

# Data Access

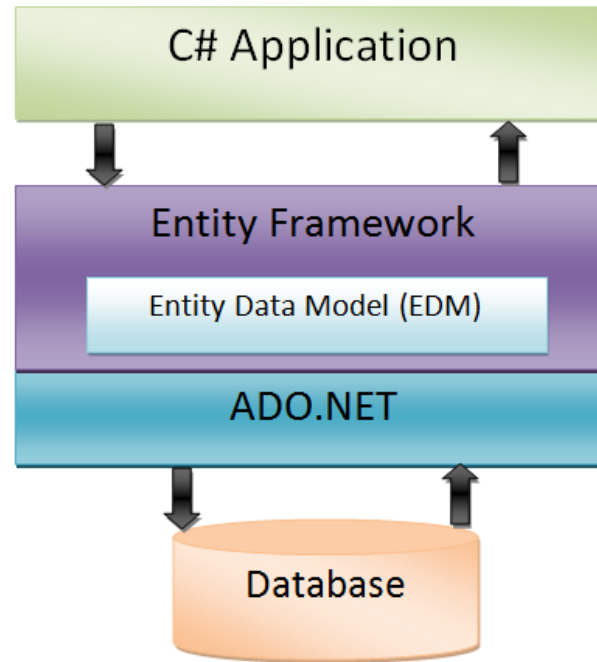
- ◆ Define and describe the Entity Framework
- ◆ Explain how to work with the Entity Framework
- ◆ Define and describe how to initialize a database with sample data
- ◆ Explain how to use LINQ queries to perform database operations

# Entity Framework

- ◆ To address the data access requirements of ASP.NET MVC application, you can use an ORM framework.
- ◆ An ORM framework:
  - ◆ Simplifies the process of accessing data from applications.
  - ◆ Performs the necessary conversions between incompatible type systems in relational databases and object-oriented programming languages.
- ◆ The Entity Framework is an ORM framework that ASP.NET MVC applications can use.
- ◆ The Entity Framework is an implementation of the Entity Data Model (EDM), which is a conceptual model that describes the entities and the associations they participate in an application.
- ◆ EDM allows you to handle data access logic by programming against entities

# Entity Framework

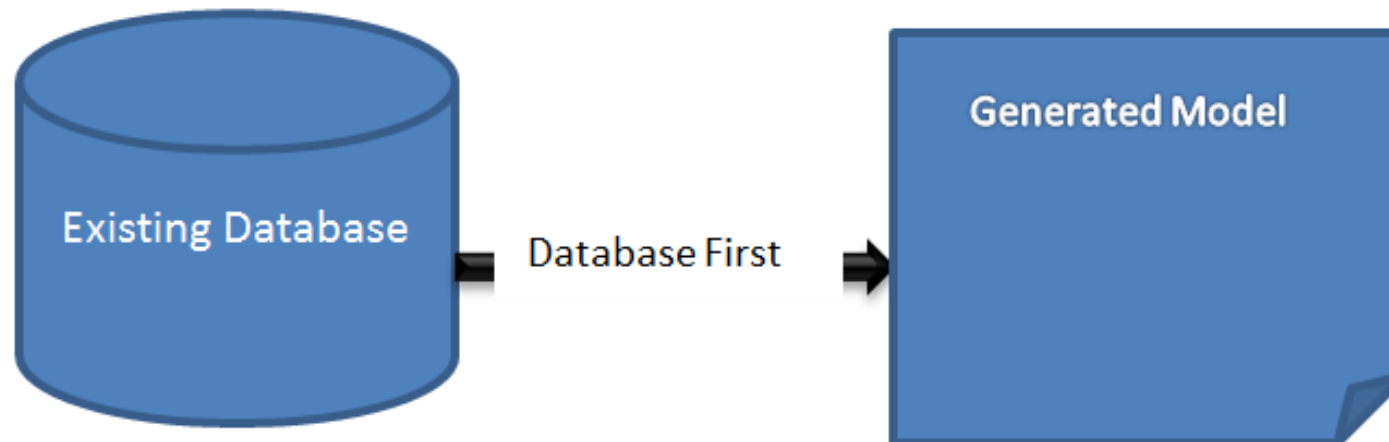
- ◆ Following figure shows the role of EDM in the Entity Framework architecture:



- ◆ Entity Framework:
  - ◆ Eliminates the need to write most of the data-access code that otherwise need to be written.
  - ◆ Uses different approaches, such as database-first and code-first to manage data related to an application.

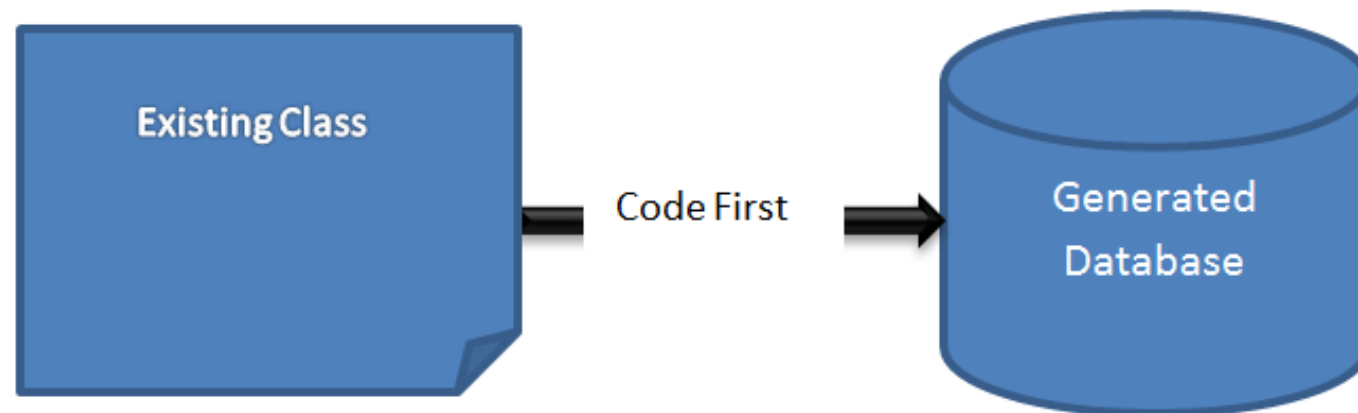
# Database-first Approach

- ◆ In the database-first approach the Entity Framework creates model classes and properties corresponding to the existing database objects, such as tables and columns.
- ◆ The database-first approach is applicable in scenario where a database already exists for the application.
- ◆ Following figure shows the database-first approach:

