

# Entity Framework (EF)

- Lịch sử ra đời
- Các khái niệm cơ bản
- Kiến trúc EF
- DB First/Model First
- Code First/Code Second
- Kế thừa
- Eager/Lazy & Explicit Loading
- Performance/Profiling

# Tài liệu tham khảo

- Entity Framework 6 Recipes 2nd Edition.pdf
- <https://msdn.microsoft.com/en-us/data/ef.aspx>
- <http://www.asp.net/entity-framework>

# Lịch sử ra đời EF

- EF aka EF1 aka EF 3.5 ra đời kèm .NET 3.5 (VS2008)
- EF 4 with .NET 4.0 (VS2010)
  - POCO Support/Lazy Loading
- EF 4.1
  - DbContext API/Code First/Nuget package
- EF 5.0
- EF 6.0

# EF là gì?

**Entity Framework (EF)** is an object-relational mapper that enables .NET developers to work with relational data using domain-specific objects.

# Tại sao phải dùng EF?

EF giảm thiểu việc cần thiết xây dựng **mã nguồn truy vấn dữ liệu** mà lập trình viên cần phải làm.

EF được Microsoft hỗ trợ và có kế hoạch phát triển lâu dài, trải qua nhiều phiên bản.

# Tải EF ở đâu?

**EF 6.0** sẵn có kèm theo **VS2013**

**Tải về:**

<https://www.microsoft.com/en-us/download/details.aspx?id=40762>

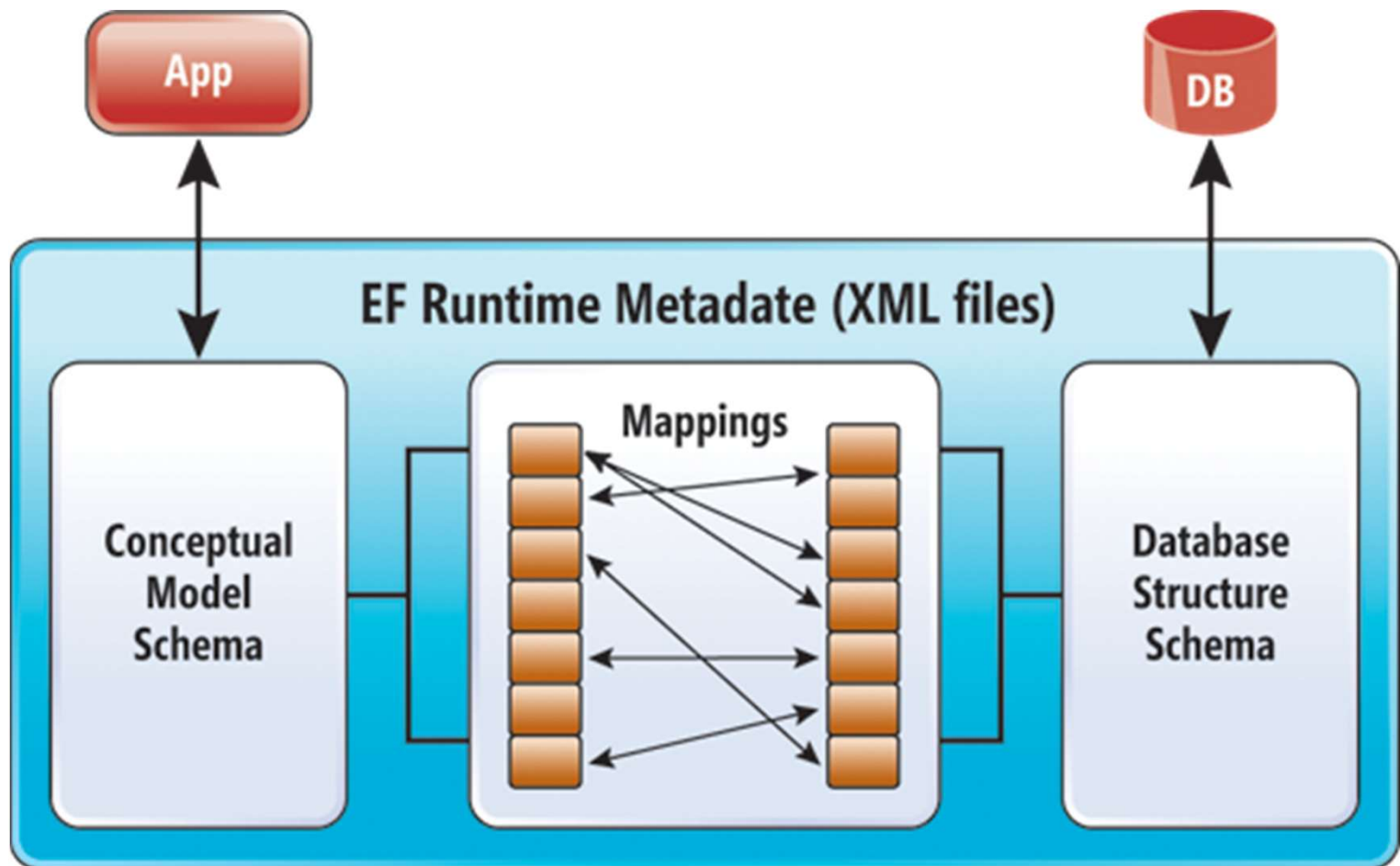
# ORM là gì

**ORM = Object-Relational Mapping**

**Wikipedia:** *“Object-relational mapping (ORM, O/RM, and O/R mapping tool) in computer science is a programming technique for converting data between incompatible type systems in object-oriented programming languages.”*

