }
```

Deployment

Create script from first migration to last one : Update--Database -Script -SourceMigration:0

Create script for a range of migrations :

Update--Database -Script -SourceMigration:<Migx> -TargetMigration:<Migy>

Data Annotations

```
}

public int Id { get; set; }

[Required]
[MaxLength(255)]
public string Name { get; set; }

[Required]
[MaxLength(2000)]
public string Description { get; set; }

public int Level { get; set; }

public float FullPrice { get; set; }

[ForeignKey("Author")]
public int AuthorId { get; set; }
public virtual Author Author { get; set; }
```

Column Name	Data Type	Allow Nulls
Id	int	☐
Name	nvarchar(MAX)	☑
Description	nvarchar(MAX)	☑
[Level]	int	☐
FullPrice	real	☐
Author_Id	int	☑
		☐

Column Name	Data Type	Allow Nulls
Id	int	☐
Name	nvarchar(255)	☐
Description	nvarchar(2000)	☐
[Level]	int	☐
FullPrice	real	☐
AuthorId	int	☐
		☐

<u>Fluent API</u>

Override OnModelCreating(DbModelBuilder modelBuilder) method in context file
Write changes in the method.
Add migration, and update database.

<u>Required and nvarchar size</u>

```
protected override void OnModelCreating(DbModelBuilder modelBuilder)
{
    modelBuilder.Entity<Course>()
        .Property(c => c.Name)
        .IsRequired()
        .HasMaxLength(255);

    base.OnModelCreating(modelBuilder);
}
```

IsRequired() - sets the property to be not nullable.
HasMaxLength(x) - set a string property to be of max length of x

<u>One many relation</u>

one Course to many Author relation

```
modelBuilder.Entity<Course>()
    .HasRequired(course => course.Author)
    .WithMany(auther => auther.Courses)
    .HasForeignKey(course => course.AuthorId)
    .WillCascadeOnDelete(false);
```

HasRequired - each Course has a Author
WithMany - every Author have many Course
HasForeignKey - defines a foreign key for Author table Author.Id column
WillCascadeOnDelete(bool x) - set the cascade to x.

Many to many relation

```
modelBuilder.Entity<Tag>()
    .HasMany(tag => tag.Courses)
    .WithMany(course => course.Tags)
    .Map(m => m.ToTable("CourseTags"));
```

HasMany - Tag has many Course
WithMay - Course has many Tag
Map(string x) - map the relation table name to x

One to one relation

In one to one relation one is parent the other is dependent.
The parent created first, must be defined.

```
modelBuilder.Entity<Course>()
    .HasRequired(course => course.Cover)
    .WithRequiredPrincipal(course => course.Course);
```

WithRequiredPrincipal - Course is the parent and Cover is the dependent.

EACH RELATION CAN BE DEFINED IN THE OTHER WAY