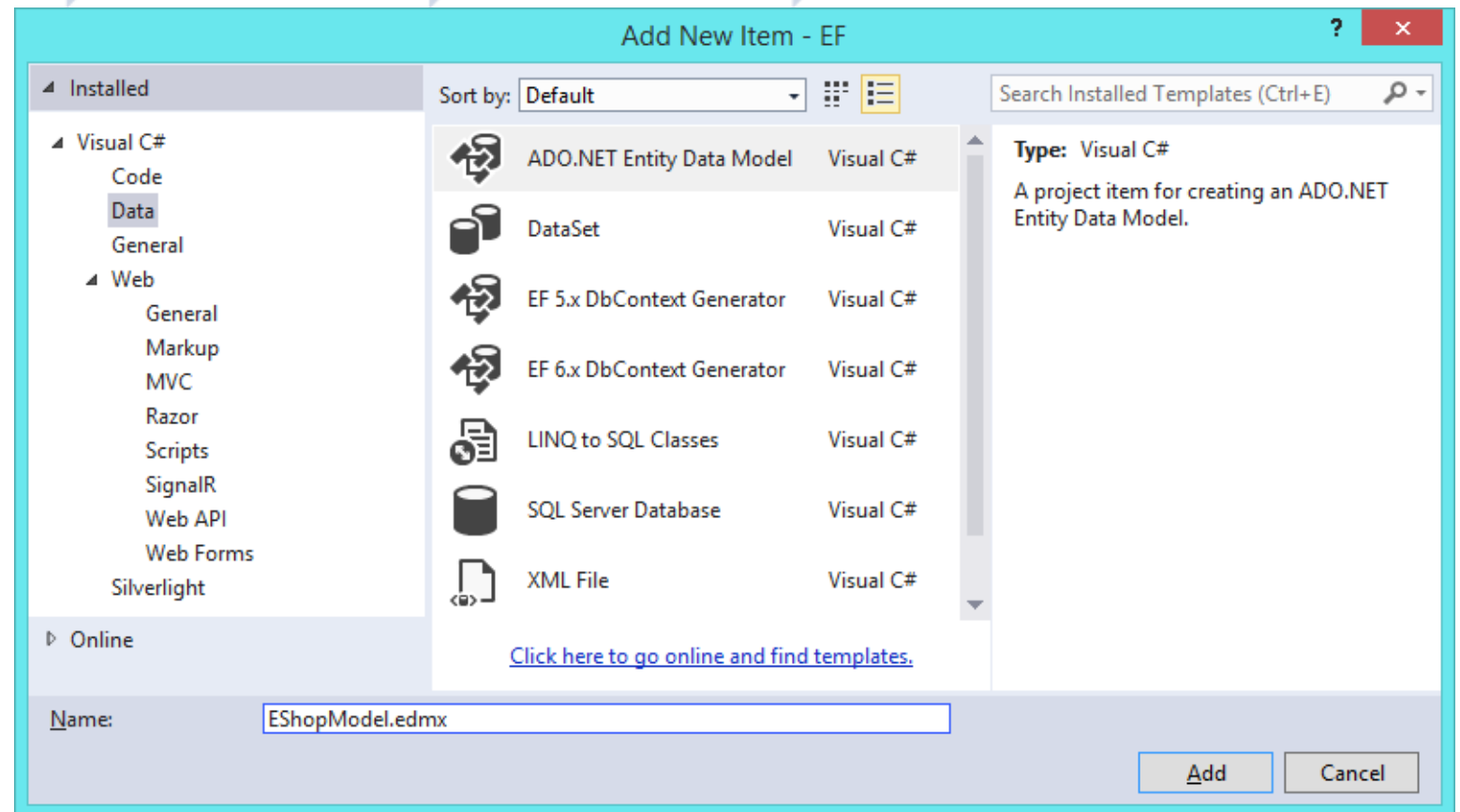
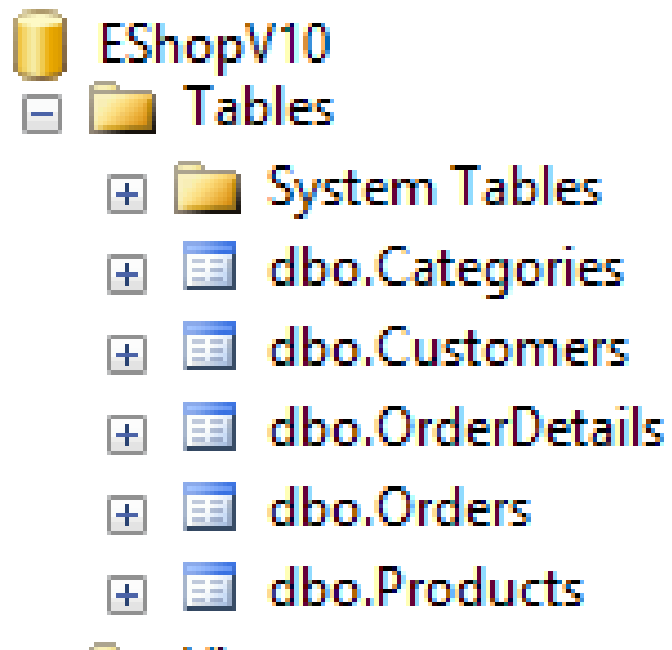
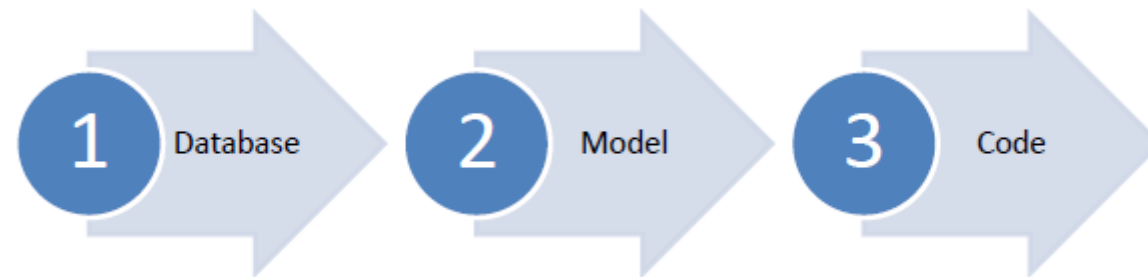