# ORM là gì

## Mô hình



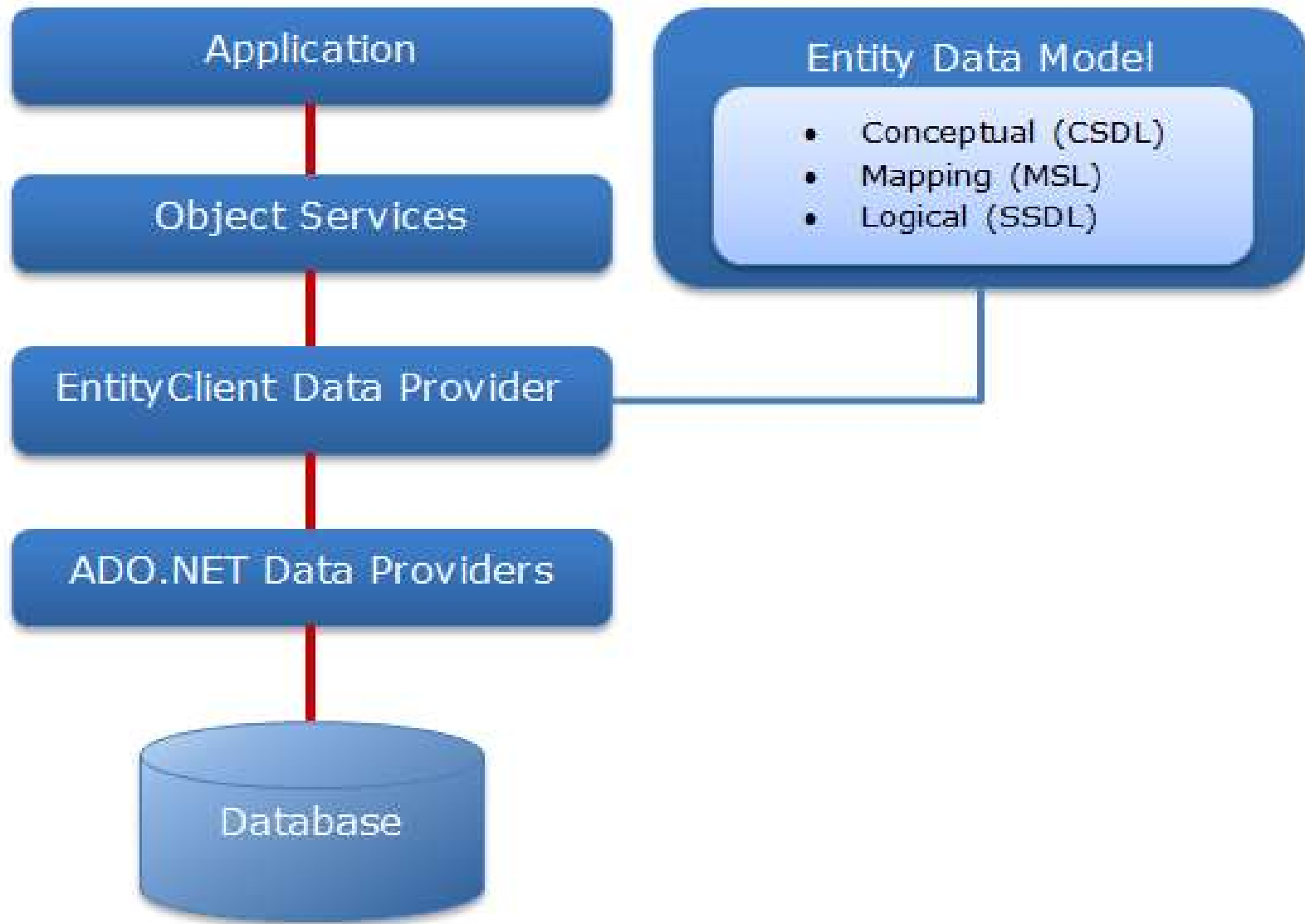


# ORM là gì

## Một số ORM nổi tiếng

- Nhibernate <http://nhibernate.info/>
- Hibernate (framework) <http://hibernate.org/>
- BLToolkit

# Kiến trúc EF



# Kiến trúc EF

- Object Services
  - Đây là nơi chứa DbContext, thể hiện quá trình tương tác giữa ứng dụng và nguồn dữ liệu. OS cung cấp các tiện ích để truy vết các thay đổi và quản lý nhận dạng, đồng thời và các quan hệ, và các thay đổi ở DB

# Kiến trúc EF

- EntityClient Data Provider
  - Đây là nơi cung cấp các kết nối, diễn dịch các truy vấn thực thể thành truy vấn nguồn dữ liệu, trả về data reader để EF dùng chuyển dữ liệu thực thể thành các đối tượng.

# Kiến trúc EF

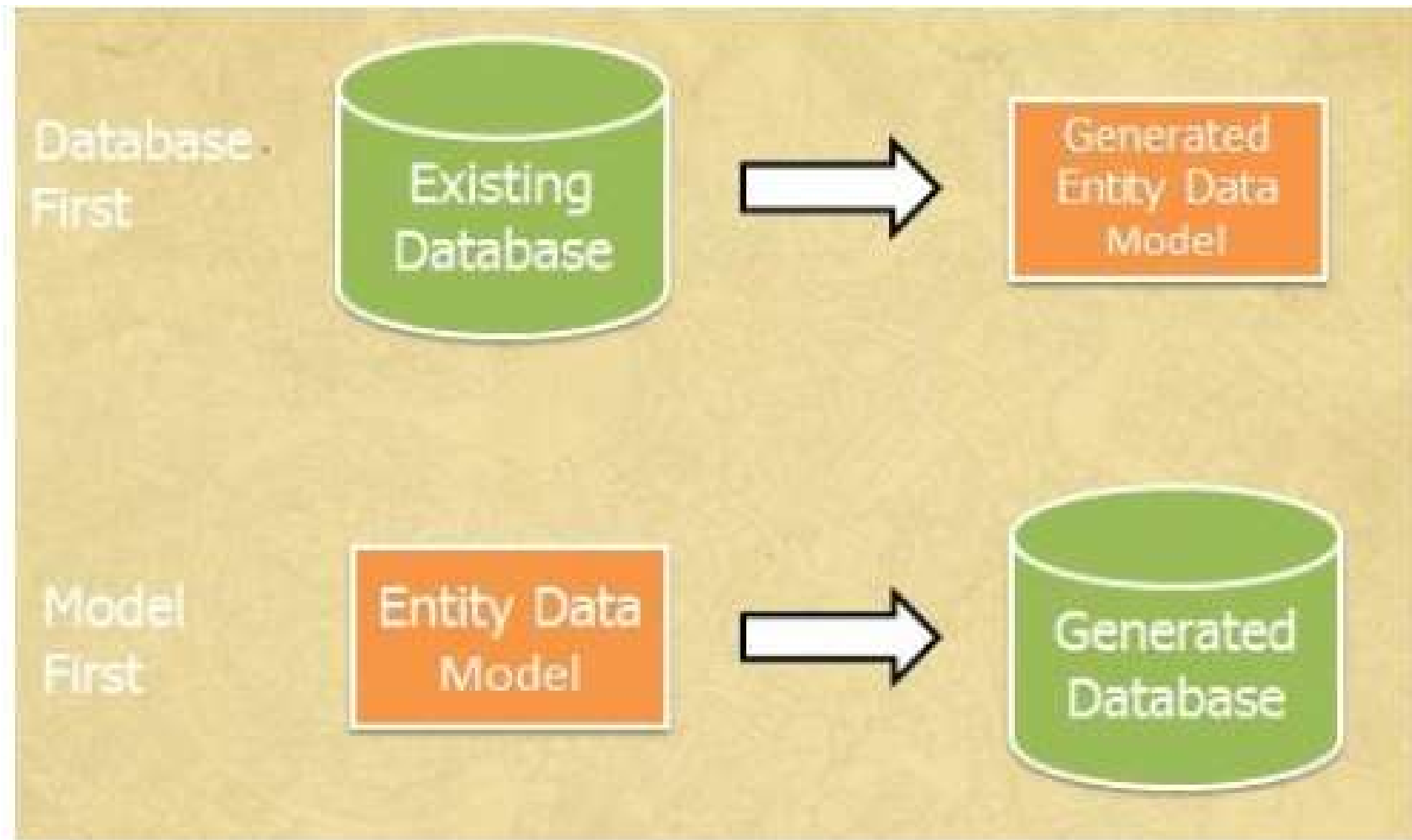
- Data Providers
  - Đây là tầng thấp nhất để dịch các truy vấn L2E (LINQ to Entity) thông qua cây lệnh thành các câu lệnh SQL và thực thi các câu lệnh trong hệ thống DBMS nào đó.

# Kiến trúc EF

- Data Providers
  - Đây là tầng thấp nhấ để dịch các truy vấn L2E (LINQ to Entity) thông qua cây lệnh thành các câu lệnh SQL và thực thi các câu lệnh trong hệ thống DBMS nào đó.
  - <http://www.slideshare.net/betclicTech/betclic-training-entityframework>

# DB First/Model First

- Mô hình đơn giản



# DB Model vs Conceptual Model

- DB Model được thiết kế dùng để lưu trữ dữ liệu thực. Nó cũng dùng để tối ưu hóa việc lưu trữ và tương tác dữ liệu.
- Mô hình khái niệm (Conceptual) hay Business Object model dùng để quản lý các yêu cầu dữ liệu từ chương trình/ứng dụng.
- 2 mô hình này khác nhau
- EF cho phép ánh xạ (mapping) giữa 2 mô hình này dễ dàng



# Code First to New Database

- Cách này định nghĩa code trước, sau đó từ code gieo (sản sinh) ra database
- Code First cho phép định nghĩa các bảng dữ liệu bằng các lớp code.
- Các lớp code chứa các thuộc tính cũng là sẽ thuộc tính bảng dữ liệu khi sản sinh ra.

# Code First to New Database

- Yêu cầu cài đặt:
  - Cần có VS 2010 trở lên
  - Nếu dùng VS 2010 thì cần có NuGet

# Code First to New Database

## 1. Tạo ứng dụng

- Mở Visual Studio
- **File -> New -> Project...**
- Chọn **Windows** từ menu trái và chọn **Console Application**
- Đặt tên **CodeFirstNewDatabaseSample**
- Chọn **OK**

# Code First to New Database

## 1. Tạo ứng dụng

- **Tools -> Library Package Manager -> Package Manager Console, gõ lệnh**
- **Install-Package EntityFramework - IncludePrerelease**

# Code First to New Database

## 2. Tạo mô hình

Bước này cần định nghĩa mô hình code cho dự án. Trong ví dụ này, định nghĩa các lớp trong tập tin **Program.cs**, trên thực tế phải định nghĩa chia thành các lớp theo tên.

