Разработка приложений на платформе .NET

Лекция 23

LINQ

Сегодня

- Parallel LINQ
- Language Integrated Query (LINQ)
  - LINQ to XML
  - LINQ to DataSet
  - LINQ to SQL
  - LINQ to Entities

Parallel LINQ

Parallel LINQ

- Выполнение запроса сразу на всех процессорах и ядрах
- Только LINQ to Objects
- IEnumerable<T>.**AsParallel()** - выполнение операций с последовательностью сразу на всех процессорах и ядрах
  - Не сохраняет порядок последовательности

```
ParallelQuery<Complex> complexes =
complexList.AsParallel();
var result = complexes.Where(…);
```

  Добавлены все расширения, аналогичные IEnumerable<T>, принимающие и возвращающие ParallelQuery<T>
- ParallelQuery<T>.**AsOrdered()** – заставляет сохранять порядок последовательности. Дополнительные расходы на синхронизацию

```
ParallelQuery<Complex> complexes =
complexList.AsParallel().AsOrdered().Where(…);
```

- ParallelQuery<T>.**ForAll**(Action<T>) – Выполняет параллельно действие Action<T> над каждым элементом последовательности
  - complexList.AsParallel().ForAll(c => c.Re *= 2);

Демонстрация

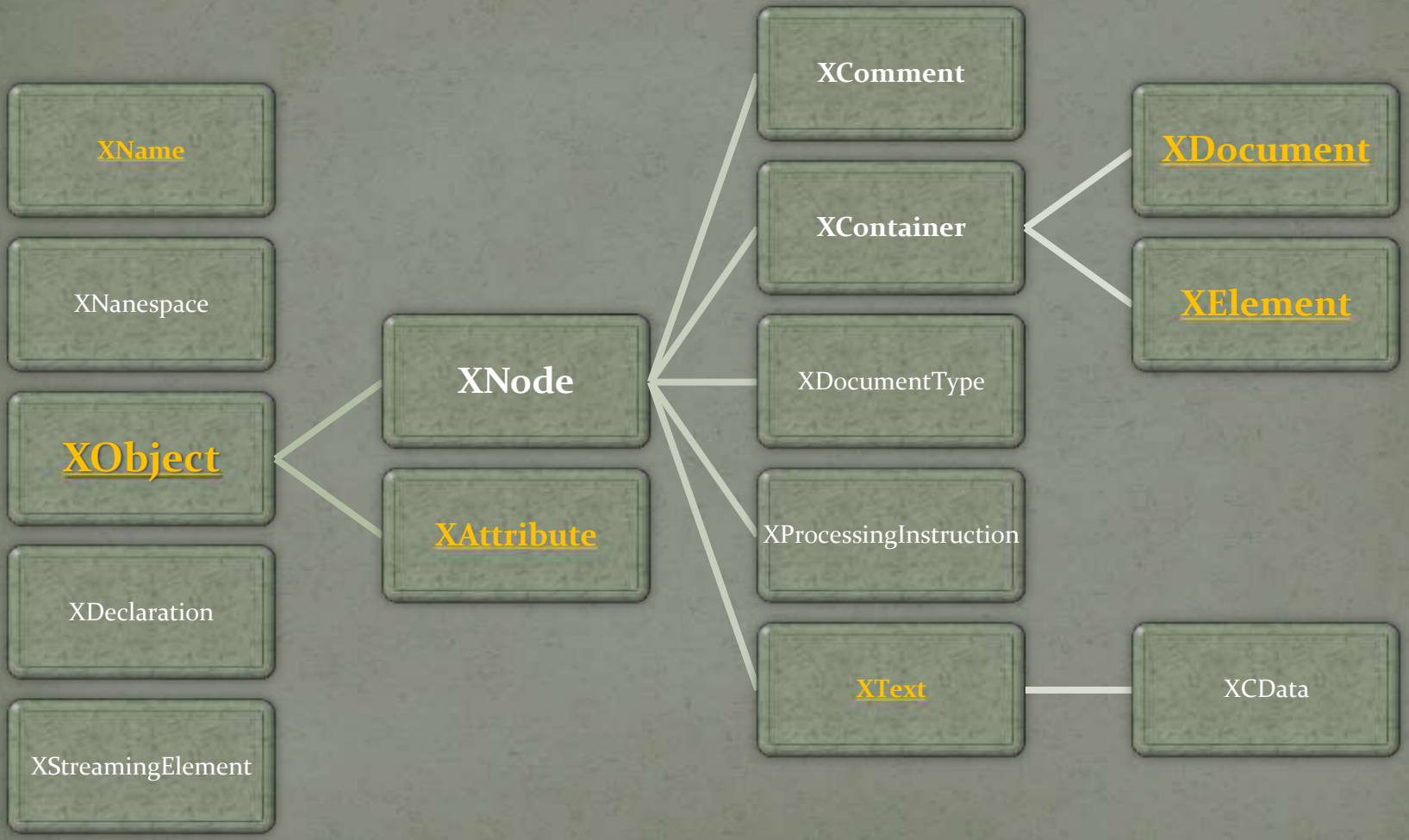
Parallel LINQ

LINQ to XML

LINQ to XML

- Представляет новый интерфейс работы с XML
- В основу старого интерфейса положен XmlDocument (весь XML документ)
  - Для работы с XML всегда нужно создавать XmlDocument и с ним работать
  - Создать остальные типы можно было только используя XmlDocument
- В основу нового интерфейса положен XElement (элемент, а не документ)
- Упрощен механизм создания и оперирования с различными частями XML
  - Все типы имеют публичные конструкторы