# Code-first Approach

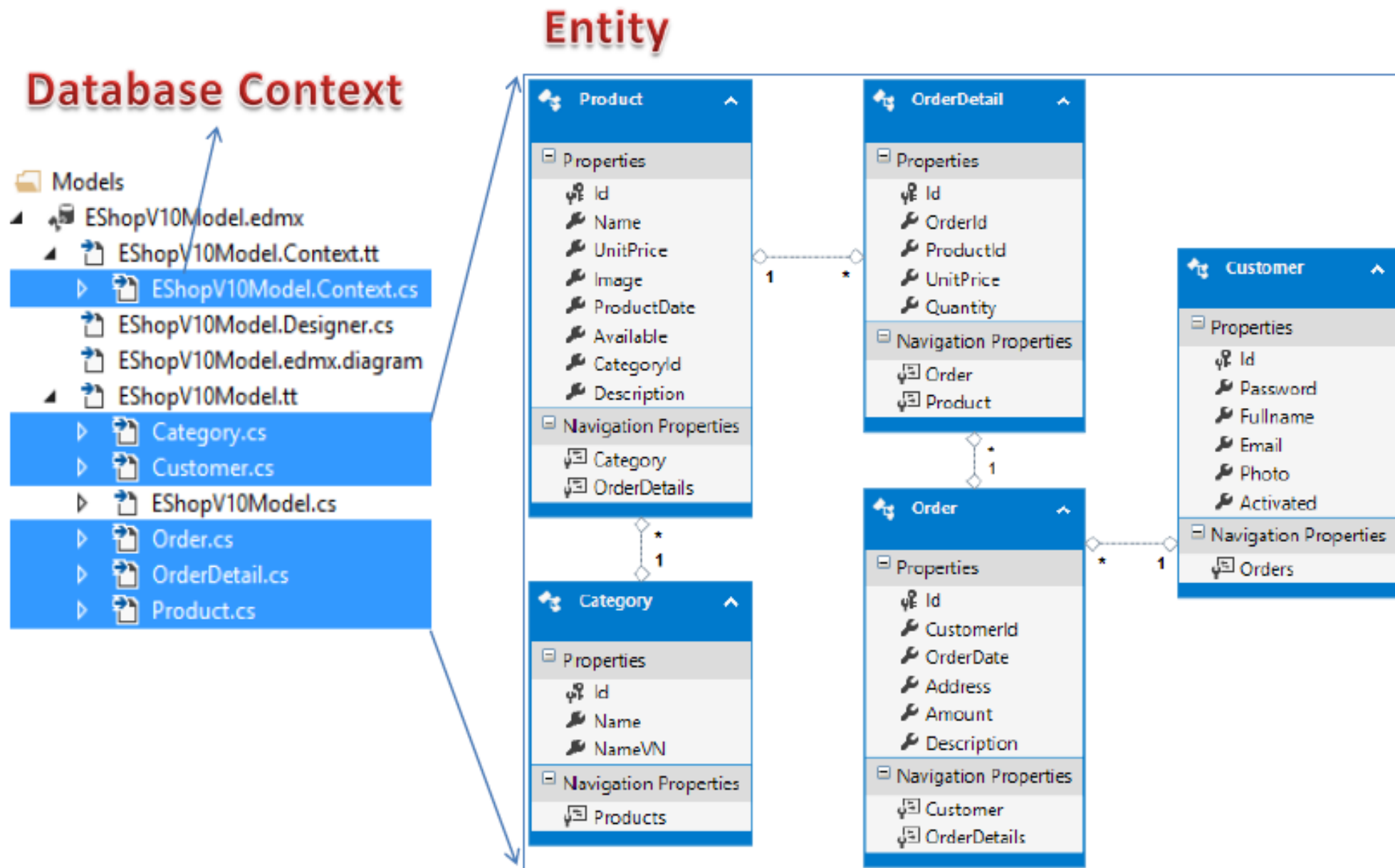
- ◆ In the code-first approach the Entity Framework creates database objects based on model classes that you create to represent application data.
- ◆ The code-first approach:
  - ◆ Is the most common approach implemented in ASP.NET MVC Framework.
  - ◆ Allows you to develop your application by coding model classes and properties and delegate the process of creating the database objects to the Entity Framework.
- ◆ Following figure shows the code-first approach:



# Database-first Approach



# Database model






# Database context

```
public partial class EShopV10 : DbContext
{
    public EShopV10()
        : base("name=EShopV10")
    {
    }

    public DbSet<Category> Categories { get; set; }
    public DbSet<Customer> Customers { get; set; }
    public DbSet<OrderDetail> OrderDetails { get; set; }
    public DbSet<Order> Orders { get; set; }
    public DbSet<Product> Products { get; set; }
}
```



```
<connectionStrings>
  <add name="EShopV10" connectionString="..."
        providerName="System.Data.EntityClient" />
</connectionStrings>
```

# Chi tiết thực thể

- ◆ Tên thực thể <=> Tên bảng
- ◆ Tên thuộc tính <=> Tên cột
- ◆ Thực thể kết hợp 1-Nhiều

```
public partial class Category
{
    public Category()
    {
        this.Products = new HashSet<Product>();
    }

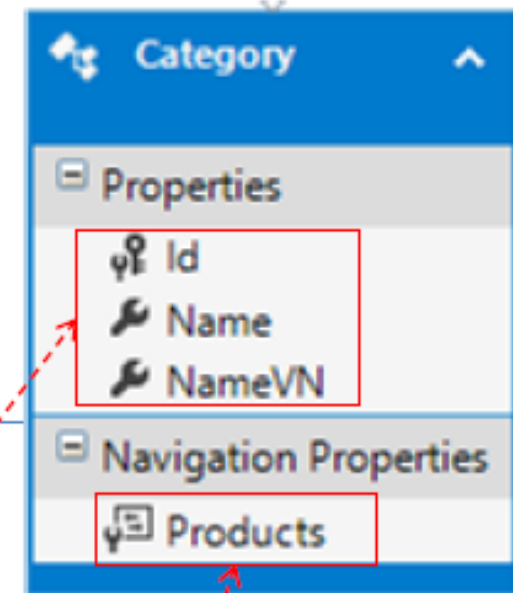
```

```
    public int Id { get; set; }
    public string Name { get; set; }
    public string NameVN { get; set; }