# Code First to New Database

## 2. Tạo mô hình

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

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

# Code First to New Database

## 2. Tạo mô hình

```
public class Blog
{
    public int BlogId { get; set; }
    public string Name { get; set; }

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

# Code First to New Database

Trong 2 lớp trên, chúng ta thêm 2 thuộc tính điều hướng ảo (**Blog.Posts** and **Post.Blog**).

Điều này cho phép tính năng **Lazy Loading** của **Entity Framework**.

**Lazy Loading** nghĩa là nội dung của các thuộc tính sẽ tự động tải từ database khi bạn muốn truy cập chúng.



# Code First to New Database

## 3. Tạo Context

Context một lớp code thể hiện hoạt động tương tác với database. Lớp code này kế thừa **System.Data.Entity.DbContext** và định nghĩa kiểu **DbSet<TEntity>** cho mỗi lớp trong mô hình.

# Code First to New Database

## 3. Tạo Context

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

# Code First to New Database

## 3. Tạo Context

A **DbContext** instance represents a combination of the Unit Of Work and Repository patterns such that it can be used to query from a database and group together changes that will then be written back to the store as a unit. DbContext is conceptually similar toObjectContext.

# Code First to New Database

## 3. Tạo Context

A non-generic version of **DbSet<TEntity>** which can be used when the type of entity is not known at build time.

# Code First to New Database

## 4. Đọc và ghi dữ liệu

Trong dự án ví dụ, chúng ta có thể thực hiện ở hàm Main, dùng LINQ tương tác với các thành phần dữ liệu trong database

# Code First to New Database

## 4. Đọc và ghi dữ liệu

```
using (var db = new BloggingContext())
{
    // Create and save a new Blog
    Console.Write("Enter a name for a new Blog: ");
    var name = Console.ReadLine();
    var blog = new Blog { Name = name };
    db.Blogs.Add(blog);
    db.SaveChanges();
}
```

*... tiếp trang sau*

# Code First to New Database

## 4. Đọc và ghi dữ liệu

```
// Display all Blogs from the database  
var query = from b in db.Blogs orderby b.Name  
            select b;
```

