Forårets valgfag

Sæt kryds i kalenderen – onsdag den 16. november, kl. 12-45-13:15

De udbudte valgfag er:

1. Frameworks (Morten Mathiasen)
2. Mobile Development, Android (Martin Knudsen)
3. Mobile Development, iOS (Kaj Schermer Didriksen)

Efter præsentation får I mulighed for at vælge jer ind på fagene i prioriteret rækkefølge. I skal følge 2 af 3.

Today's Agenda

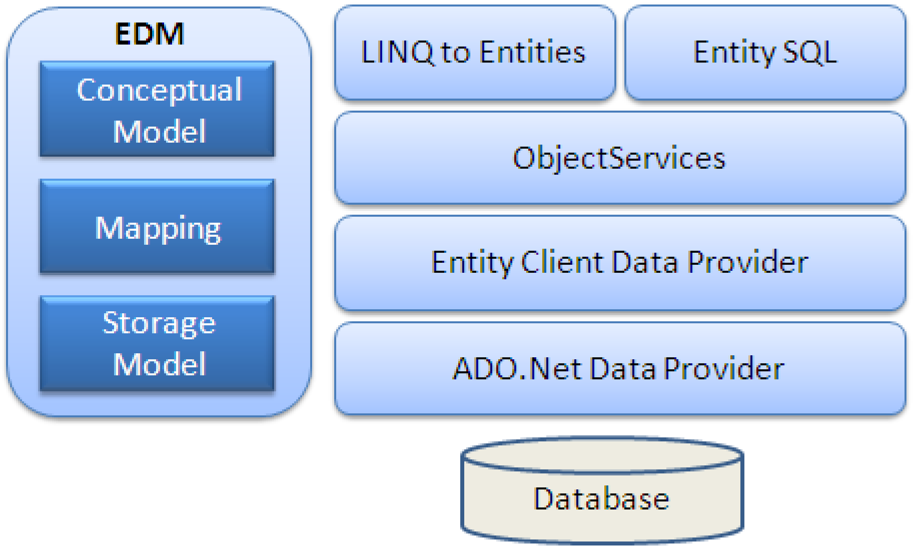
- Introduction to Entity Framework
  - What Is Entity Framework?
  - Entity Framework Workflows
  - Installation of EF
  - Release History
- Answer Questions in Groups
- Exercises
- An introduction to LINQ and Lamda Expressions
- Exercises (continued)

Entity Framework

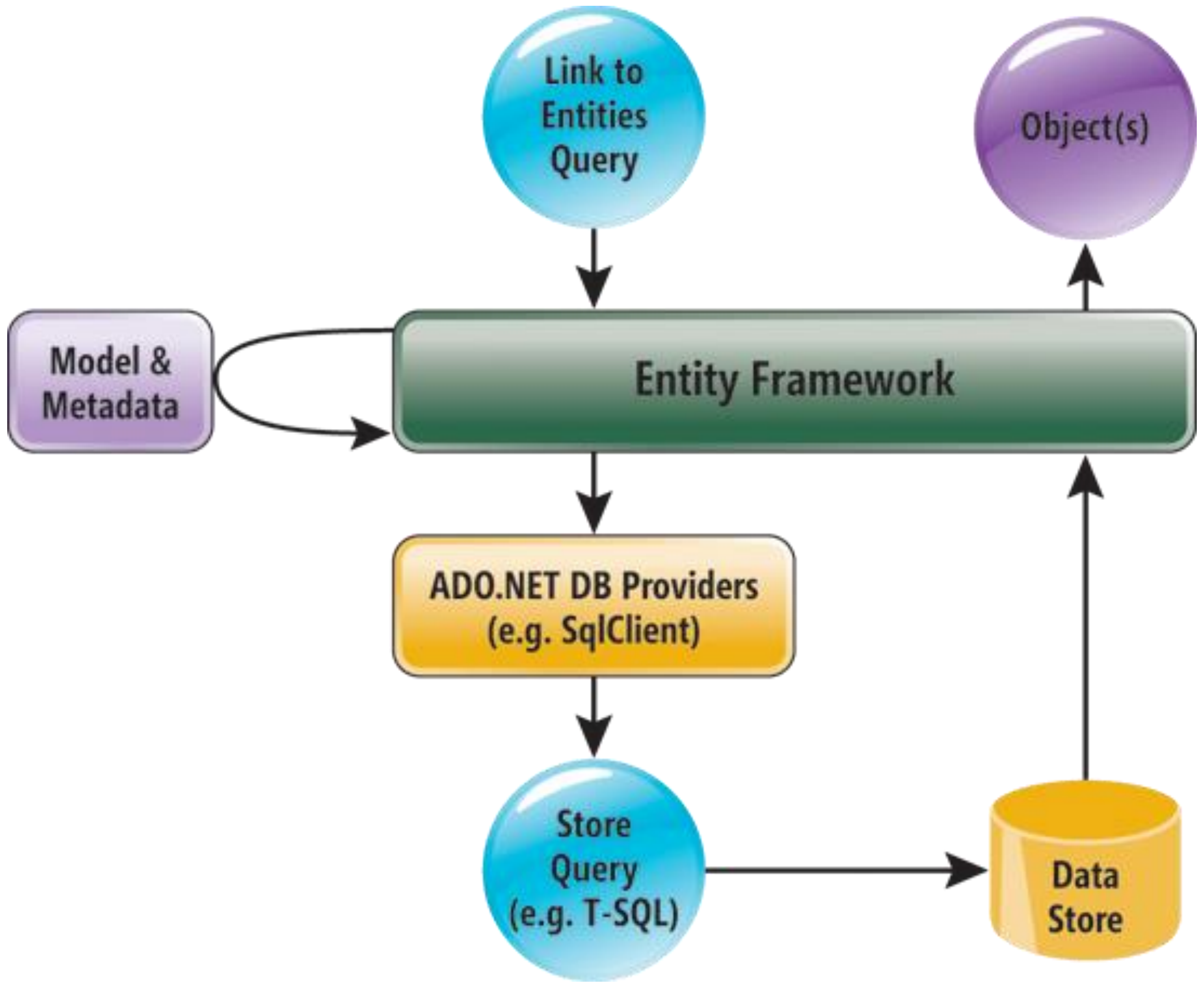
An Introduction

What is Entity Framework?

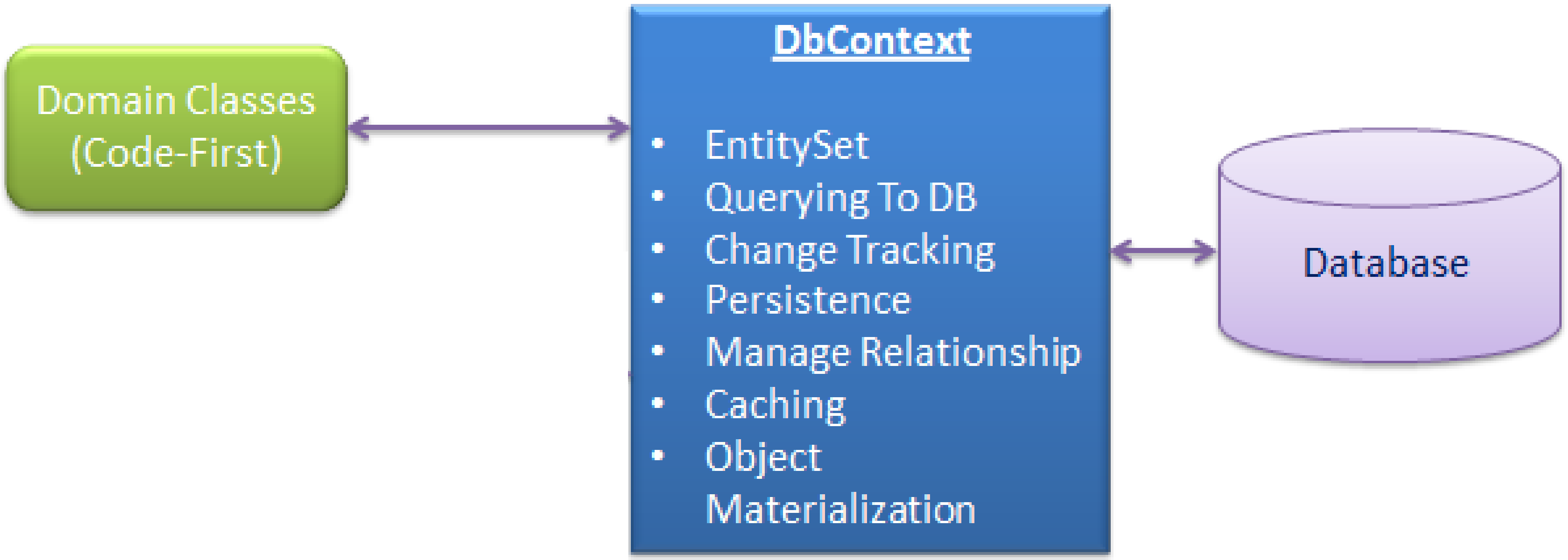
- Entity framework is an ORM (object-relational mapper) that enables us to connect to the DB (SQL server) and map DB to our models and vice versa.
- One of the core capabilities of the Entity Framework is generation of SELECT, UPDATE, INSERT and DELETE commands.
- The focal point of the Entity Framework is the Entity Data Model (EDM), a conceptual model that describes an application's domain objects (which is normally in the **`Models`** folder).