```

```
    public virtual ICollection<Product> Products { get; set; }
}

```

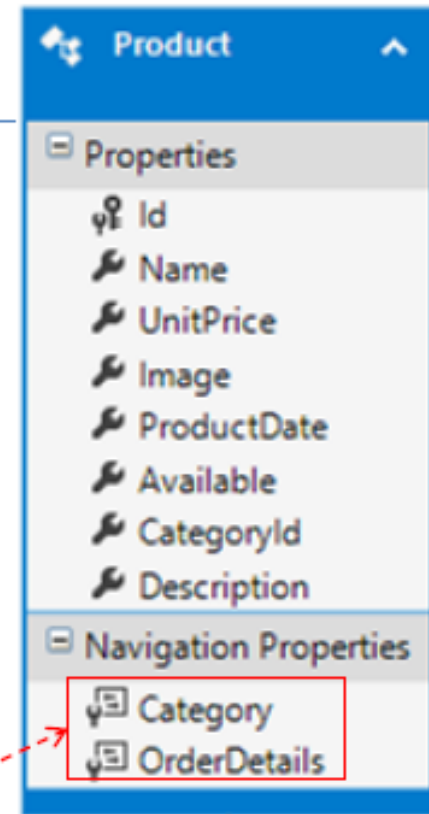


# Thực thể kết hợp

```
public partial class Product
{
    public Product()
    {
        this.OrderDetails = new HashSet<OrderDetail>();
    }

    public int Id { get; set; }
    public string Name { get; set; }
    public double UnitPrice { get; set; }
    public string Image { get; set; }
    public System.DateTime ProductDate { get; set; }
    public bool Available { get; set; }
    public int CategoryId { get; set; }
    public string Description { get; set; }

    public virtual Category Category { get; set; }
    public virtual ICollection<OrderDetail> OrderDetails { get; set; }
}
```



# Entity Framework API

## ◆ Tạo đối tượng db\_context

```
var db = new EShopV10();
```

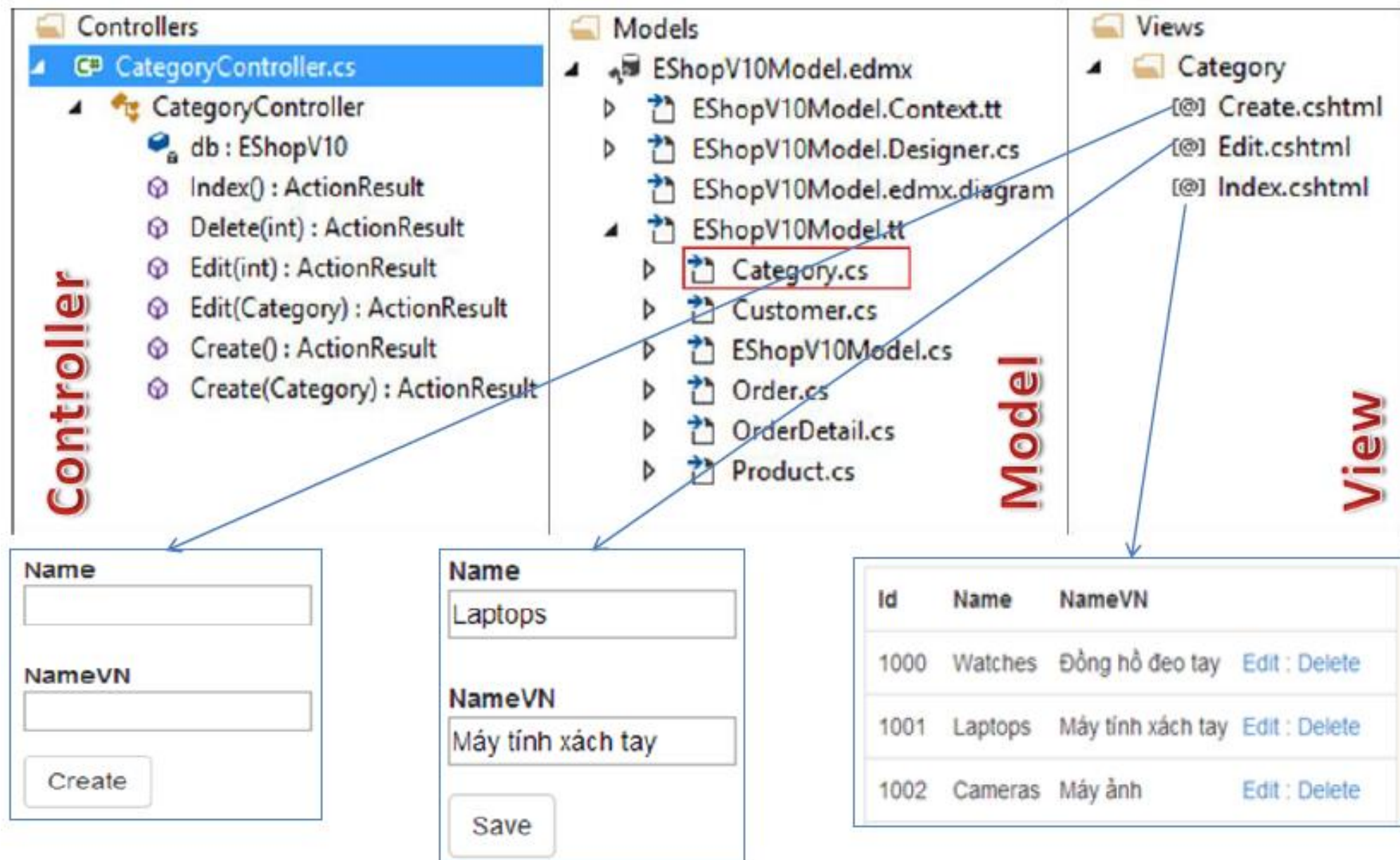
## ◆ Thao tác & truy vấn thực thể

Thêm mới thực thể	db.Categories. <b>Add</b> (category)
Cập nhật thông tin của thực thể	db. <b>Entry</b> (category). <b>State</b> = EntityState.Modified
Xóa thực thể	db.Categories. <b>Remove</b> (category)
Truy vấn một thực thể theo mã	var entity = db.Categories. <b>Find</b> (id)
Truy vấn tất cả các thực thể	var list = db. <b>Categories</b>