```
Console.WriteLine("All blogs in the database:");  
foreach (var item in query)  
{  
    Console.WriteLine(item.Name);  
}
```

# Code First to New Database

## 5. Database được tạo ra thế nào và dữ liệu lưu ở đâu

Theo mặc định DbContext sẽ tạo dữ liệu ở:

- Nếu là SQL Express (cài kèm với VS 2010) thì Code First tạo database ở đây
- Nếu SQL Express không có thì Code First sẽ dùng LocalDb (cài kèm với VS 2012)



# Code First to New Database

## 5. Database được tạo ra thế nào và dữ liệu lưu ở đâu

Theo mặc định DbContext sẽ tạo dữ liệu ở:

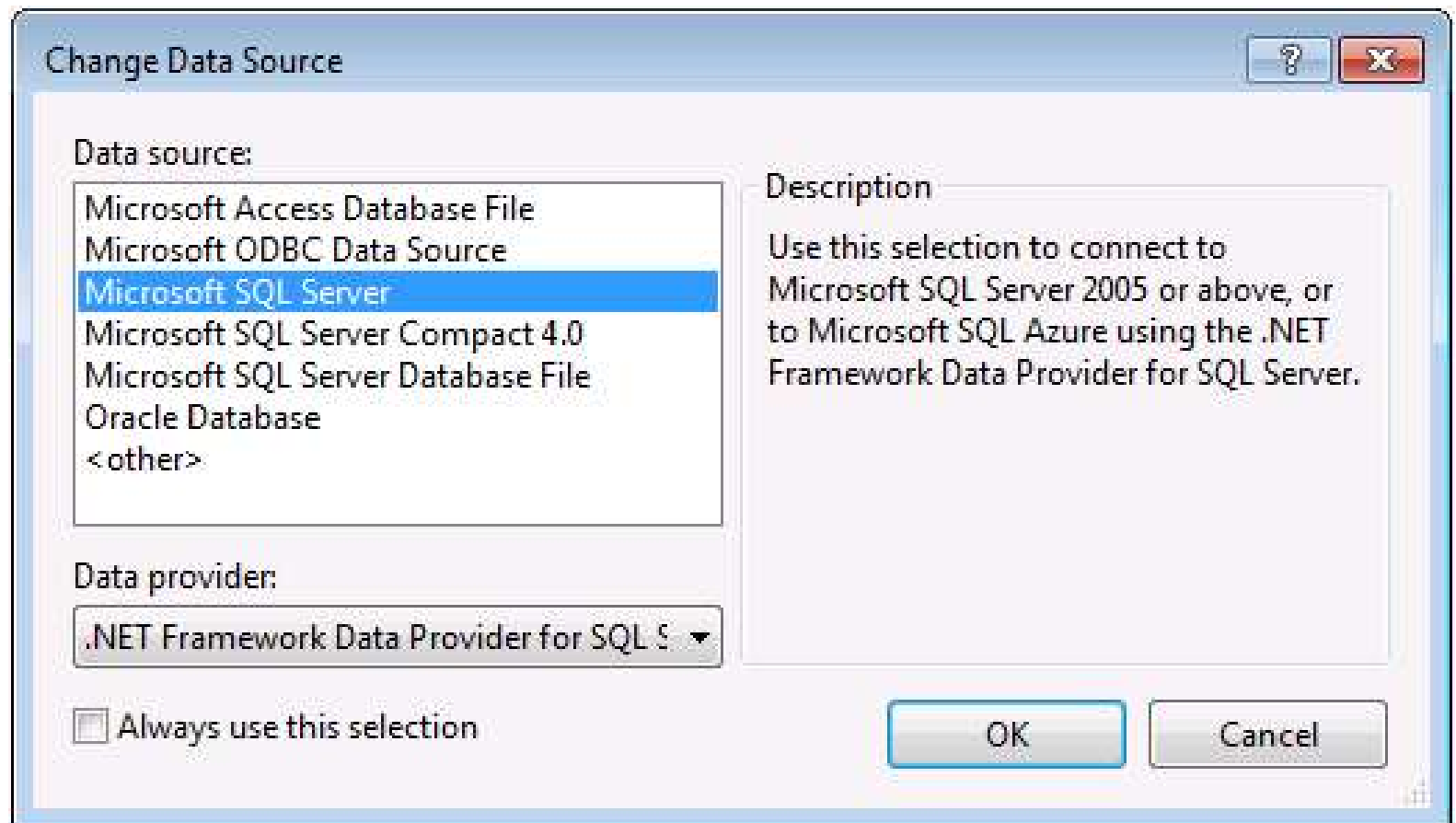
- Tên database đặt theo tên của Context, trong ví dụ trên là

**CodeFirstNewDatabaseSample.BloggingContext**

# Code First to New Database

## 6. Kết nối với database

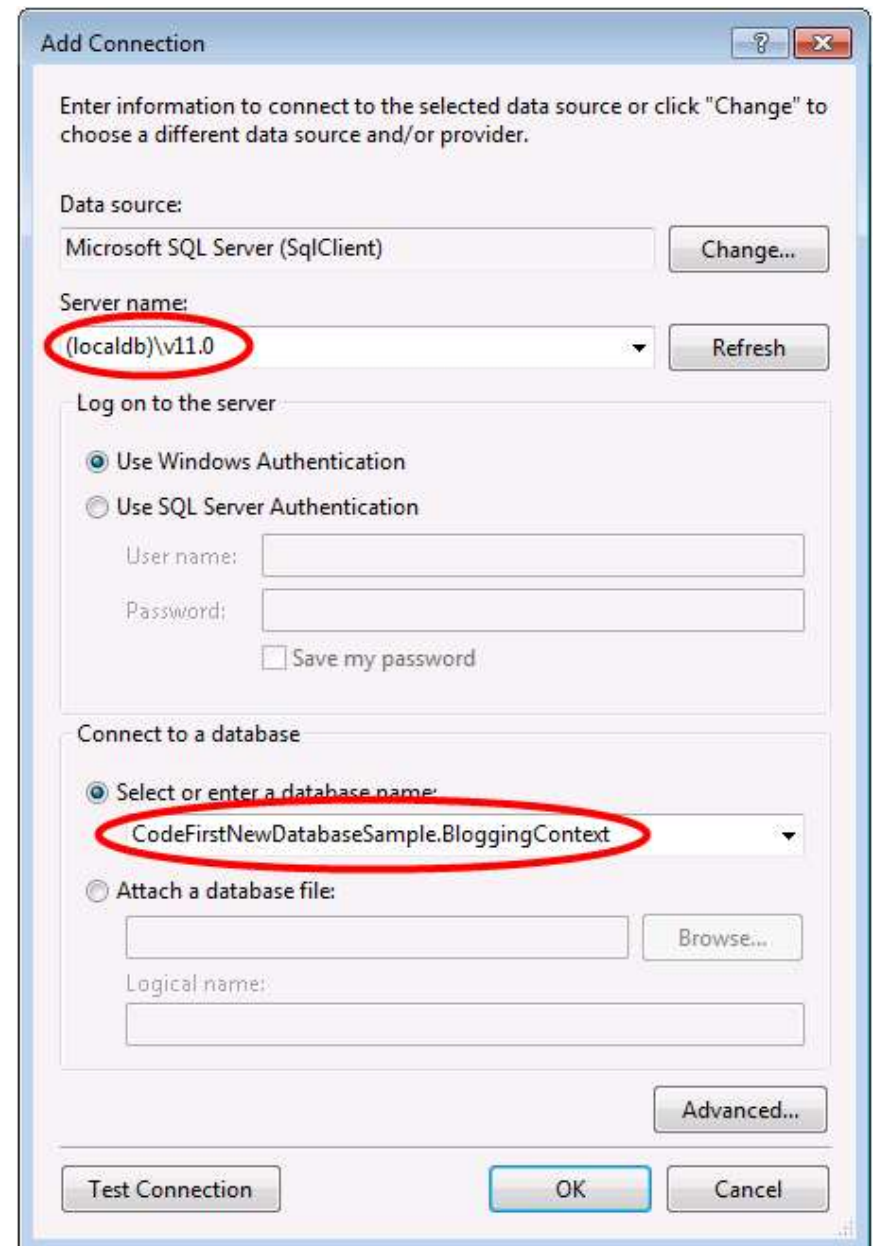
- **View -> Server Explorer. Chuột phải Data Connections chọn Add Connection...**



# Code First to New Database

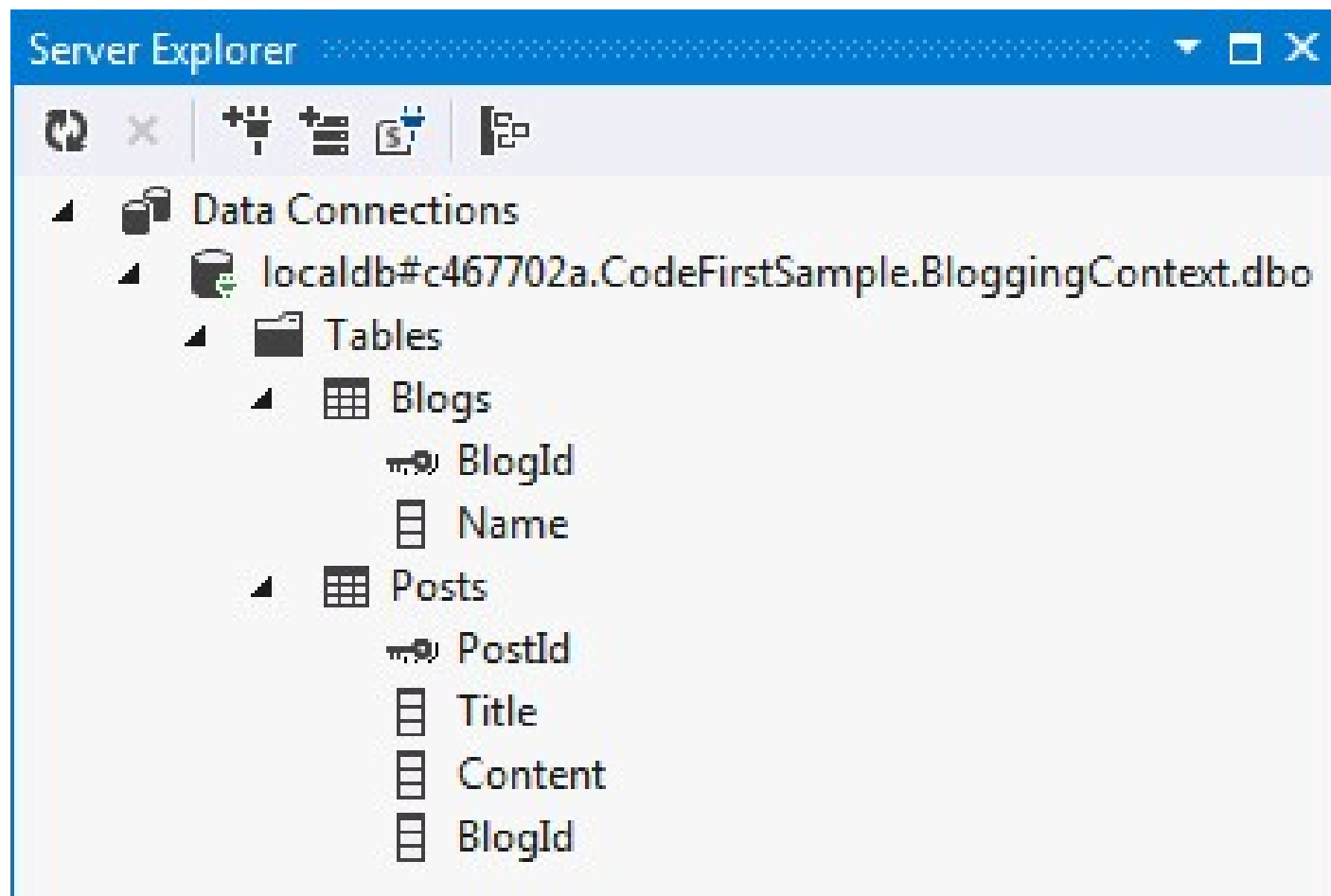
## 6. Kết nối với database

- Kết nối với LocalDb  
((localdb)\v11.0) hay  
SQL Express  
(.\SQLEXPRESS), tùy  
vào cấu hình



# Code First to New Database

## 6. Kết nối với database



# Code First to New Database

## 7. Vấn đề thay đổi mô hình

Trong trường hợp muốn thay đổi hay nâng cấp mô hình thì dùng tính năng **Code First Migrations**

# Code First to New Database

## 7. Vấn đề thay đổi mô hình

Các bước thay đổi:

- **Tools -> Library Package Manager -> Package Manager Console**
- Chạy lệnh **Enable-Migrations** ở Package Manager Console

# Code First to New Database

## 7. Vấn đề thay đổi mô hình

Các bước thay đổi:

- Sau đó một thư mục mới có tên **Migrations** được thêm vào dự án gồm:
  - **Configuration.cs**: chứa các thiết lập dùng cho tích hợp BloggingContext
  - **<timestamp>\_InitialCreate.cs**: đây là phiên bản tích hợp đầu tiên, thể hiện các thay đổi được áp dụng với database

# Code First to New Database

## 7. Vấn đề thay đổi mô hình

Thay đổi lớp Blog thêm một thuộc tính URL

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



# Code First to New Database

## 7. Vấn đề thay đổi mô hình

Chạy dòng lệnh **Add-Migration AddUrl** ở Package Manager Console.

Lệnh **Add-Migration** dùng để kiểm tra các thay đổi mô hình và cập nhật nếu thấy thay đổi. Chúng ta có thể đặt tên cho việc tích hợp, ví dụ tên là **AddUrl**.

# Code First to New Database

## 7. Vấn đề thay đổi mô hình

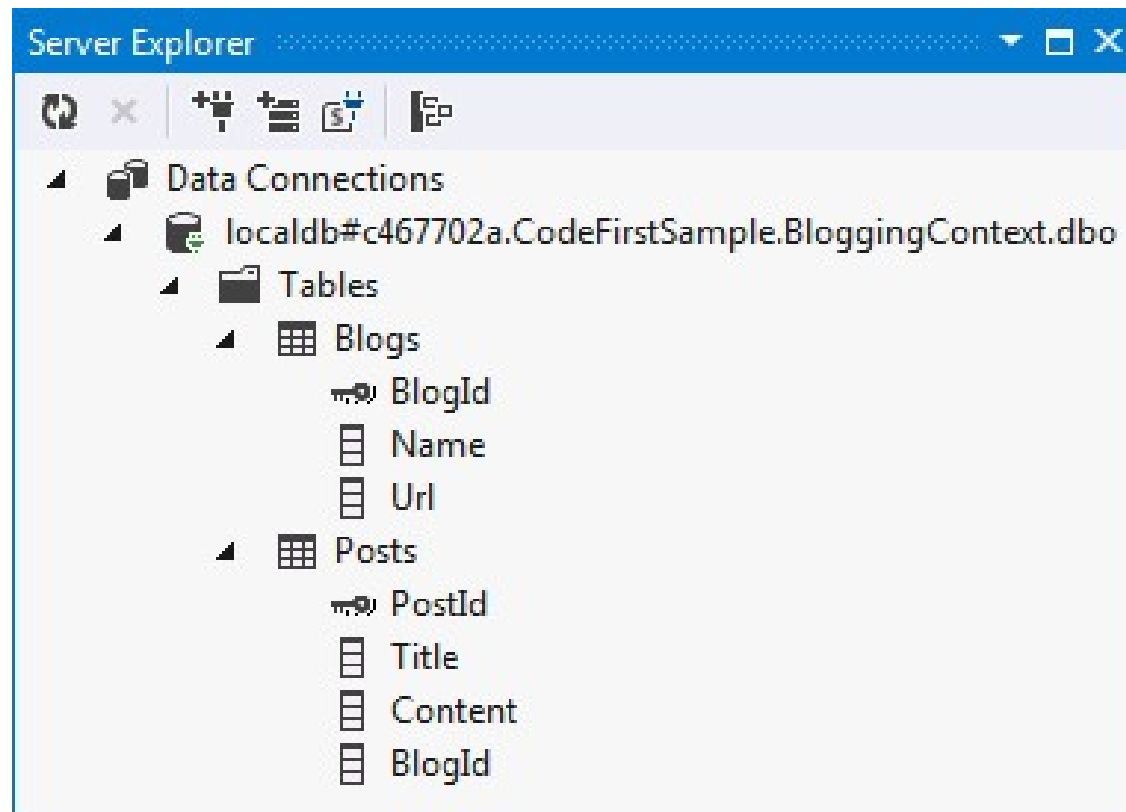
Sau khi dùng lệnh **Add-Migration** thì chương trình sản sinh ra lớp code