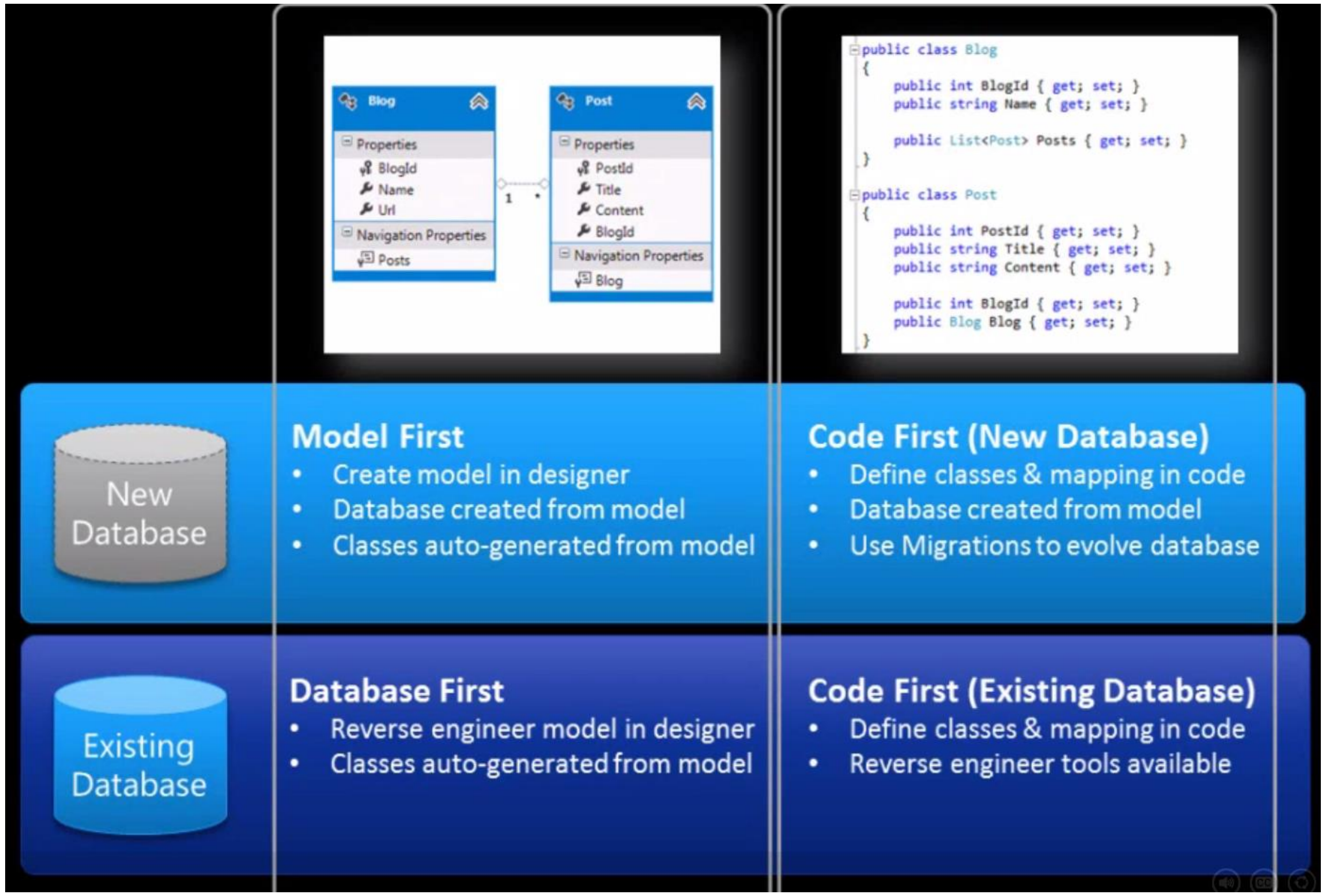
The Entity Framework lets developers express queries against the model rather than concern themselves with details of the database.



The `DbContext` object gives you access to the model



Entity Framework Workflows



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

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

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

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

✗	New Database	**Model First** • Create model in designer • Database created from model • Classes auto-generated from model	**Code First (New Database)** • Define classes & mapping in code • Database created from model • Use Migrations to evolve database	✓
✗	Existing Database	**Database First** • Reverse engineer model in designer • Classes auto-generated from model	**Code First (Existing Database)** • Define classes & mapping in code • Reverse engineer tools available	(✓)

Application Design Strategies

1. Domain Centric Approach (Domain Driven Design)
   - In this approach we design our models in such a way that they can be stored/persisted anywhere -> Entity Framework Code First

2. Data Centric Applications (Data Driven Design)
   - Data Model comes first -> Entity Framework Database First or Code First with an Existing Database (since EF 6.1)

Essential EF Classes

- **`System.Data.Entity.DbContext`**
  - A class that represents a session with the database, allowing you to query and save data. A context derives from the **`DbContext`** or **`ObjectContext`** class
  - The primary class that is responsible for interacting with data as objects
- **Fluent API**
  - An API that can be used to configure a Code First model (e.g. relationships)

POCO data classes

- Custom data classes that only contain properties (no attributes or methods) that maps to entities and relationships in EF
- They are Plain Old CLR (Common Language Runtime) Objects (POCO)
- POCO are classes that remain free from a backing infrastructure, such as Entity Framework
- A POCO class is known as persistence-ignorant object

Model class `Customer` as an example of a POCO class

```
public class Customer {
    public int ID { get;  set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public string Address { get; set; }
    public string City { get; set; }
    public string Zip { get; set; }
    public string Phone { get; set; }
    public string Email { get; set; }
}
```

DBContext Class Example

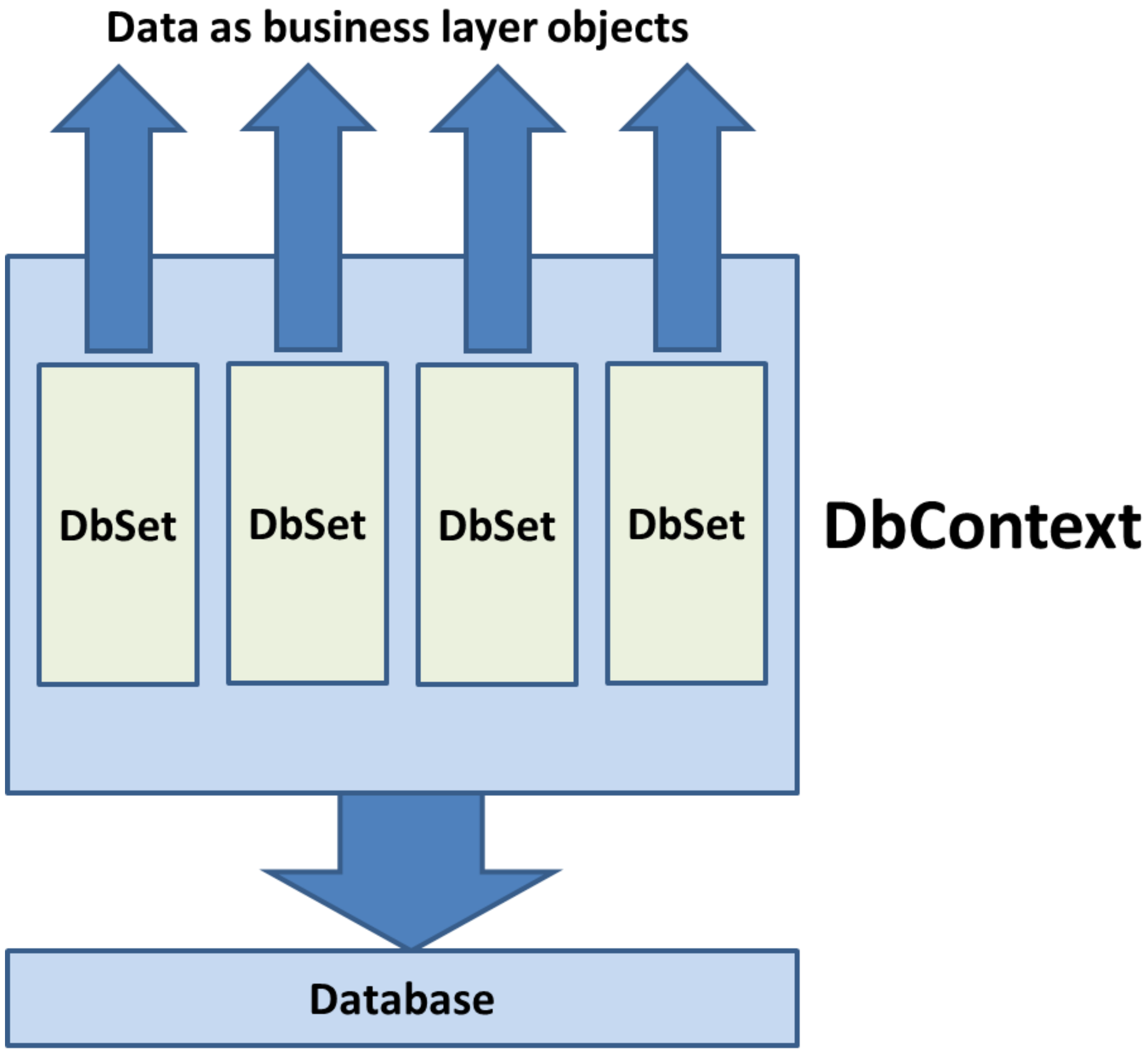
```
namespace Lesson08.DAL {
    public class MbmStoreContext : DbContext {
        public MbmStoreContext() : base("PetHotelContext") { }
        public DbSet<Customer> Customers { get; set; }
        public DbSet<OrderItem> OrderItems { get; set; }
        public DbSet<Invoice> Invoices { get; set; }

        protected override void OnModelCreating(DbModelBuilder modelBuilder) {
            modelBuilder.Conventions.Remove<PluralizingTableNameConvention>();
        }
}
```