## ◆ Lưu sự thay đổi

```
db.SaveChanges();
```

## Ví dụ



# Sử dụng Create template

```
// GET:/Category/Create
public ActionResult Create()
{
    return View();
}

// POST:/Category/Create
[HttpPost]
public ActionResult Create(Category model)
{
    // Thêm mới
    db.Categories.Add(model);
    db.SaveChanges();
    return RedirectToAction("Index");
}
```

Name

NameVN

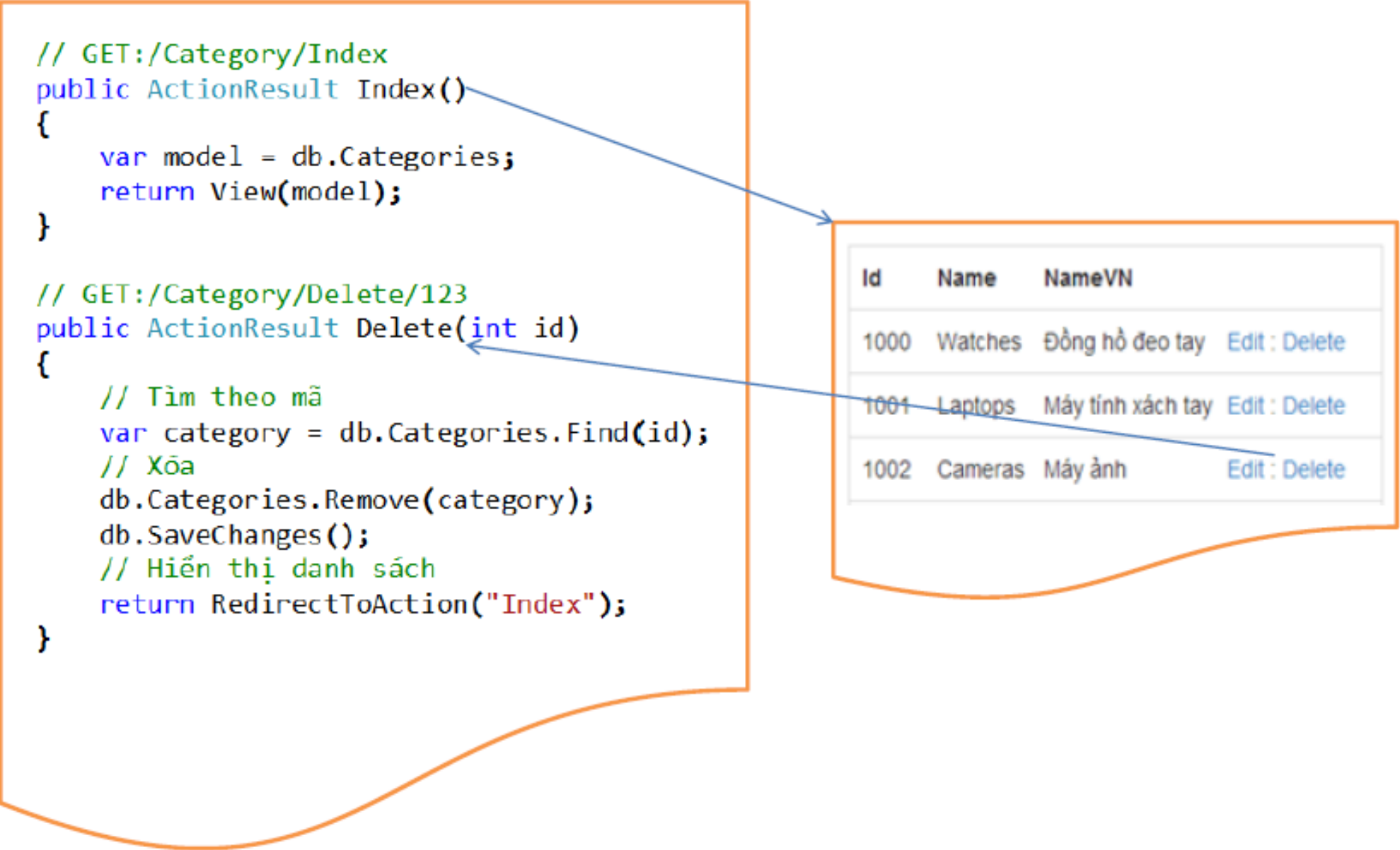
Create



# Sử dụng List và Delete template

```
// GET:/Category/Index
public ActionResult Index()
{
    var model = db.Categories;
    return View(model);
}

// GET:/Category/Delete/123
public ActionResult Delete(int id)
{
    // Tìm theo mã
    var category = db.Categories.Find(id);
    // Xóa
    db.Categories.Remove(category);
    db.SaveChanges();
    // Hiển thị danh sách
    return RedirectToAction("Index");
}
```



The diagram illustrates the relationship between the code and the UI. A blue arrow points from the `Index()` method to the table, indicating it provides the data. Another blue arrow points from the `Delete(int id)` method to the row with `Id` 1001, indicating it handles the deletion of that specific record. An orange bracket at the bottom groups the code and the table together.

Id	Name	NameVN	
1000	Watches	Đồng hồ đeo tay	<a href="#">Edit</a> : <a href="#">Delete</a>
1001	Laptops	Máy tính xách tay	<a href="#">Edit</a> : <a href="#">Delete</a>
1002	Cameras	Máy ảnh	<a href="#">Edit</a> : <a href="#">Delete</a>

# Sử dụng Edit template

```
// GET:/Category/Edit/123
public ActionResult Edit(int id)
{
    var model = db.Categories.Find(id);
    return View(model);
}

// POST:/Category/Edit
[HttpPost]
public ActionResult Edit(Category model)
{
    db.Entry(model).State = EntityState.Modified;
    db.SaveChanges();
    return View(model);
}
```

Id	Name	NameVN	
1000	Watches	Đồng hồ đeo tay	<a href="#">Edit</a> : <a href="#">Delete</a>
1001	Laptops	Máy tính xách tay	<a href="#">Edit</a> : <a href="#">Delete</a>
1002	Cameras	Máy ảnh	<a href="#">Edit</a> : <a href="#">Delete</a>

Name

NameVN



# Code-first Approach



- ☐ Entity
  - ✓ Category
  - ✓ Product
  - ✓ Customer
  - ✓ Order
  - ✓ OrderDetail
- ☐ Database Context
  - ✓ EShopDbContext
  - ✓ Web.config

- 
- A screenshot of the SQL Server Enterprise Manager interface showing the structure of the EShopV10 database. The 'Tables' folder is expanded, showing a list of tables in the 'dbo' schema. The tables listed are: System Tables, dbo.Categories, dbo.Customers, dbo.OrderDetails, dbo.Orders, and dbo.Products. There are plus signs next to each table name, indicating they can be expanded to show their schema details.
- EShopV10
    - Tables
      - System Tables
      - dbo.Categories
      - dbo.Customers
      - dbo.OrderDetails
      - dbo.Orders
      - dbo.Products
      - ...