```
namespace CodeFirstNewDatabaseSample.Migrations
{
    using System; using System.Data.Entity.Migrations;
    public partial class AddUrl : DbMigration
    {
        public override void Up() {
            AddColumn("dbo.Blogs", "Url", c => c.String()); }
        public override void Down() {
            DropColumn("dbo.Blogs", "Url"); }
    }
}
```

# Code First to New Database

## 7. Vấn đề thay đổi mô hình

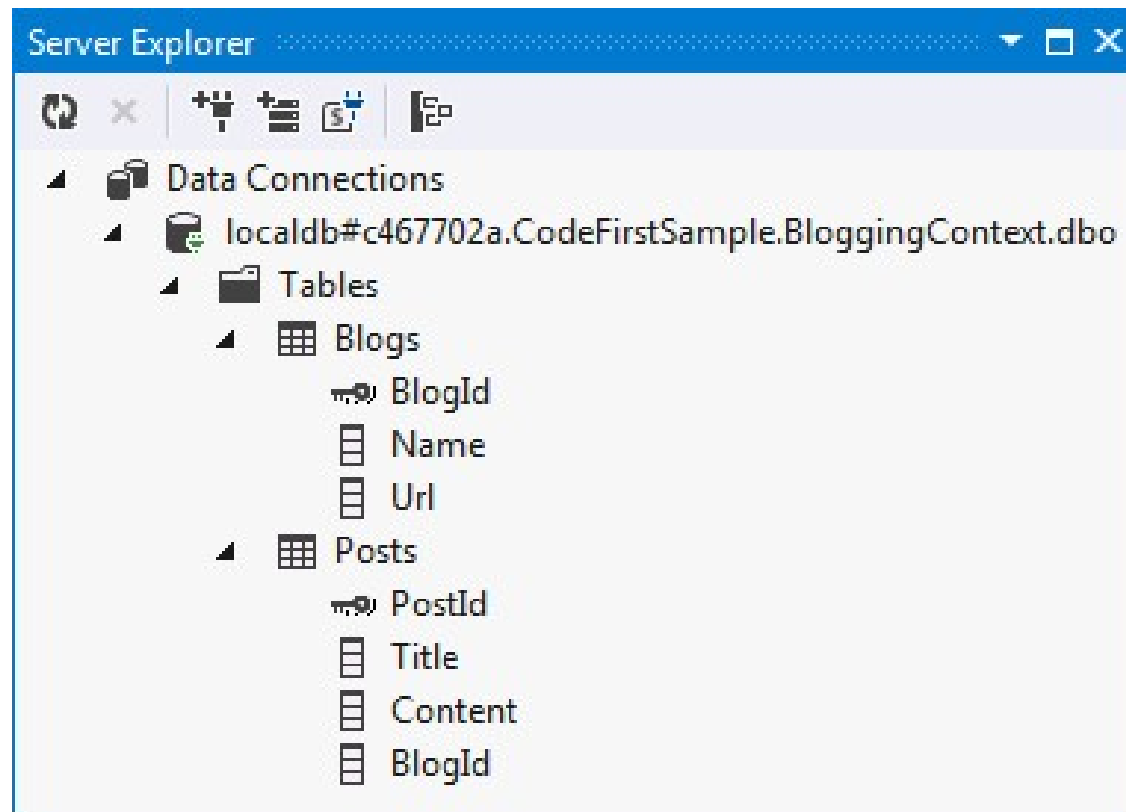
Tiếp đến chạy lệnh **Update-Database** để cập nhật ở phía database. Cuối cùng thuộc URL được thêm ở database ở bảng Blogs



# Code First to New Database

## 7. Vấn đề thay đổi mô hình

Tiếp đến chạy lệnh **Update-Database** để cập nhật ở phía database. Cuối cùng thuộc URL được thêm ở database ở bảng Blogs



# Code First to New Database

## 8. Chú thích dữ liệu (Data Annotations)

Chú thích dữ liệu trong .NET được định nghĩa ở **System.ComponentModel.DataAnnotations**

Lớp này cung cấp các lớp thuộc để định nghĩa dữ liệu siêu liên kết cho một thực thể nào đó.

# Code First to New Database

## 8. Chú thích dữ liệu (Data Annotations)

EF tạo database dựa theo mô hình code quy ước trước, trong trường hợp một số lớp, thuộc tính không theo quy ước thì lập trình viên phải tự thêm mã nguồn. Ví dụ thêm lớp User

```
public class User  
{  
    public string Username { get; set; }  
    public string DisplayName { get; set; }  
}
```

# Code First to New Database

## 8. Chú thích dữ liệu (Data Annotations)

Kế đến thêm dòng code xác định danh sách User ở Context

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

# Code First to New Database

## 8. Chú thích dữ liệu (Data Annotations)

Tiếp đến, dùng lệnh **Add-Migration** thì phát sinh lỗi *“EntityType ‘User’ has no key defined. Define the key for this EntityType.”* vì chưa xác định khóa chính.

Để xác định khóa chính, cần phải định nghĩa khóa chính ở lớp **User**.



# Code First to New Database

## 8. Chú thích dữ liệu (Data Annotations)

Thêm dòng lệnh ở Program.cs để xác định dùng namespace **DataAnnotations** cho dữ liệu