DBContext

Entity

A class or object that represents application data such as Customers, Products, and Orders: **`DbSet<TEntity>`**



web.config

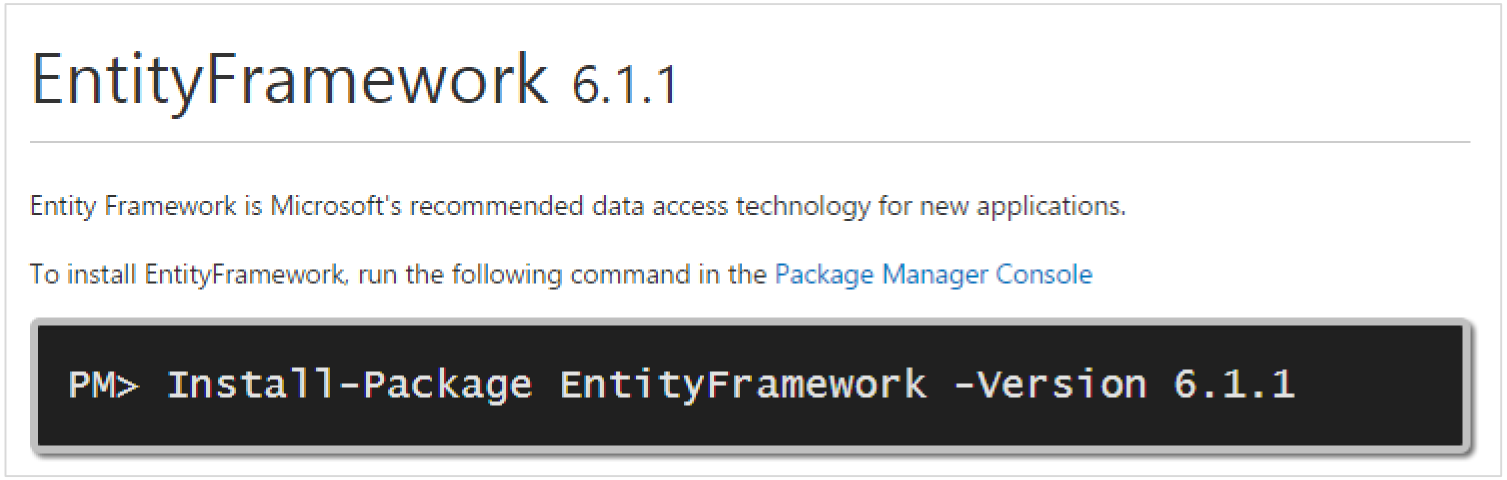
```
<configuration>
   <connectionStrings>
      <add name="PetHotelContext"
         connectionString="Data Source=(localdb)\v11.0;
         Integrated Security=True;
         AttachDbFilename=|DataDirectory|MbmStoreContext.mdf"
         providerName="System.Data.SqlClient" />
  </connectionStrings>
</configuration>
```

Using EF from the Controller

```
public class CustomerController : Controller {
   private MbmStoreContext db = new MbmStoreContext();

   // GET: Customers from DB
   public ActionResult Index() {
      return View(db.Customers.ToList());
   }
}
```

Installation



EntityFramework 6.1.1

Entity Framework is Microsoft's recommended data access technology for new applications.

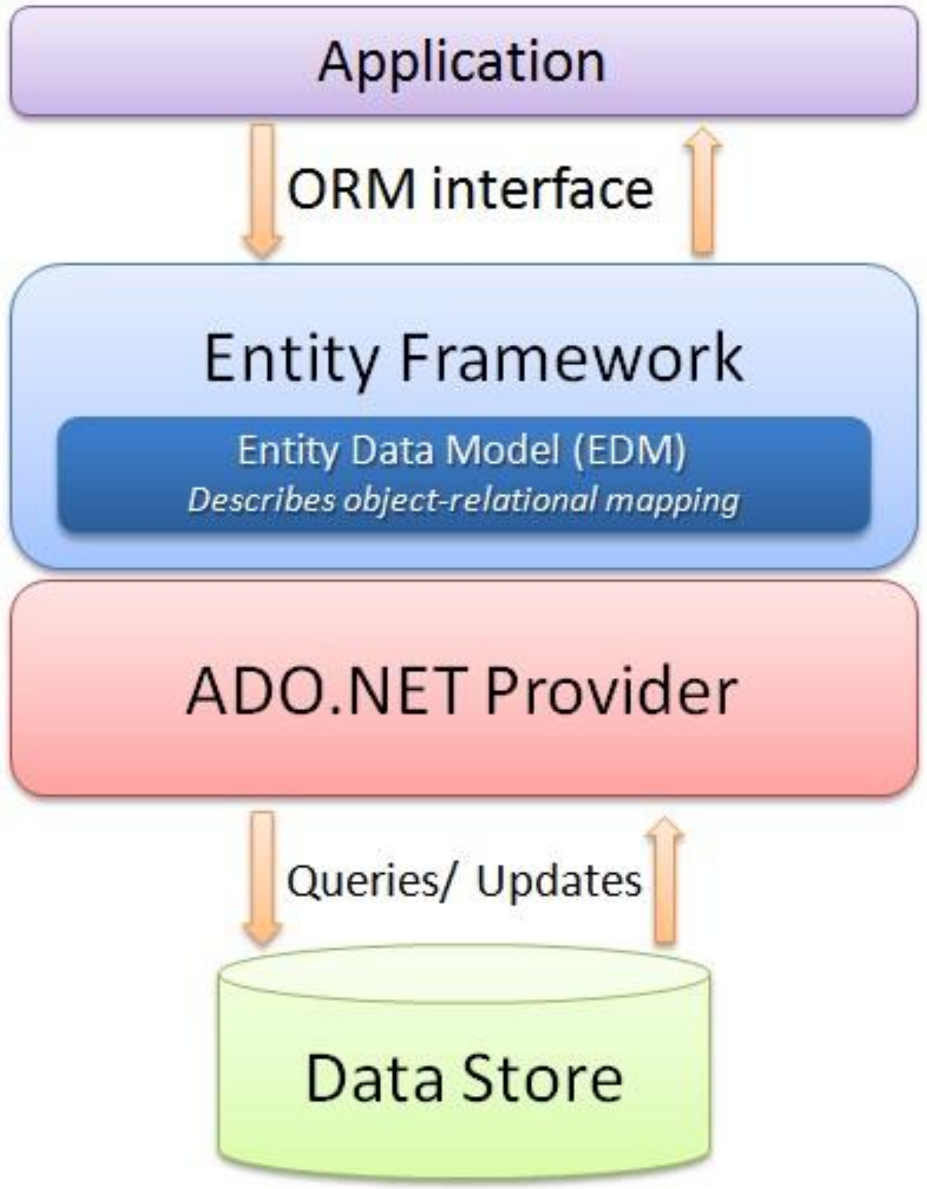
To install EntityFramework, run the following command in the Package Manager Console

```
PM> Install-Package EntityFramework -Version 6.1.1
```

Entity Framework Release History

- EF v1 released on 11 August **2008**. This version was widely criticized
- EF v4 (next version) released as part of .NET 4.0 on 12 April 2010 and addressed many of the criticisms made of version 1
- EF 4.1 released on April 12, 2011, with Code First support
- 4.3.1 released on February 29, 2012. There were a few updates, like support for migration
- EF 5.0 released on August 11, 2012 and is targeted at .NET framework 4.5. Without any runtime advantages over version 4
- EF 6.0 released on October 17, 2013 and is now an open source project. This version has a number of improvements for code-first support
- EF 6.1. released on March 17, 2014. Code First reverse engineering from an existing database.
- EF 6.1.2 released on June 20, 2014. Mainly bug fixes.
- **EF 6.1.3** released on March 10, 2015. Bug fixes. https://github.com/aspnet/EntityFramework6
- EF Core 1.0 released on June 27, 2016. https://github.com/aspnet/EntityFramework