# Tạo các Thực thể (Entity)

```
public class Product
{
    public int Id { get; set; }
    public string Name { get; set; }
    public double UnitPrice { get; set; }
    public string Image { get; set; }
    public DateTime ProductDate { get; set; }
    public bool Available { get; set; }
    public int CategoryId { get; set; }
    public string Description { get; set; }

    public virtual Category Category { get; set; }
    public virtual List<OrderDetail> OrderDetails { get; set; }
}

public class Category
{
    public int Id { get; set; }
    public string Name { get; set; }
    public string NameVN { get; set; }

    public virtual List<Product> Products { get; set; }
}

public class Order
{
    public int Id { get; set; }
    public string CustomerId { get; set; }
    public DateTime OrderDate { get; set; }
    public string Address { get; set; }
    public double Amount { get; set; }
    public string Description { get; set; }

    public virtual Customer Customer { get; set; }
    public virtual List<OrderDetail> OrderDetails { get; set; }
}

public class OrderDetail
{
    public int Id { get; set; }
    public int OrderId { get; set; }
    public int ProductId { get; set; }
    public double UnitPrice { get; set; }
    public int Quantity { get; set; }

    public virtual Order Order { get; set; }
    public virtual Product Product { get; set; }
}

public class Customer
{
    public string Id { get; set; }
    public string Name { get; set; }
    public string NameVN { get; set; }
    public string Address { get; set; }
    public string Phone { get; set; }
    public string Email { get; set; }

    public virtual List<Order> Orders { get; set; }
}
```

# Database Context

- ◆ using namespace System.Data.Entity;

```
public class EShopV10 : DbContext
{
    public EShopV10() : base("name=EShopV10") { }

    public DbSet<Category> Categories { get; set; }
    public DbSet<Customer> Customers { get; set; }
    public DbSet<OrderDetail> OrderDetails { get; set; }
    public DbSet<Order> Orders { get; set; }
    public DbSet<Product> Products { get; set; }
}
```

```
<connectionStrings>
  <add name="EShopV10"
    connectionString="Server=.;Database=EShopV10;Integrated Security=True"
    providerName="System.Data.SqlClient" />
</connectionStrings>
```

# Quy ước Ánh xạ Thực thể

- ◆ Tên thực thể sẽ ánh xạ với bảng cùng tên
  - ◆ Tùy biến bằng: [Table<tên bảng>]
- ◆ Tên thuộc tính cùng tên với cột
  - ◆ Tùy biến bằng: [Column<tên cột>]
- ◆ Tên thuộc tính khóa phải là Id hoặc EntityId
  - ◆ Tùy biến bằng: [Key]
- ◆ Khóa int được hiểu là tự tăng
  - ◆ Tùy biến với: [DatabaseGenerated(DatabaseGeneratedOption.Identity)]

```
[Table("KhoaHoc")]
public class Course
{
    [Key]
    [Column("MaKH")]
    public int Id { get; set; }
    [Column("TenKH")]
    public String Name { get; set; }
    [Column("HocPhi")]
    public double UnitPrice { get; set; }
}
```

# Khởi tạo dữ liệu

```
public class MvcApplication : System.Web.HttpApplication
{
    protected void Application_Start()
    {
        Database.SetInitializer(new MusicStoreDbInitializer());
    }
}
```



```
public class MusicStoreDbInitializer
    : DropCreateDatabaseIfModelChanges<MusicStoreDbContext>
{
    protected override void Seed(MusicStoreDbContext context)
    {
        base.Seed(context);
        var genres = new List<Genre>
        {
            new Genre { Name = "Rock" },
        };
        var artists = new List<Artist>
        {
            new Artist { Name = "Aaron Copland & London Symphony Orchestra" },
        };
        var albums = new List<Album>
        {
            new Album { Title = "A Copland Celebration, Vol. I", Genre :
            };
        };
        albums.ForEach(a => context.Albums.Add(a));
    }
}
```

# Truy vấn bằng LINQ

## ◆ LINQ: **L**anguage **I**Ntegrated **Q**uery

- ◆ Là API cho phép tạo các câu truy vấn độc lập với nguồn dữ liệu.
- ◆ Là một mô hình lập trình cung cấp một cú pháp chuẩn cho việc truy vấn vào các nguồn dữ liệu khác nhau.
- ◆ Các câu truy vấn được tạo ra có thể được hỗ trợ bởi các ngôn ngữ lập trình thuộc về .Net Framework như: VB, C#...
- ◆ Hỗ trợ thao tác với dữ liệu nhanh chóng vào nhiều nguồn dữ liệu khác nhau mà không cần phải học thêm các ngôn ngữ truy vấn khác.

# Hai dạng thức LINQ

## ◆ Dạng truy vấn

```
var result = from s in students  
where s.Marks > 9  
orderby s.Marks descending  
select new { s.Name, s.Marks };
```

Biểu thức  
truy vấn

Kiểu nội bộ  
tự suy

Kiểu nặc danh

Khởi tạo đối tượng

## ◆ Dạng phương thức

```
var result2 = students  
    .Where(s => s.Marks > 9)  
    .OrderByDescending(s => s.Marks)  
    .Select(s => new { s.Name, s.Marks });
```

Biểu thức  
lambda

Phương thức  
mở rộng

# Sử dụng truy vấn LINQ

- ◆ Trong ứng dụng ASP.Net MVC
  - ◆ Để truy vấn dữ liệu, trước hết ta cần tạo ra câu truy vấn bằng LINQ
  - ◆ Sau đó là chạy câu truy vấn LINQ. Để chạy câu truy vấn này ta cần chỉ rõ nguồn dữ liệu của câu truy vấn.
  - ◆ Truy vấn LINQ có thể được thực hiện trên các đối tượng mà đối tượng đó thực thi một trong hai giao diện IEnumerable<T> hoặc IQueryable<T>
  - ◆ Cả hai giao diện trên đều cho phép tạo câu truy vấn LINQ để truy vấn dữ liệu từ các nguồn khác nhau.