```
using System.ComponentModel.DataAnnotations;
```

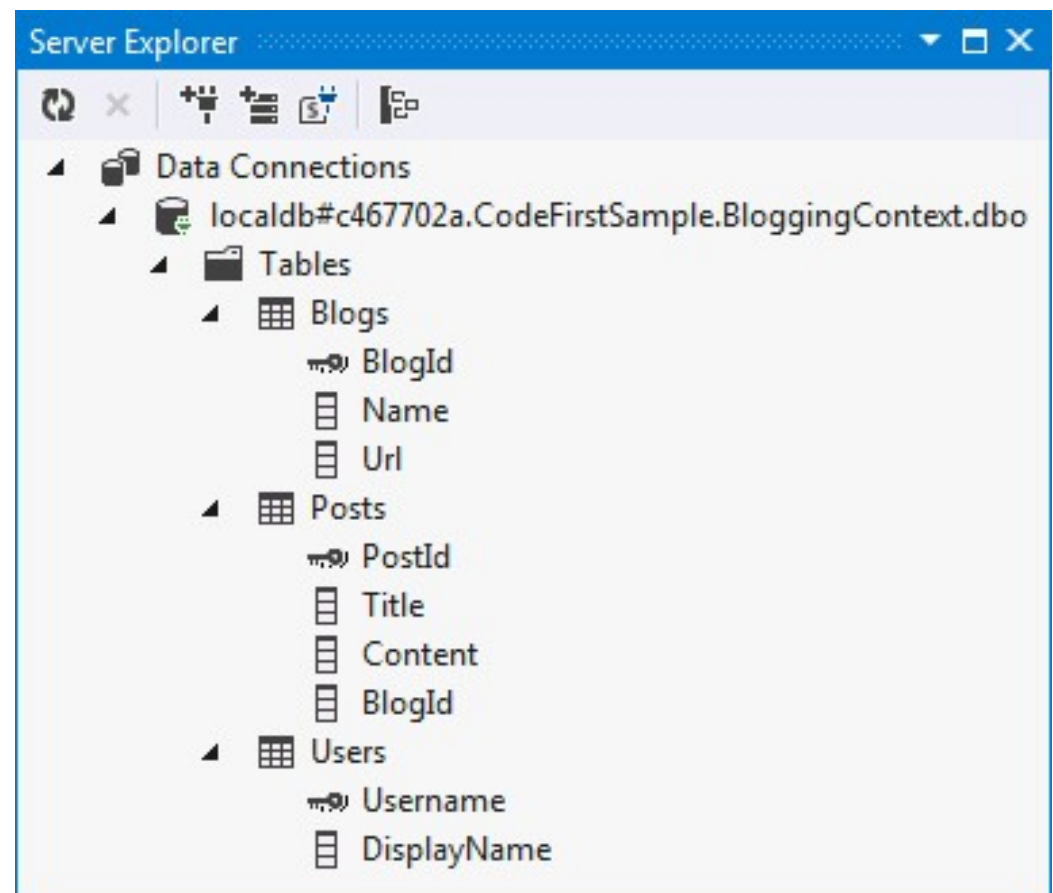
ở lớp User thì thêm

```
public class User
{
    [Key]
    public string Username { get; set; }
    public string DisplayName { get; set; }
}
```

# Code First to New Database

## 8. Chú thích dữ liệu (Data Annotations)

- Dùng 2 lệnh **Add-Migration AddUser** và **Update-Database** để tích hợp mô hình và cập nhật sự thay đổi vào database



# Code First to New Database

## 9. Fluent API

Ở phần **Data Annotations** cho phép bổ sung hoặc ghi đè mã nguồn theo quy ước. Ở phần này cũng là cách làm tương tự nhưng thông qua **Code First fluent API**.

# Code First to New Database

## 9. Fluent API

**Fluent API** cho phép đặc tả mô hình tốt hơn nhiều so với Data Annotations. Fluent API và Data Annotations có thể dùng cùng nhau.

Để sử dụng thì chỉ cần tạo phương thức override **OnModelCreating(DbModelBuilder modelBuilder)**

# Code First to New Database

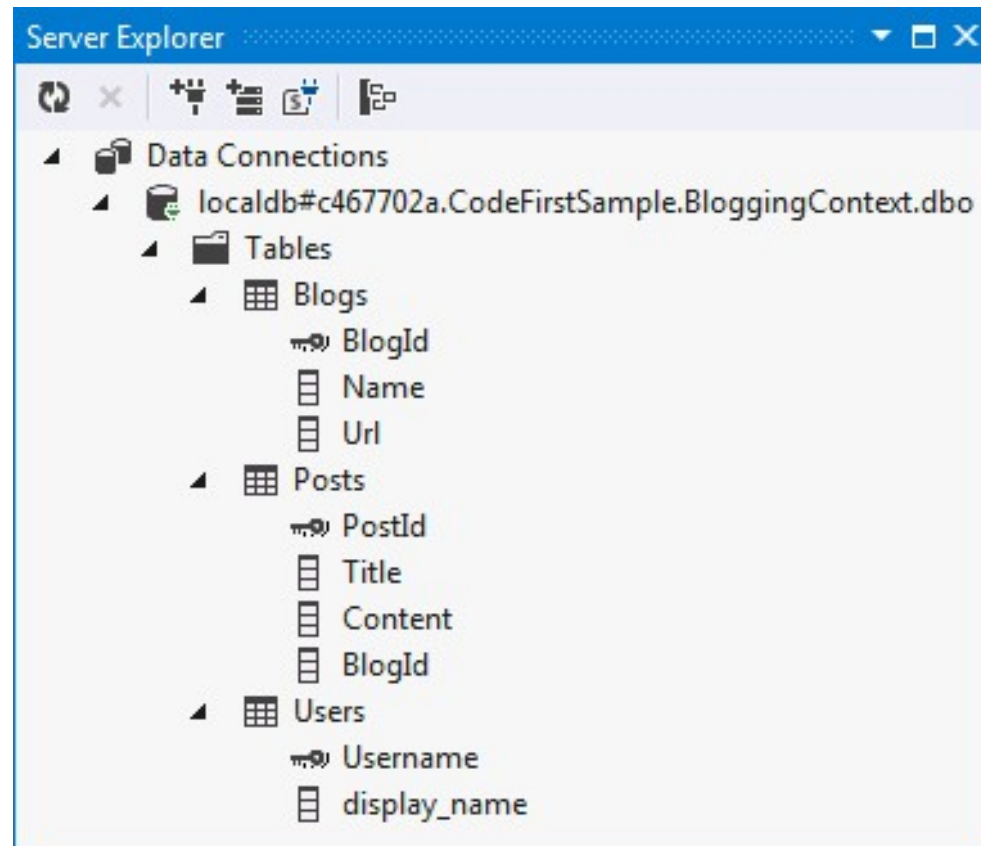
## 9. Fluent API

- **public class** BloggingContext : DbContext  
{  
    **public** DbSet<Blog> Blogs { get; set; }  
    **public** DbSet<Post> Posts { get; set; }  
    **public** DbSet<User> Users { get; set; }  
  
    **protected override void** OnModelCreating(DbModelBuilder modelBuilder)  
    {  
        modelBuilder.Entity<User>()  
            .Property(u => u.DisplayName)  
            .HasColumnName("display\_name");  
    }  
}

# Code First to New Database

## 9. Fluent API

Chạy lệnh **Add-Migration ChangeDisplayName** và **Update-Database** để tích hợp mô hình và cập nhật database



# Code First to New Database

## 10. Kết luận

- Lợi ích của Code First?
- Nhược điểm của Code First?

# Code First to New Database

## 10. Kết luận

- Lợi ích của Code First to New Database?
  - **Tốc độ phát triển dự án** – không cần phải thiết kế database mà chỉ cần viết code
  - **Simple** - không cần mô hình edmx để cập nhập hay duy trì



# Code First to New Database

## 10. Kết luận

- Nhược điểm Code First to New Database?
  - Viết code tạo đối tượng bằng tay
  - Việc thay đổi, duy trì dữ liệu ít chức năng hơn các phương pháp và chưa thể hiện trực quan

# Code First to an Existing Database

Cách này áp dụng cho một dự án đã có sẵn database, khi đó chỉ cần gieo mã nguồn dự án thông qua **Visual Studio**.

# Code First to an Existing Database

## 1. Tạo database

Tạo một database ở SQL Server bất kỳ, trong ví dụ này là mã script

```
CREATE TABLE [dbo].[Blogs] (  
    [BlogId] INT IDENTITY (1, 1) NOT NULL,  
    [Name] NVARCHAR (200) NULL,  
    [Url] NVARCHAR (200) NULL,  
    CONSTRAINT [PK_dbo.Blogs] PRIMARY KEY CLUSTERED ([BlogId] ASC)  
);
```

# Code First to an Existing Database

## 1. Tạo database

Tạo một database ở SQL Server bất kỳ, trong ví dụ này là mã script