Entity Framework in a nutshell



Other ORM (Object/Relational Mapper) Tools

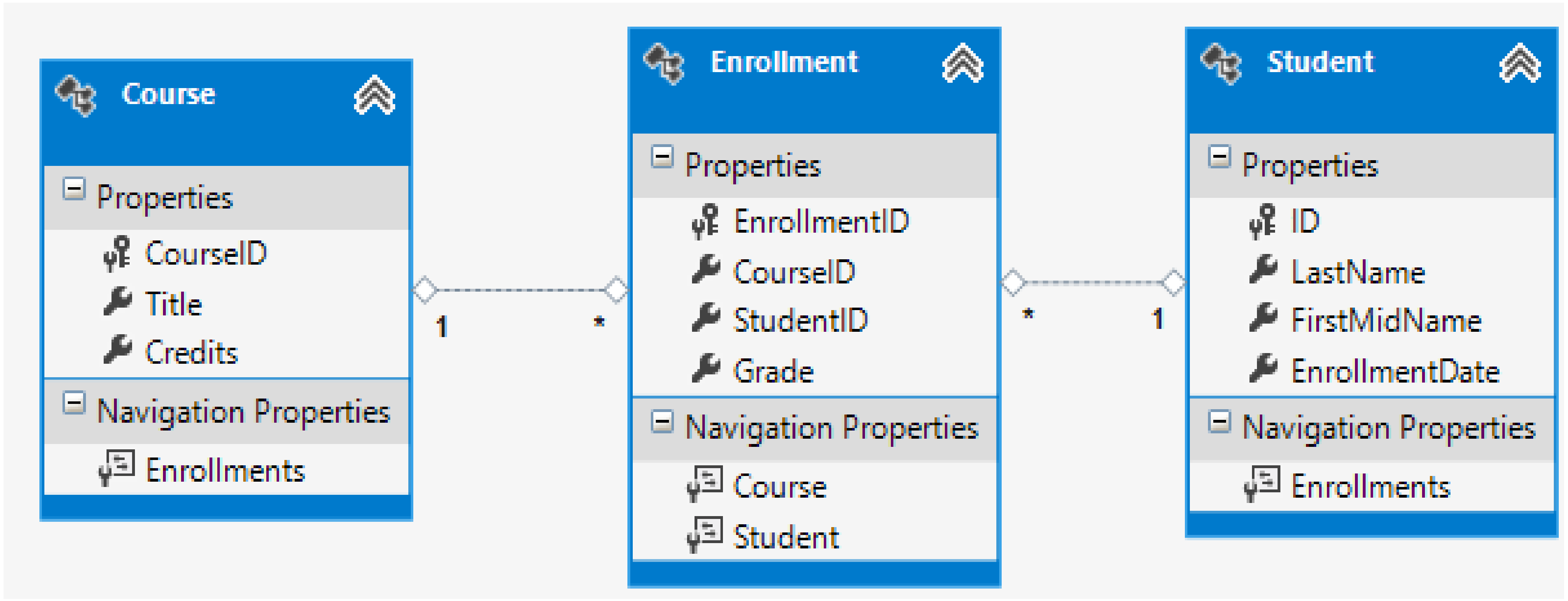
- Dapper.NET ORM (StackExchange/dapper-dot-net)
- NHibernate (sourceforge.net/projects/nhibernate)
- NET ORM tool (www.telerik.com/data-access)

Questions

Q1: Naming conventions

- Table names are *pluralized*
  E.g. Entity class name `Student` -> table name `Students`
- Entity property names are used for column names
  E.g. Entity `Student` class property `FirstName` -> table column `FirstName`
- Entity properties that are named ID or *classname*ID are recognized as primary key properties
- A property is interpreted as a foreign key property if it's named *<navigation property name><primary key property name>* (for example, `StudentID` for the Student navigation property since the Student entity's primary key is **`ID`**).
  Foreign key properties can also be named the same simply <primary key property name> (for example, **`EnrollmentID`** since the Enrollment entity's primary key is `EnrollmentID`).

Q2: What is a navigation property?



Q2: Navigational properties

- Navigation properties are typically defined as virtual. Please explain why?
  - To enable lazy loading (loads only object when needed)
- A navigation property that holds multiple entities must be a list. Which type must that list be??

```
public class Student {
    public int ID { get; set; }
    public string LastName { get; set; }
    public string FirstMidName { get; set; }
    public DateTime EnrollmentDate { get; set; }
    public string Secret { get; set; }
    public virtual ICollection<Enrollment> Enrollments { get; set; }
}
```

Q3: Override conventions

- You can use:
  - Data Annotations
  - The Fluent API

- Primary key?

```
[Key]
public string CourseCode {get; set}
```

- Avoid DB automatic generation of primary key values

```
[DatabaseGenerated(DatabaseGeneratedOption.None)]
public string CourseID {get; set}
```

Q3: Foreign key?

```
public class Course {
        public int CourseID { get; set; }
        public string CourseName { get; set; }

        //Foreign key for Department
        public int DepartmentRefId { get; set; }

        [ForeignKey("DepartmentRefId")]
        public Departement Department { get; set; }
}
```

```
public class Department {
        public int DepartmentId { get; set; }
        public string DepartmentName { get; set; }
        public ICollection<Course> Courses { get; set; }
}
```

Q4: Explain this code

```
using ContosoUniversity.Models;
using System.Data.Entity;
using System.Data.Entity.ModelConfiguration.Conventions;

namespace ContosoUniversity.DAL {

    public class SchoolContext : DbContext {
        public SchoolContext() : base("SchoolContext") {}

        public DbSet<Student> Students { get; set; }
        public DbSet<Enrollment> Enrollments { get; set; }
        public DbSet<Course> Courses { get; set; }

        protected override void OnModelCreating(DbModelBuilder modelBuilder) {
            modelBuilder.Conventions.Remove<PluralizingTableNameConvention>();
        }
    }
}
```

Q5: The Seed method
- in the **`SchoolInitializer.cs`** class

- The purpose of the seed method?
    - Insert example data
- How do you tell the EF to use the initializer class?
    - `web.config` file (or `global.asax` in `Application_Start`)
- When is the Seed method inside the initializer class called?
    - Whenever the model is changed
- What happens when the Seed method is called?
    - The is database dropped and recreated (drop/recreate) (default behavior – can be changed)
- Where should the seed method be located
    - Inside the DAL folder (maybe)

Q6: LocalDB

- What is LocalDB?
  - Lightweight SQLExpress instance. LocalDB is created specifically for developers.
  - It is very easy to install - developers no longer have to install and manage a full instance of SQL Server Express on their laptops
  - Offers the same T-SQL language, programming surface and client-side providers as the regular SQL Server Express.
- How do you specify the name and location of you LocalDB database file?
  - Inside the `web.config` file (inside the `connectionStrings` node)
- What is the preferred location for LocalDB database files for ASP.NET MVC web development?
  - Inside the `/App_Data` folder

Q6: Connection String, please explain

```
<connectionStrings>
    <add name="SchoolContext"

      connectionString="Data Source=(localdb)\MSSQLLocalDB;
      Integrated Security=True;
      MultipleActiveResultSets=True;
      AttachDbFilename=|DataDirectory|ContosoUniversity.mdf“

      providerName="System.Data.SqlClient" />
</connectionStrings>
```

Q8: Which files and methods does the scaffolder *MVC 5 Controller with Views, using Entity Framework* create?

Action Method	View
`Index()`	`Index.cshtml` – list of students
`Details(int? id)`	`Details.cshtml` – detailed student view
`Create()`	`Create.cshtml` – empty student form
`[HttpPost]` `Create(Student student)`	`Create.cshtml` – form if error, else student list
`Edit(int? id)`	`Edit.cshtml` – populated student form
`[HttpPost]` `Edit(Student student)`	`Edit.cshtml` – form if error, else student list
`Delete(int? id)`	`Delete.cshtml` – detailed student view
`[HttpPost]` `Delete(int id)`	`Index.cshtml` – list of students

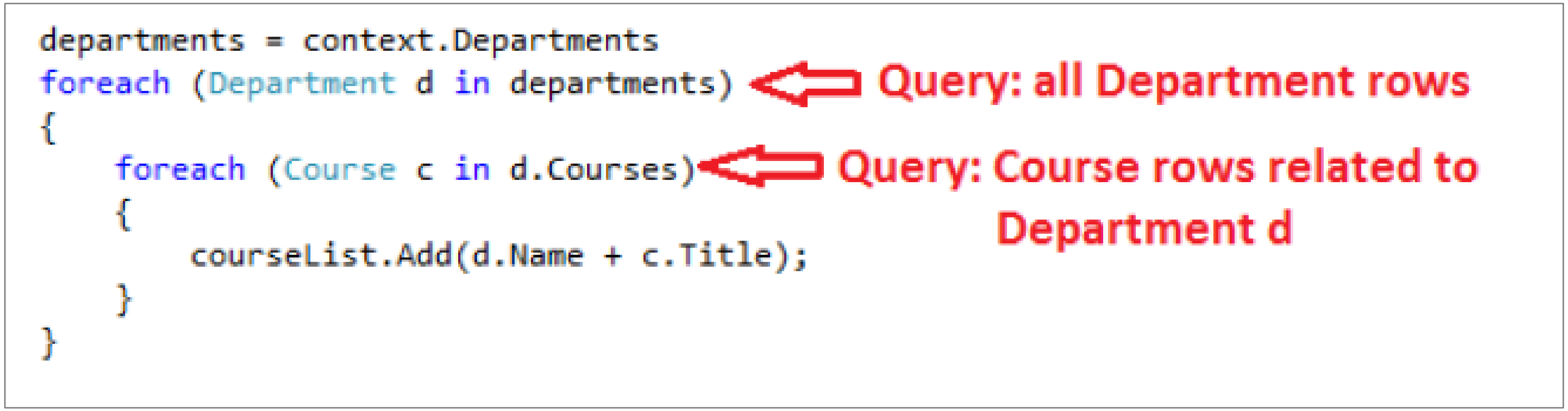
Q8: MVC 5 Controller with views, using Entity Framework

The first time you run the scaffolder **MVC 5 Controller with views, using Entity Framework** in a project it also:

1. Installs Entity Framwork
2. Creates a DbContext class in the Models folder
3. Adds a connection string to the `Web.config` file
4. Adds a reference to the `DbContext` class in `Web.config` file

Q9: What is Lazy Loading?