# Sử dụng truy vấn LINQ

- ◆ Dòng lệnh sau tạo câu truy vấn LINQ để truy vấn dữ liệu từ bảng Customers

```
[IQueryable<Customer> q = from s in db.customers select s;
```

- ◆ Trong đó,
  - ◆ Mệnh đề from chỉ nguồn chứa dữ liệu cần truy vấn
  - ◆ db là một đối tượng thuộc về lớp DbContext đã tạo trước đó
  - ◆ s là biến chạy
  - ◆ Mệnh đề select chỉ ra rằng mỗi phần tử trong tập kết quả sẽ chứa một Customer

# Sử dụng truy vấn LINQ

- ◆ Sau khi tạo truy vấn LINQ, ta cần chạy truy vấn và duyệt qua từng phần tử trong tập kết quả bằng việc sử dụng vòng lặp foreach
- ◆ Đoạn mã sau mô tả việc truy vấn dữ liệu bằng LINQ

```
string names = "";  
IQueryable<Customer> q = from s in db.customers select s;  
foreach (var cust in q)  
{  
    names = names + " " + cust.Name;  
}  
ViewBag.Name = names;
```

- ◆ Trong đó,
  - ◆ Lệnh foreach được sử dụng để duyệt qua các phần tử trong tập kết quả trả về bởi truy vấn LINQ

# Sử dụng truy vấn LINQ

- ◆ LINQ cũng có thể được sử dụng để tạo các câu truy vấn thực hiện các thao tác khác nhau trên dữ liệu như:
  - ◆ Thực hiện phép chiếu: Chỉ rõ tên trường trong mệnh đề select
  - ◆ Lọc dữ liệu: Mệnh đề where với điều kiện lọc
  - ◆ Sắp xếp: Mệnh đề order by chỉ định trường được sắp xếp tăng/giảm
  - ◆ Phân nhóm: Mệnh đề group by chỉ định trường được phân nhóm

# Sử dụng truy vấn LINQ

- ◆ Trong một số trường hợp ta chỉ cần quan tâm tới một vài thông tin nhất định (không phải tất cả) của một đối tượng trong Model. Ví dụ như ta chỉ cần lấy thuộc tính Name của đối tượng Customer
- ◆ Thực hiện phép chiếu trên nguồn dữ liệu bằng cách được mô tả trong ví dụ sau

```
public static void DisplayCustomerNames() {  
    using (EShopDbContext db = new EShopDbContext()) {  
        IQueryable<String> query = from c in db.Customers select c.Name;  
        Console.WriteLine("Customer Names:");  
        foreach (String custName in query){  
            Console.WriteLine(custName);  
        }  
    }  
}
```

# Sử dụng truy vấn LINQ

- ◆ Mệnh đề where trong câu truy vấn LINQ cho phép lọc dữ liệu theo một điều kiện logic
- ◆ Trong đoạn mã sau mệnh đề where được sử dụng để lọc các bản ghi Customers

```
public static void DisplayCustomerByName() {  
    using (EShopDbContext db = new EShopDbContext()) {  
        IQueryable<Customer> query = from c in db.Customers  
                                     where c.Name == "Alex Parker" select c;  
        Console.WriteLine("Customer Information:");  
        foreach (Customer cust in query)  
        {  
            Console.WriteLine("Customer ID: {0}, Name: {1},  
                              Address: {2}", cust.CustomerId, cust.Name, cust.Address);  
        }  
    }  
}
```

# Sử dụng truy vấn LINQ

- ◆ Để sắp xếp dữ liệu trong câu truy vấn LINQ ta dùng mệnh đề orderby
- ◆ Trong đoạn mã sau mệnh đề orderby được sử dụng để sắp xếp

```
IQueryable<Customer> q = from s in db.customers  
                        where s.City == "New Jersey"  
                        orderby s.Name ascending  
                        select s;
```

# Sử dụng truy vấn LINQ

- ◆ Ta cũng có thể phân nhóm dữ liệu trong câu truy vấn LINQ bằng mệnh đề groupby
- ◆ Trong đoạn mã sau mệnh đề group by được sử dụng để phân nhóm dữ liệu theo thuộc tính City

```
var q = from s in db.customers  
        groups by s.City;
```

# Truy vấn LINQ dạng phương thức

- ◆ LINQ cũng có thể được sử dụng để tạo các câu truy vấn dữ liệu theo dạng gọi các phương thức với các tham số truyền vào là các **biểu thức Lambda**
- ◆ Đoạn mã sau mô tả sử dụng LINQ tạo truy vấn dữ liệu như cách gọi một phương thức Select của các đối tượng dữ liệu

```
public static void DisplayPropertiesMethodBasedQuery() {  
    using (EShopDbContext db = new EShopDbContext()) {  
        var cusNames = db.Customers.Select(c => c.Name);  
        Console.WriteLine("Customer Names and Addresses:");  
        foreach (var n in cusNames)  
        {  
            Console.WriteLine("Name: {0}", n);  
        }  
    }  
}
```

**biểu thức Lambda**



## Review: Phương thức nặc danh

- ◆ C# 3.0 giới thiệu phương thức nặc danh (anonymous method) cho phép khởi tạo đối tượng delegate, không cần tách riêng phương thức mà delegate tham chiếu đến như ví dụ sau:

```
// Khai báo delegate
public delegate void SimpleDelegate(int x);
// Khởi tạo delegate sử dụng phương thức nặc danh
SimpleDelegate simpleDelegate = delegate(int n)
{
    Console.WriteLine("Phương thức được gọi bởi delegate");
    Console.WriteLine(n);
};
// Gọi delegate
simpleDelegate(2);
```

# Review: Biểu thức Lambda

- ◆ C# 3.0 đưa ra khái niệm biểu thức Lambda (Lambda expression). Biểu thức Lambda là một hàm nặc danh (anonymous function) chứa biểu thức hay khối lệnh, có thể sử dụng để tạo delegate hay cây biểu thức (expression tree).
- ◆ Biểu thức Lambda sử dụng **toán tử Lambda =>**. Vế trái toán tử Lambda là các tham đối đầu vào, và vế phải là biểu thức hay khối lệnh:

(DanhSáchThamSố) => BiểuThức/ KhốiLệnh

Ví dụ: (x,y) => x + y;

Nếu chỉ có 1 tham số thì có thể bỏ dấu ()

Ví dụ: x => x\*x;

Nếu không có tham số thì cú pháp là:

() => BiểuThức/ KhốiLệnh

# Ví dụ sử dụng biểu thức Lambda

- ◆ Gán biểu thức Lambda gán cho delegate

```
// Khai báo delegate
public delegate int SimpleDelegate(int x);
// Khởi tạo delegate sử dụng biểu thức Lambda
SimpleDelegate simpleDelegate = x => x * x;
// Gọi delegate
int y = simpleDelegate(2);
Console.WriteLine(y);
```

- ◆ Trong trường hợp này biểu thức Lambda  $x \Rightarrow x * x$  chỉ chứa một biểu thức đó là trả về  $x * x$

# Ví dụ sử dụng biểu thức Lambda

- ◆ Biểu thức Lambda chứa khối lệnh có thể gán đến đối tượng delegate:

```
// Khai báo delegate
public delegate void SimpleDelegate(int x, int y);
// Khởi tạo delegate sử dụng biểu thức Lambda
SimpleDelegate simpleDelegate = (x, y) =>
{
    Console.WriteLine(x);
    Console.WriteLine(y);
};
// Gọi delegate
simpleDelegate(2,3);
```

- ◆ Trong trường hợp này biểu thức Lambda chứa một khối lệnh trong cặp dấu { }

# LINQ: Mệnh đề Select

- ◆ Ví dụ sau cho thấy cách sử dụng mệnh đề Select để thực hiện phép chiếu lấy Name và Address để tạo ra lớp các đối tượng nặc danh tương ứng với hai thuộc tính là CustomerName và CustomerAddress.

```
public static void DisplayPropertiesMethodBasedQuery() {  
    using (EShopDbContext db = new EShopDbContext()) {  
        var query = db.Customers.Select(c=> new {CustomerName = c.Name,  
                                                CustomerAddress = c.Address});  
        Console.WriteLine("Customer Names and Addresses:");  
        foreach (var custInfo in query)  
        {  
            Console.WriteLine("Name: {0}, Address: {1}",  
                              custInfo.CustomerName, custInfo.CustomerAddress);  
        }  
    }  
}
```

# Các phương thức LINQ

- ◆ Trong ví dụ trên, mệnh đề Select được sử dụng để thực hiện phép chiếu trên bảng Customer để lấy các thông tin gồm Name và Address
- ◆ Bằng cách tương tự ta cũng có thể thực hiện các thao tác truy vấn khác bằng cách gọi các phương thức khác như Where, GroupBy, Max...

Phương thức	Mô tả	Ví dụ
.Where(e=>điều kiện)	Lọc	Students.Where(s=>s.Marks > 9)
.GroupBy(e=>biểu thức)	Nhóm	Students.GroupBy(s=>s.Clazz)
.OrderBy(e=>biểu thức) .OrderByDescending(e=>biểu thức)	Sắp xếp	Students.OrderBy(s=>s.Name)
.Select(e=>đối tượng)	Chọn	Students.Select(s=>new{s.Name, s.Marks})
.Distinct()	Giữ 1 của các đối tượng giống nhau	Numbers.Distinct()

# LINQ: Mệnh đề Where

- ◆ Trong ví dụ sau, sử dụng phương thức Where để lọc dữ liệu theo Name

```
public static void DisplayPropertiesMethodBasedQuery() {  
    using (EShopDbContext db = new EShopDbContext()) {  
        var query = db.Customers.Where(c => c.Name="John");  
        Console.WriteLine("Customer Names and Addresses:");  
        foreach (var custInfo in query)  
        {  
            Console.WriteLine("Name: {0}, Address: {1}",  
                custInfo.Name, custInfo.Address);  
        }  
    }  
}
```

biểu thức Lambda

# LINQ: Method chaining

- ◆ Trong ví dụ sau, sử dụng mệnh đề Where để lọc dữ liệu theo Name sau đó lại dùng mệnh đề Select để chiếu lấy Name và Address (Chaining Method)

```
public static void DisplayPropertiesMethodBasedQuery() {  
    using (EShopDbContext db = new EShopDbContext()) {  
        var query = db.Customers.Where(c => c.Name="John")  
                                .Select(c => new {CustomerName=c.Name,  
                                                CustomerAddress=c.Address});  
  
        Console.WriteLine("Customer Names and Addresses:");  
        foreach (var custInfo in query)  
        {  
            Console.WriteLine("Name: {0}, Address: {1}",  
                              custInfo.CustomerName, custInfo.CustomerAddress);  
        }  
    }  
}
```



# LING: Mệnh đề OrderBy

- ◆ Trong ví dụ sau, sử dụng mệnh đề OrderByDescending để sắp xếp dữ liệu giảm dần theo Customer.Id

```
public static void DisplayPropertiesMethodBasedQuery() {  
    using (EShopDbContext db = new EShopDbContext()) {  
        var query = db.Customers.OrderByDescending(c => c.Id)  
            .Select(c => new { CustomerName=c.Name,  
                             CustomerAddress=c.Address});  
  
        Console.WriteLine("Customer Names and Addresses:");  
        foreach (var custInfo in query)  
        {  
            Console.WriteLine("Name: {0}, Address: {1}",  
                custInfo.CustomerName, custInfo.CustomerAddress);  
        }  
    }  
}
```

# LINQ: Mệnh đề GroupBy

- ◆ Trong ví dụ sau, sử dụng mệnh đề GroupBy để phân nhóm Product theo CategoryId và đếm số Product trong từng nhóm

```
public static void DisplayPropertiesMethodBasedQuery() {  
    using (EShopDbContext db = new EShopDbContext()) {  
        var groups = db.Products.GroupBy(p => p.CategoryId)  
                                .Select(g => new {CatId = g.Key, Num = g.Count()});  
        Console.WriteLine("Customer Names and Addresses:");  
        foreach (var g in groups)  
        {  
            Console.WriteLine("Category: {0}, NumOfProduct: {1}",  
                              g.CatId, g.Num);  
        }  
    }  
}
```