```
CREATE TABLE [dbo].[Posts] (  
    [PostId] INT IDENTITY (1, 1) NOT NULL,  
    [Title] NVARCHAR (200) NULL,  
    [Content] NTEXT NULL,  
    [BlogId] INT NOT NULL,  
    CONSTRAINT [PK_dbo.Posts] PRIMARY KEY CLUSTERED ([PostId] AS  
C),  
    CONSTRAINT [FK_dbo.Posts_dbo.Blogs_BlogId] FOREIGN KEY ([Blog  
Id]) REFERENCES [dbo].[Blogs] ([BlogId]) ON DELETE CASCADE  
);
```

# Code First to an Existing Database

## 1. Tạo database

Tạo một database ở SQL Server bất kỳ, trong ví dụ này là mã script

```
INSERT INTO [dbo].[Blogs] ([Name],[Url])  
VALUES ('The Visual Studio Blog', 'http://blogs.msdn.com/visualstudio/')
```

```
INSERT INTO [dbo].[Blogs] ([Name],[Url])  
VALUES ('.NET Framework Blog', 'http://blogs.msdn.com/dotnet/')
```

# Code First to an Existing Database

## 2. Tạo ứng dụng

Tạo 1 ứng dụng console tên là  
**CodeFirstExistingDatabaseSample**, mã nguồn  
C#

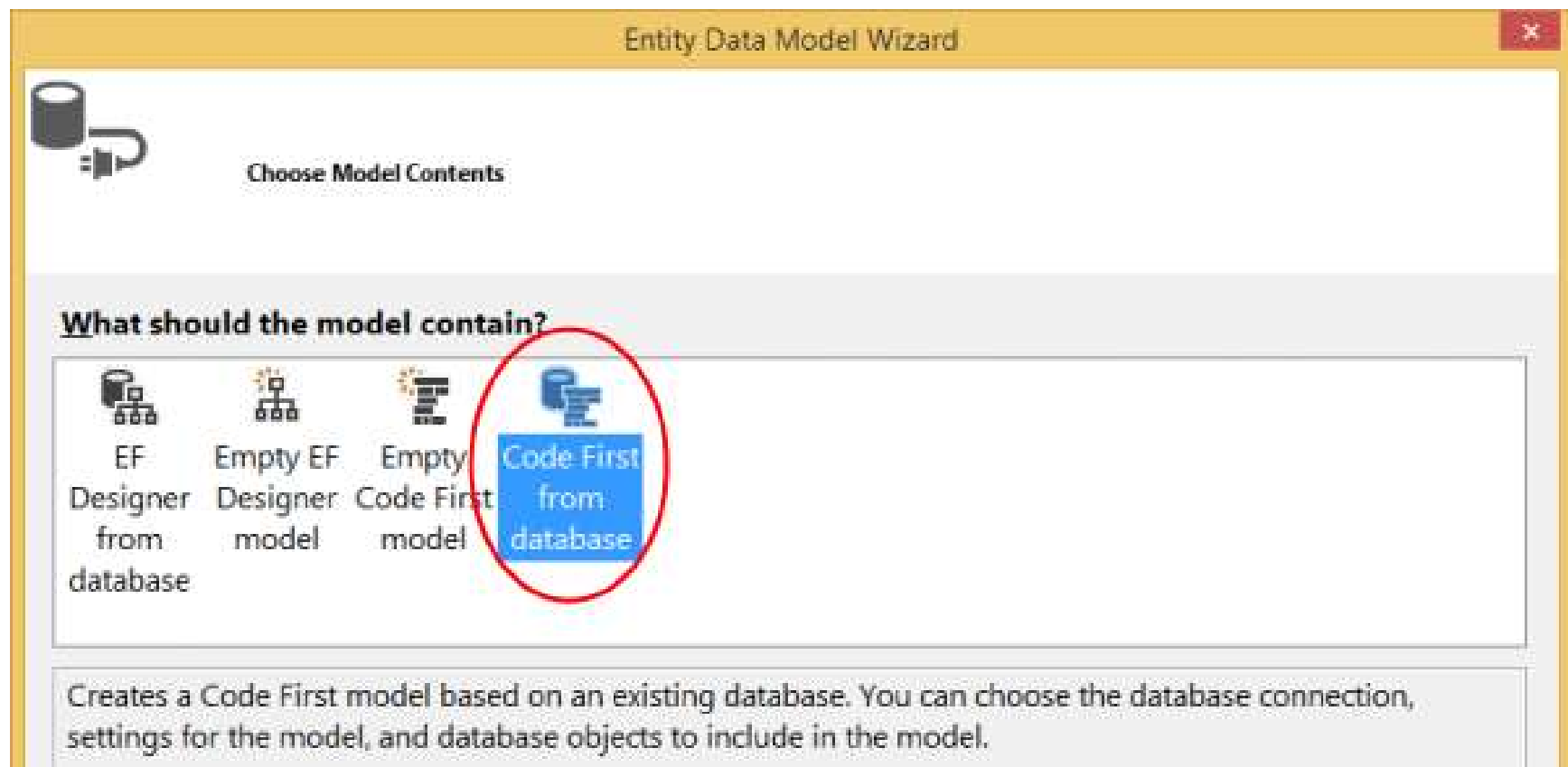
# Code First to an Existing Database

## 3. Gieo mã nguồn mô hình

- **Project -> Add New Item...**
- Chọn **Data** ở menu trái và **ADO.NET Entity Data Model**
- Ghi tên **BloggingContext** và nhấn **OK**
- Cửa sổ **Entity Data Model Wizard** được bật lên
- Chọn **Code First from Database** và chọn **Next**

# Code First to an Existing Database

## 3. Gieo mã nguồn mô hình

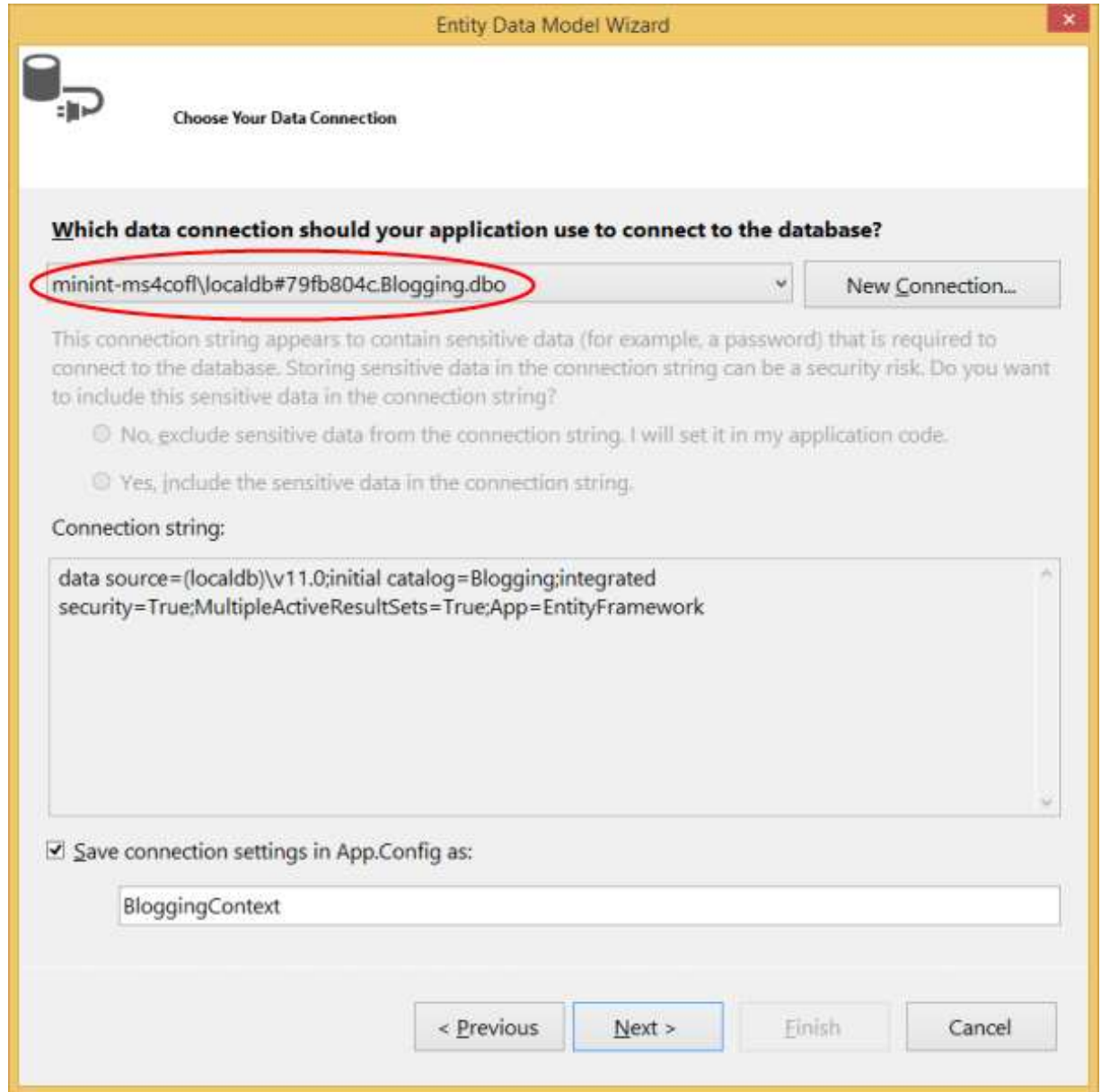




# Code First to an Existing Database

## 3. Gieo mã nguồn mô hình

- Chọn kết nối và tên kết nối sẽ được lưu ở App.Config

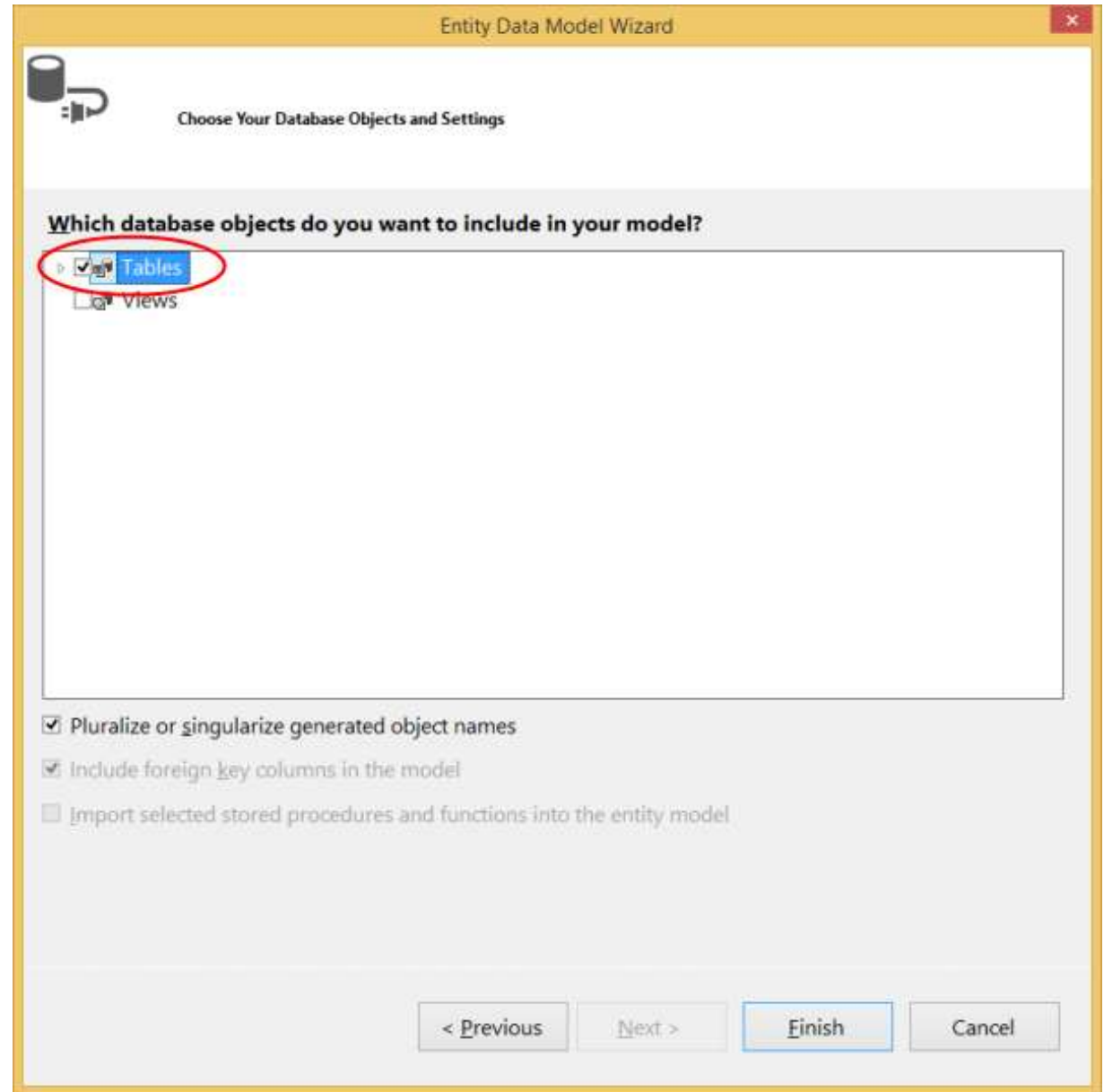


The image shows the 'Entity Data Model Wizard' dialog box, specifically the 'Choose Your Data Connection' step. The title bar reads 'Entity Data Model Wizard'. The main heading is 'Choose Your Data Connection'. Below this, a question asks: 'Which data connection should your application use to connect to the database?'. A dropdown menu shows the selected connection: 'minint-ms4cofl\localdb#79fb804c.Blogging.dbo', which is circled in red. To the right of the dropdown is a 'New Connection...' button. Below the dropdown, a warning message states: 'This connection string appears to contain sensitive data (for example, a password) that is required to connect to the database. Storing sensitive data in the connection string can be a security risk. Do you want to include this sensitive data in the connection string?'. There are two radio buttons: 'No, exclude sensitive data from the connection string. I will set it in my application code.' (selected) and 'Yes, include the sensitive data in the connection string.'. Below this, the 'Connection string:' is displayed in a text box: 'data source=(localdb)\v11.0;initial catalog=Blogging;integrated security=True;MultipleActiveResultSets=True;App=EntityFramework'. At the bottom, there is a checkbox 'Save connection settings in App.Config as:' which is checked. Below the checkbox is a text box containing 'BloggingContext'. At the very bottom, there are four buttons: '< Previous', 'Next >', 'Finish', and 'Cancel'.

# Code First to an Existing Database

## 3. Gieo mã nguồn mô hình

- Chọn bảng dữ liệu cần gieo mã



# Code First to an Existing Database

## 4. Tập tin cấu hình App.config