- When the entity is first read, **related data isn't retrieved**. However, the first time you attempt to access a navigation property, the data required for that navigation property is automatically retrieved. The `DbContext` class enables lazy loading by default

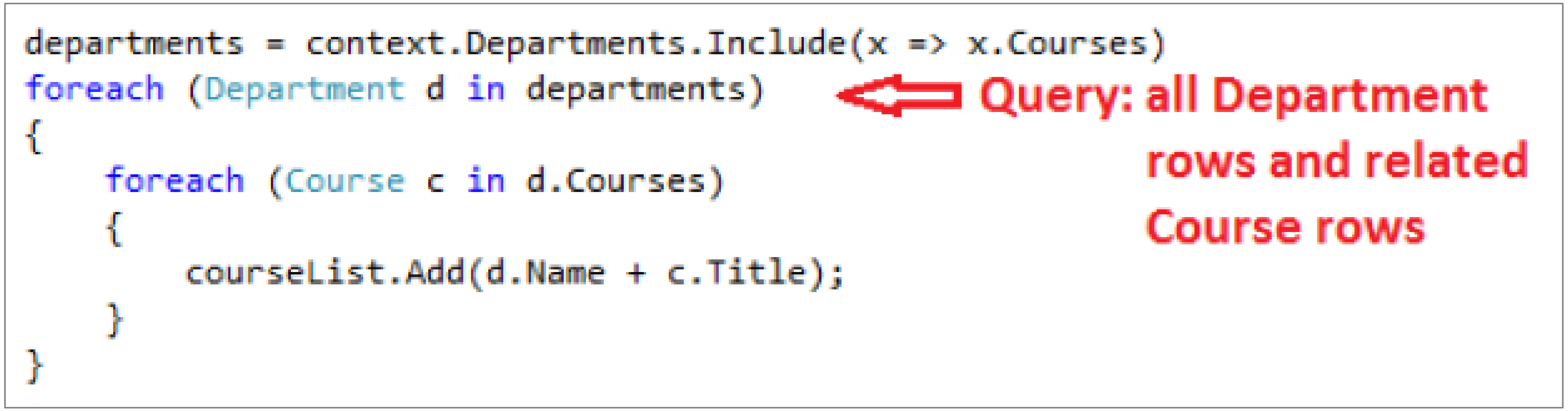


```
departments = context.Departments
foreach (Department d in departments)
{
    foreach (Course c in d.Courses)
    {
        courseList.Add(d.Name + c.Title);
    }
}
```

Query: all Department rows
Query: Course rows related to Department d

Q9: What is Eager Loading?

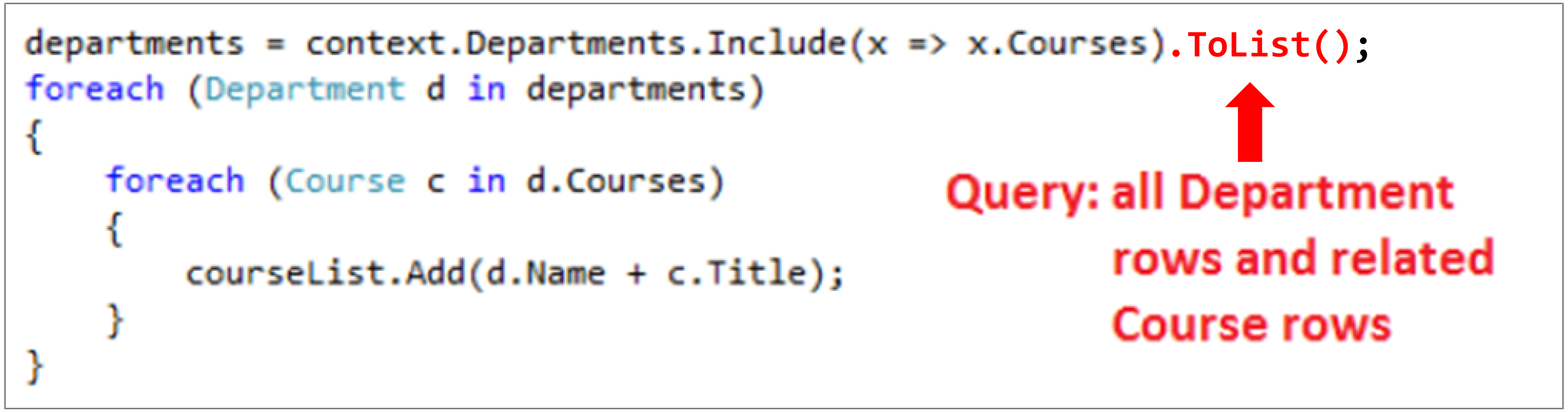
- When the entity is read, related data is retrieved along with it. This typically results in a single join query that retrieves all of the data that's needed. You specify eager loading by using the **`Include`** method.



```
departments = context.Departments.Include(x => x.Courses)
foreach (Department d in departments)
{
    foreach (Course c in d.Courses)
    {
        courseList.Add(d.Name + c.Title);
    }
}
```

Query: all Department rows and related Course rows

Q9: Force Immediate Query Execution



```
departments = context.Departments.Include(x => x.Courses).ToList();
foreach (Department d in departments)
{
    foreach (Course c in d.Courses)
    {
        courseList.Add(d.Name + c.Title);
    }
}
```

Query: all Department rows and related Course rows

LINQ Methods that Force Entity Framework to return the result from the DB

Enumerables

- ToList
- ToArray
- ToDictonary

Singleton

- Average
- Count
- First
- FirstOrDefault
- Max
- Single
- SingleOrDefault

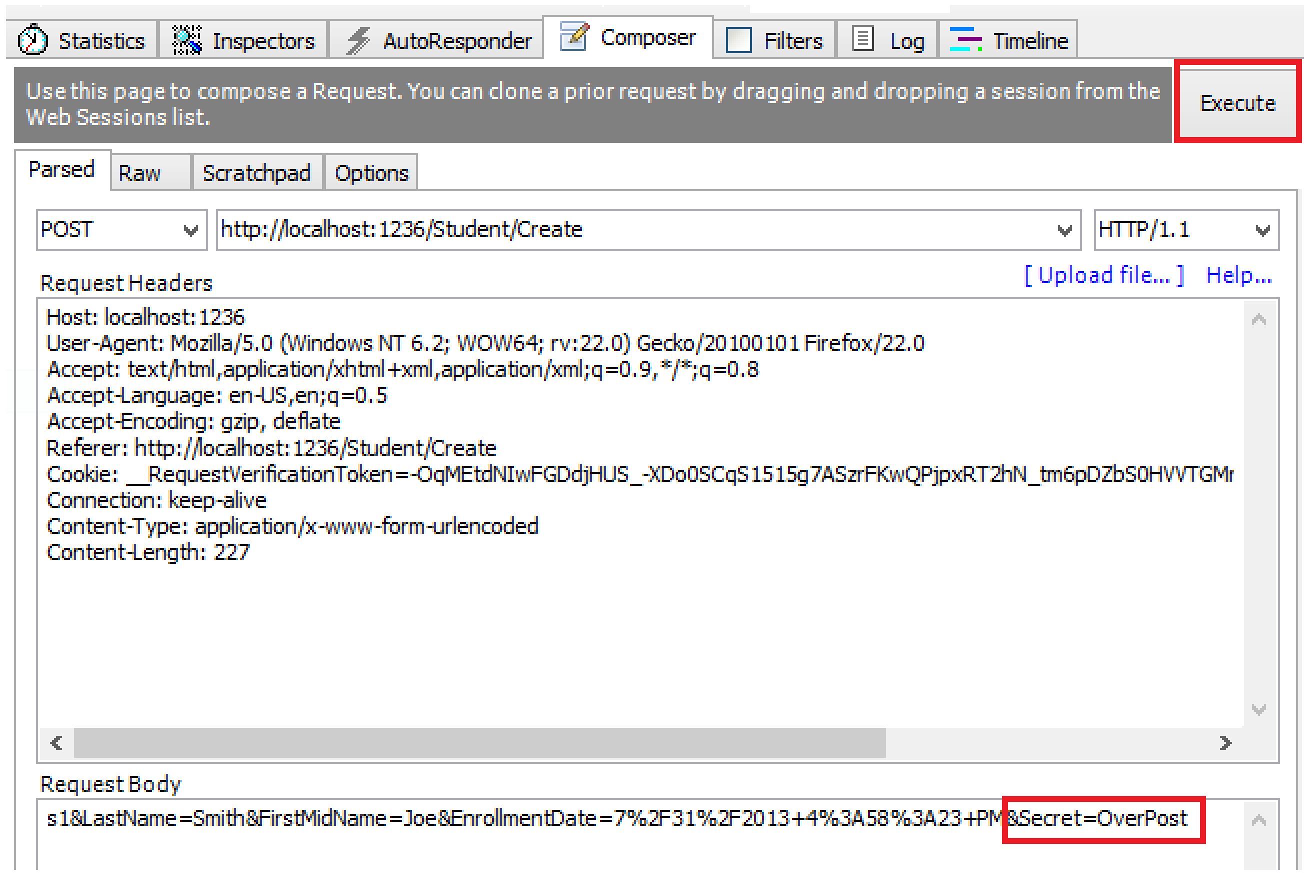
Q10: Securing the code

In the view

```
@Html.AntiForgeryToken()
```

```
[ValidateAntiForgeryToken]
public ActionResult Create([Bind(Include = "LastName,
FirstMidName, EnrollmentDate")]Student student)   {
    try {
        if (ModelState.IsValid) {
            db.Students.Add(student);
            db.SaveChanges();
            return RedirectToAction("Index");
        }
    }
    catch (DataException /* dex */) {
        //Log the error
    }
    return View(student);
}
```

With a tool like Fiddler (or JavaScript) a hacker can POST extra form values



Use this page to compose a Request. You can clone a prior request by dragging and dropping a session from the Web Sessions list.