```
<connectionStrings>  
  <add  
    name="BloggngContext"  
    connectionString="data source=(localdb)\v11.0;ini  
tial catalog=Bloggng;integrated security=True;Multip  
leActiveResultSets=True;App=EntityFramework"  
    providerName="System.Data.SqlClient" />  
</connectionStrings>
```

# Code First to an Existing Database

## 5. Lớp Context

```
public partial class BloggingContext : DbContext
{
    public BloggingContext()
        : base("name=BloggingContext")
    {
    }

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

    protected override void OnModelCreating(DbModelBuild
er modelBuilder)
    {
    }
}
```

# Code First to an Existing Database

## 6. Mô hình các lớp

```
public partial class Blog
{
    public Blog() { Posts = new HashSet<Post>(); }

    public int BlogId { get; set; }

    [StringLength(200)]
    public string Name { get; set; }

    [StringLength(200)]
    public string Url { get; set; }

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

# Code First to an Existing Database

## 7. Đọc ghi dữ liệu

Tương tự như Code First to New Database

## 8. Các phần khác, tham khảo thêm:

<https://msdn.microsoft.com/en-us/data/jj200620>

# Bài tập

- Thực hiện ví dụ bài giảng
- Thử cách Code First to Existing Database, gieo bảng `_MigrationHistory` thành lớp code, sau đó chỉnh sửa thuộc tính ở bảng bất kỳ, thử cập nhật từ code.
- Email: [tahoangthang@gmail.com](mailto:tahoangthang@gmail.com)
- Số điện thoại: 01689797946