POST http://localhost:1236/Student/Create HTTP/1.1

Request Headers

```
Host: localhost:1236
User-Agent: Mozilla/5.0 (Windows NT 6.2; WOW64; rv:22.0) Gecko/20100101 Firefox/22.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://localhost:1236/Student/Create
Cookie: __RequestVerificationToken=-OqMEtdNIwFGDdjHUS_-XDo0SCqS1515g7ASzrFKwQPjpxRT2hN_tm6pDZbS0HVVTGMr
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 227
```

Request Body

```
s1&LastName=Smith&FirstMidName=Joe&EnrollmentDate=7%2F31%2F2013+4%3A58%3A23+PM&Secret=OverPost
```

Q10: How does the Edit method differ from **Create** method

```
[ValidateAntiForgeryToken]
public ActionResult Edit([Bind(Include = "LastName,
FirstMidName, EnrollmentDate")]Student student)  {
    try {
        if (ModelState.IsValid) {
            db.Books.Add(book);
            db.SaveChanges();
            return RedirectToAction("Index");
        }
    }
    catch (DataException /* dex */) {
        //Log the error
    }
    return View(student);
}
```

Q10: How does the **`Edit`** method differ from **`Create`** method

```
[ValidateAntiForgeryToken]
public ActionResult Edit([Bind(Include = "ID, LastName,
FirstMidName, EnrollmentDate")]Student student)  {
    try {
        if (ModelState.IsValid) {
            db.Entry(student).State = EntityState.Modified;
            db.SaveChanges();
            return RedirectToAction("Index");
        }
    }
    catch (DataException /* dex */) {
        //Log the error
    }
    return View(student);
}
```

Exercises 1-10

An adaption of Tom Dykstra and Rick Anderson: *Getting Started with Entity Framework 6 Code First using MVC 5 for the* MbmStore project

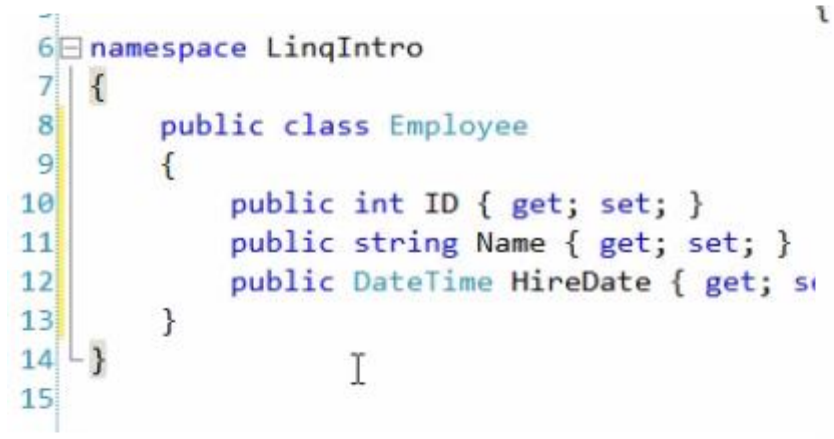
LINQ & Lamda Expressions

The purpose of LINQ

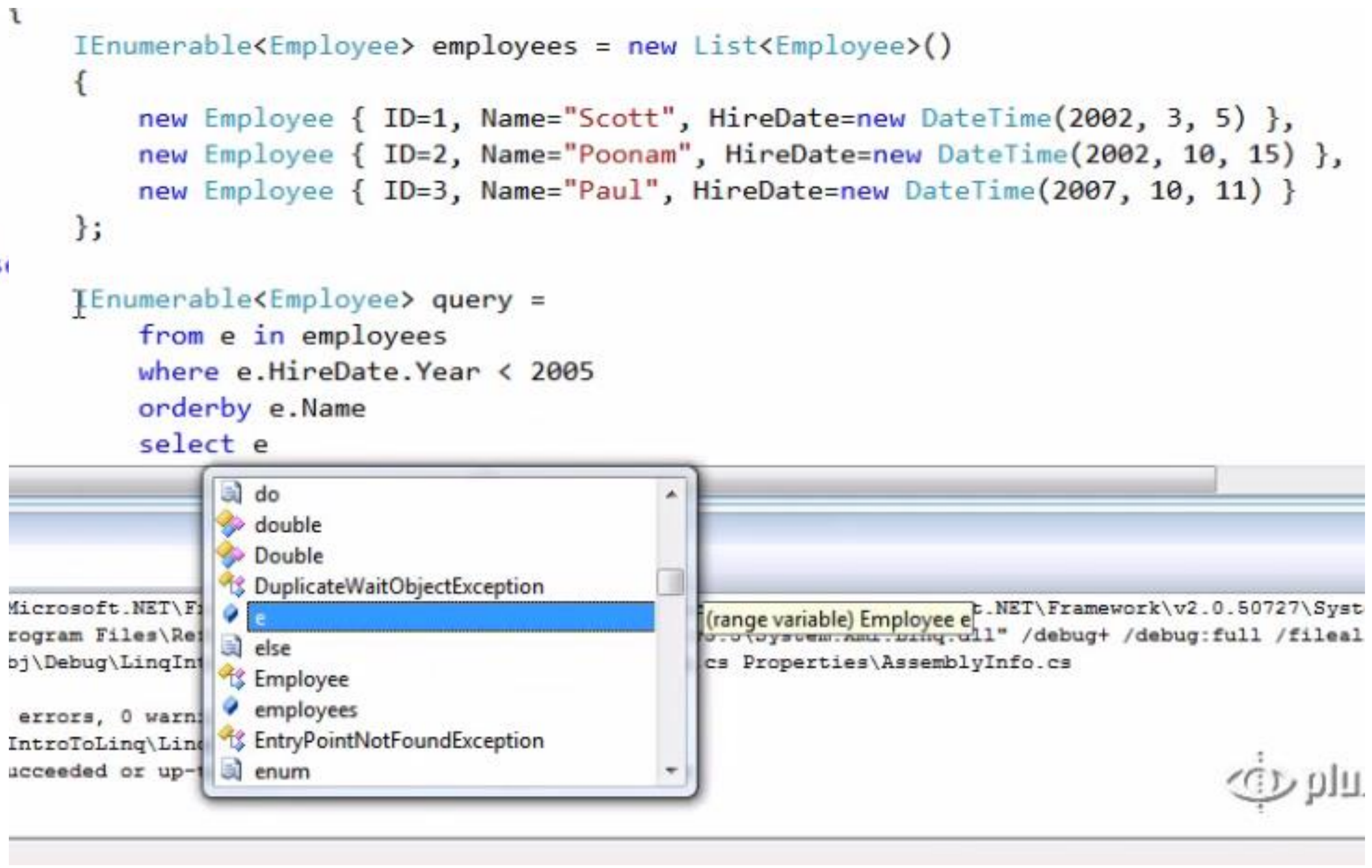
- Making data access easier by integrating querying capabilities into a programming language like C#

- Linq to objects (from collections)
- Linq to XML
- Linq to SQL
- Linq to Entity

LINQ expressons are supported by intellisense



```
namespace LinqIntro
{
    public class Employee
    {
        public int ID { get; set; }
        public string Name { get; set; }
        public DateTime HireDate { get; s
    }
}
```



```
IEnumerable<Employee> employees = new List<Employee>()
{
    new Employee { ID=1, Name="Scott", HireDate=new DateTime(2002, 3, 5) },
    new Employee { ID=2, Name="Poonam", HireDate=new DateTime(2002, 10, 15) },
    new Employee { ID=3, Name="Paul", HireDate=new DateTime(2007, 10, 11) }
};

IEnumerable<Employee> query =
    from e in employees
    where e.HireDate.Year < 2005
    orderby e.Name
    select e
```

```
Microsoft.NET\F                                         t.NET\Framework\v2.0.50727\Syst
rogram Files\Re                                                 ll" /debug+ /debug:full /fileal
bj\Debug\LinqIn                                          cs Properties\AssemblyInfo.cs

 errors, 0 warn
IntroToLinq\Lin
ucceeded or up-
```

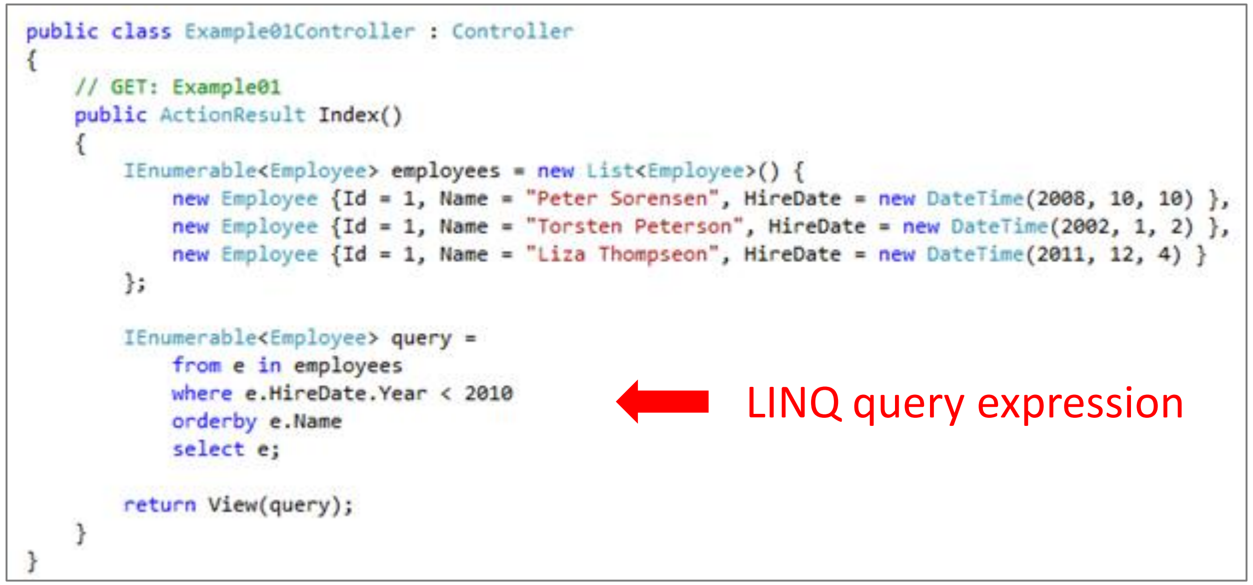
do
double
Double
DuplicateWaitObjectException
e
else
Employee
employees
EntryPointNotFoundException
enum
(range variable) Employee e
plu

LINQ Query Expressens

- The same standard query operators work everywhere
    - Objects
    - Relational data
    - XML data
- More than 50 operators defined
    - Projection
    - Filtering
    - Joining
    - Ordering
    - Agregating
- Similar to
    - Select, Where, From, Join, OrderBy, GroupBy

Defered execution:
Queries are not executed until we access the result

The Controller



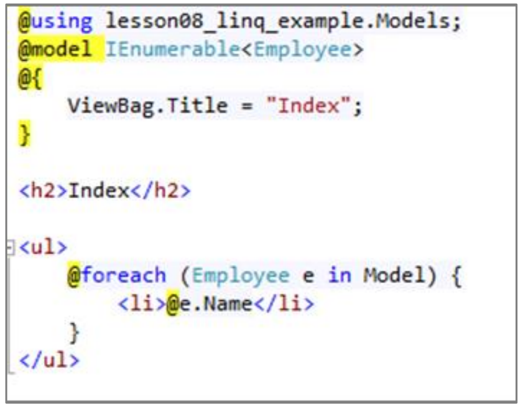
```
public class Example01Controller : Controller
{
    // GET: Example01
    public ActionResult Index()
    {
        IEnumerable<Employee> employees = new List<Employee>() {
            new Employee {Id = 1, Name = "Peter Sorensen", HireDate = new DateTime(2008, 10, 10) },
            new Employee {Id = 1, Name = "Torsten Peterson", HireDate = new DateTime(2002, 1, 2) },
            new Employee {Id = 1, Name = "Liza Thompseon", HireDate = new DateTime(2011, 12, 4) }
        };

        IEnumerable<Employee> query =
            from e in employees
            where e.HireDate.Year < 2010
            orderby e.Name
            select e;

        return View(query);
    }
}
```

LINQ query expression

The View



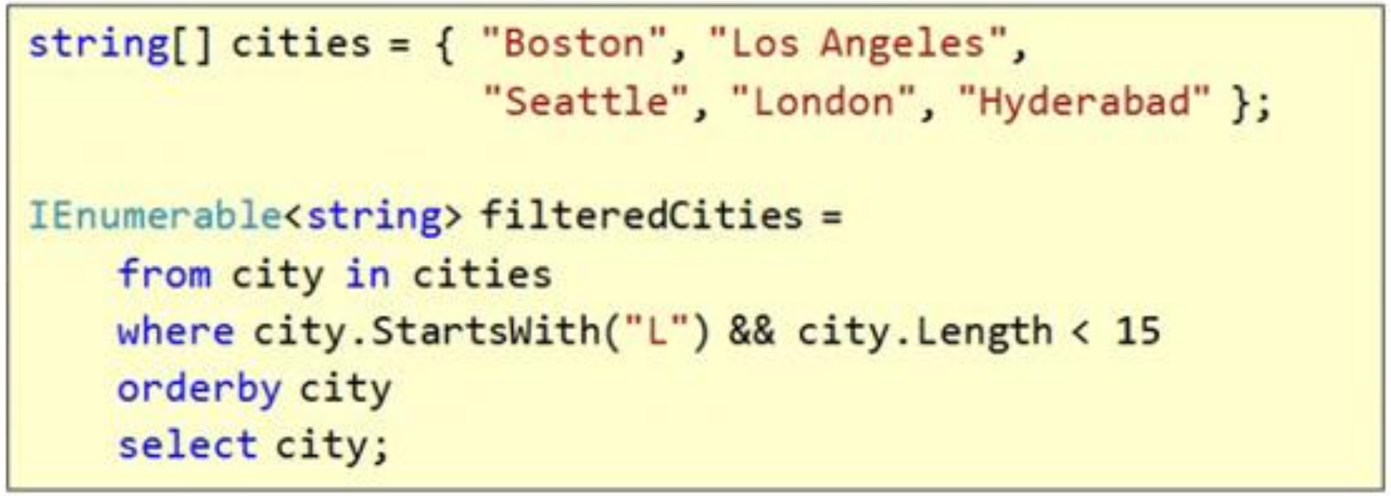
```
@using lesson08_linq_example.Models;
@model IEnumerable<Employee>
@{
    ViewBag.Title = "Index";
}

<h2>Index</h2>

<ul>
    @foreach (Employee e in Model) {
        <li>@e.Name</li>
    }
</ul>
```

Query sent

Comprehensive query syntax in LINQ



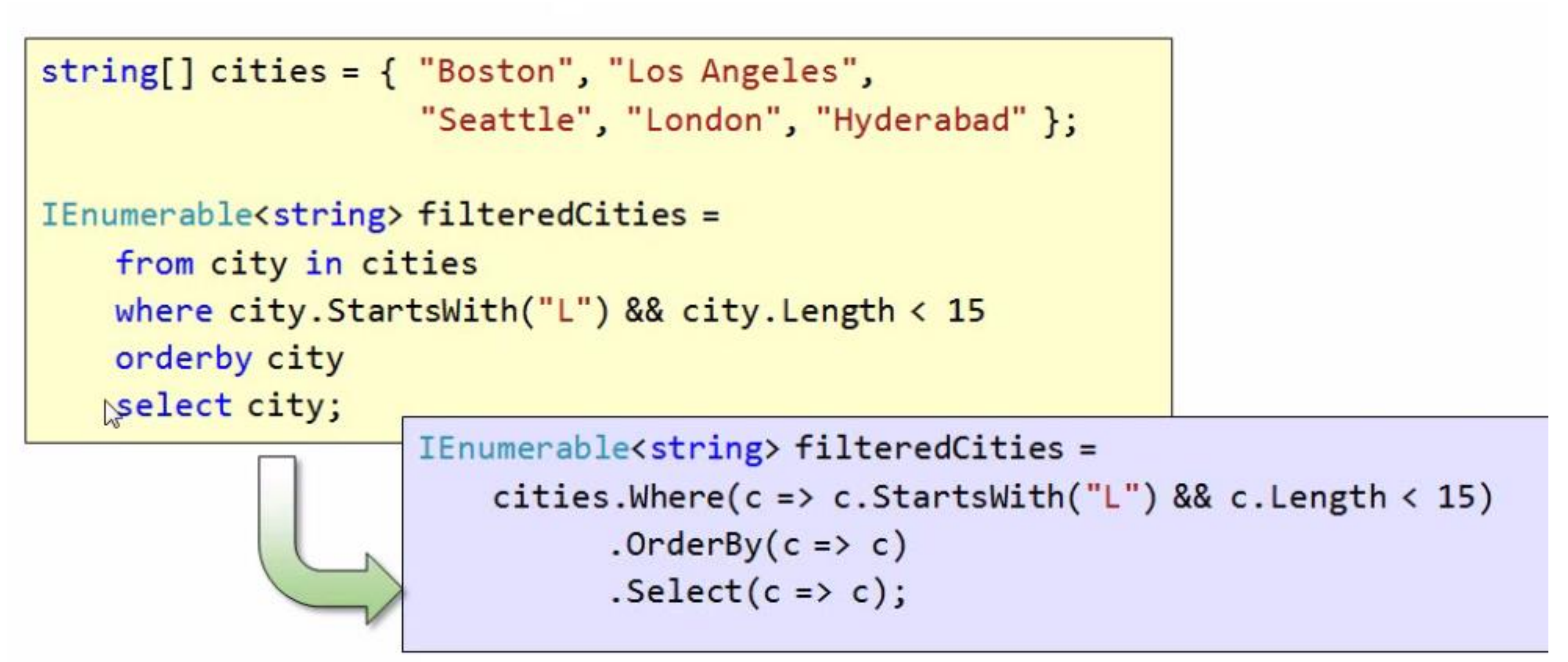
```
string[] cities = { "Boston", "Los Angeles",
                    "Seattle", "London", "Hyderabad" };

IEnumerable<string> filteredCities =
    from city in cities
    where city.StartsWith("L") && city.Length < 15
    orderby city
    select city;
```

- Begings with a *from* clause, ends with a *select* or *group*
- Looks like a SQL query
  - *from* logically comes first (also helps Intellisense)

Sweet and Sugary Syntax

- Compiler transfers query expressions into a series of LINQ method calls with lamda expressions

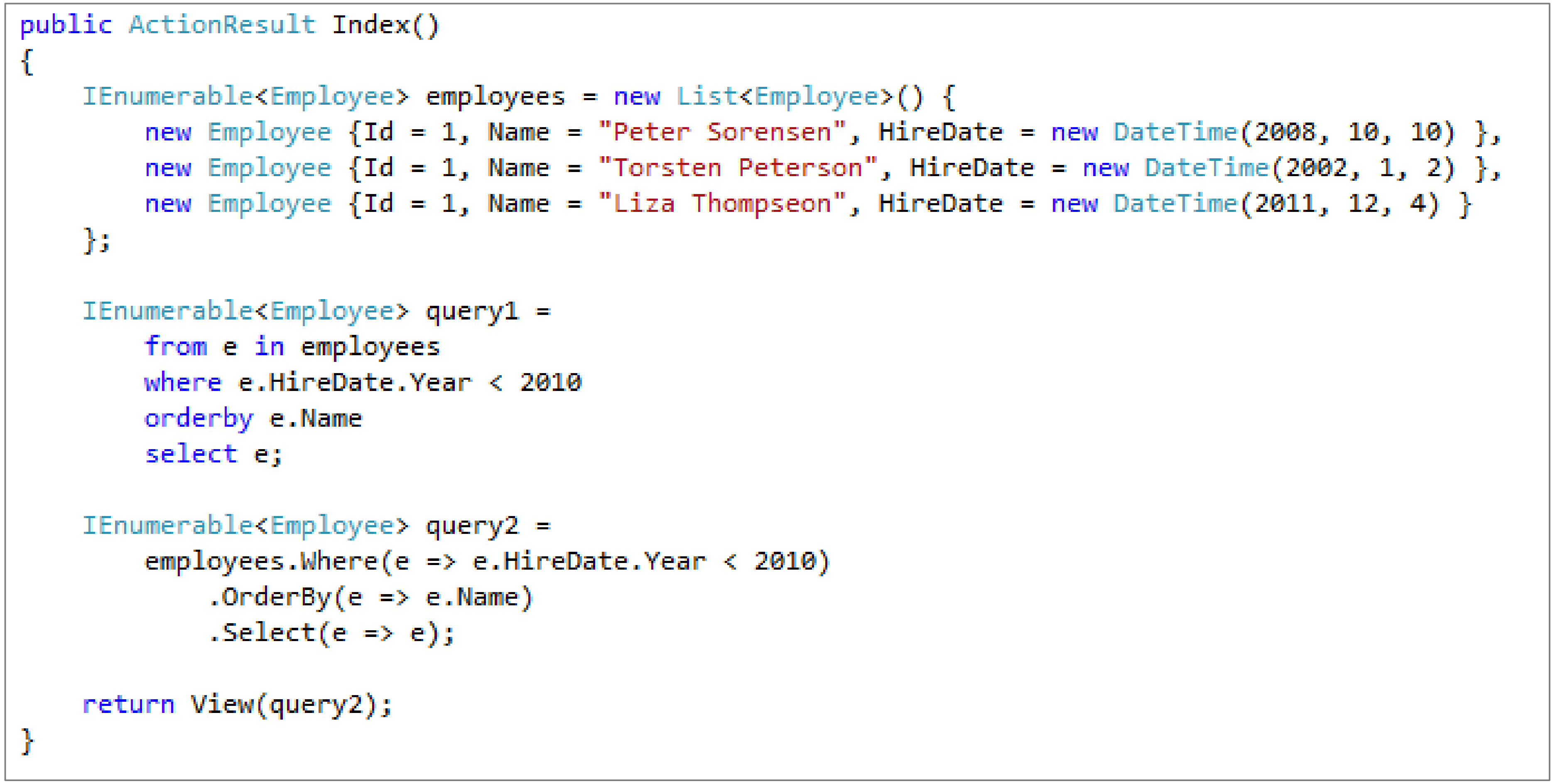


```
string[] cities = { "Boston", "Los Angeles",
                    "Seattle", "London", "Hyderabad" };

IEnumerable<string> filteredCities =
    from city in cities
    where city.StartsWith("L") && city.Length < 15
    orderby city
    select city;
```

```
IEnumerable<string> filteredCities =
    cities.Where(c => c.StartsWith("L") && c.Length < 15)
          .OrderBy(c => c)
          .Select(c => c);
```

Live example



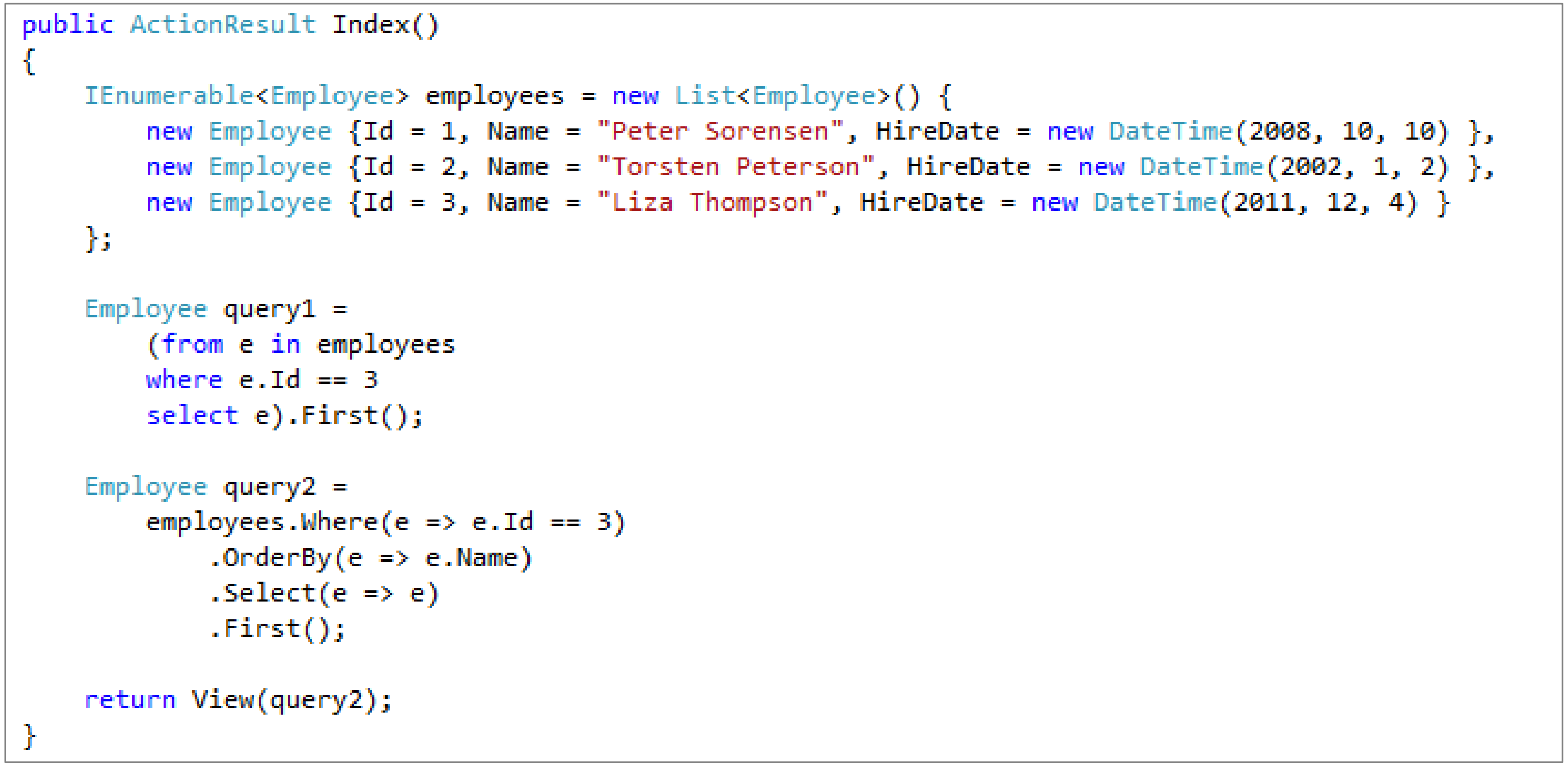
```
public ActionResult Index()
{
    IEnumerable<Employee> employees = new List<Employee>() {
        new Employee {Id = 1, Name = "Peter Sorensen", HireDate = new DateTime(2008, 10, 10) },
        new Employee {Id = 1, Name = "Torsten Peterson", HireDate = new DateTime(2002, 1, 2) },
        new Employee {Id = 1, Name = "Liza Thompseon", HireDate = new DateTime(2011, 12, 4) }
    };

    IEnumerable<Employee> query1 =
        from e in employees
        where e.HireDate.Year < 2010
        orderby e.Name
        select e;

    IEnumerable<Employee> query2 =
        employees.Where(e => e.HireDate.Year < 2010)
            .OrderBy(e => e.Name)
            .Select(e => e);

    return View(query2);
}
```

Select a single object



```
public ActionResult Index()
{
    IEnumerable<Employee> employees = new List<Employee>() {
        new Employee {Id = 1, Name = "Peter Sorensen", HireDate = new DateTime(2008, 10, 10) },
        new Employee {Id = 2, Name = "Torsten Peterson", HireDate = new DateTime(2002, 1, 2) },
        new Employee {Id = 3, Name = "Liza Thompson", HireDate = new DateTime(2011, 12, 4) }
    };

    Employee query1 =
        (from e in employees
        where e.Id == 3
        select e).First();

    Employee query2 =
        employees.Where(e => e.Id == 3)
            .OrderBy(e => e.Name)
            .Select(e => e)
            .First();

    return View(query2);
}
```

Comprehensive Query syntax versus Lamda Query

- Comprehensive Query
  - If you already know SQL, you'll probably find that easier to grasp

- Lamda query
  - Generally offers more control and flexibility
  - Chaining of query operators looks like a pipeline
  - Select operator is optional (when not doing a projection)
  - Many query operators have no comprehensive query equivalent

- It is possible to mix syntaxes

Exercises 1-10

Continued …