Объектная модель



Пространство имен System.Xml.Linq

Создание элемента

- Несколько публичных конструкторов:
  - XElement(XName name);
  - XElement(XName name, params object[] content);
  - XElement(XElement other);

```
XElement students = new XElement("Students",
        new XElement("Student",
            new XAttribute("Year", 2),
            new XElement("FirstName", "Василий"),
            new XElement("LastName", "Петров")),
        new XElement("Student",
            new XComment("Отличник"),
            new XAttribute("Year", 3),
            new XElement("FirstName", "Андрей"),
            new XElement("LastName", "Сидоров")));

Console.WriteLine(students);
```

```
<Students>
  <Student Year="2">
    <FirstName>Василий</FirstName>
    <LastName>Петров</LastName>
  </Student>
  <Student Year="3">
    <!--Отличник-->
    <FirstName>Андрей</FirstName>
    <LastName>Сидоров</LastName>
  </Student>
</Students>
```

- Строка в качестве имени объекта автоматически конвертируется в XName
- Строка в качестве значения элемента автоматически конвертируется в XText

Создание элемента

- Трактовка объектов при добавлении их в элемент

string	Автоматически преобразуется в XText
XText	Текстовое содержимое элемента
XCData	Добавляется как CData
XElement	Дочерний элемент
XAttribute	Атрибут элемента
XComment	Комментарий
IEnumerable	Каждый элемент последовательности обрабатывается и добавляется отдельно
null	Нет дочернего содержимого
Любой (почти) прочий тип	Вызывается ToString() и трактуется как строка

Создание атрибута

- Атрибут представляет собой пару “имя-значение”
- Несколько публичных конструкторов:
  - XAttribute(XAttribute other);
  - XAttribute(XName name, object value);
- Можно создавать сразу добавляя к элементу

```
XElement students = new XElement("Students",
        new XElement("Student",
            new XAttribute("Year", 2),
            new XElement("FirstName", "Василий"),
            new XElement("LastName", "Петров")),
        new XElement("Student",
            new XComment("Отличник"),
            new XAttribute("Year", 3),
            new XElement("FirstName", "Андрей"),
            new XElement("LastName", "Сидоров")));

Console.WriteLine(students);
```

```
<Students>
  <Student Year="2">
    <FirstName>Василий</FirstName>
    <LastName>Петров</LastName>
  </Student>
  <Student Year="3">
    <!--Отличник-->
    <FirstName>Андрей</FirstName>
    <LastName>Сидоров</LastName>
  </Student>
</Students>
```

- Строка в качестве имени атрибута автоматически конвертируется в XName

Создание комментария

- Представляет текстовое значение – комментарий XML
- Несколько публичных конструкторов:
  - XComment(string value);
  - XComment(XComment other);
- Можно создавать сразу добавляя к элементу

```
XElement students = new XElement("Sudents",
        new XElement("Student",
            new XAttribute("Year", 2),
            new XElement("FirstName", "Василий"),
            new XElement("LastName", "Петров")),
        new XElement("Student",
            new XComment("Отличник"),
            new XAttribute("Year", 3),
            new XElement("FirstName", "Андрей"),
            new XElement("LastName", "Сидоров")));

Console.WriteLine(students);
```

```
<Sudents>
  <Student Year="2">
    <FirstName>Василий</FirstName>
    <LastName>Петров</LastName>
  </Student>
  <Student Year="3">
    <!--Отличник-->
    <FirstName>Андрей</FirstName>
    <LastName>Сидоров</LastName>
  </Student>
</Sudents>
```

Создание документа

- Несколько публичных конструкторов:
  - XDocument();
  - XDocument(params object[] content);
  - XDocument(XDeclaration declaration, params object[] content);

```
XDocument doc = new XDocument(new XDeclaration("1.0", "UTF-8", "yes"),
                new XElement("Students", null));
Console.WriteLine(doc);
```

```
<Students />
```

- Объявление XML
  - Класс XDeclaration
  - Может добавляться только к документу
  - XDeclaration(string version, string encoding, string standalone);

XName и XNamespace

- XName
  - Не имеет публичных конструкторов. Автоматически конвертируется из строки
    - XName name = "Students";
  - Состоит из LocalName – имени и пространства имен Namespace

- XNamespace – представляет пространство имен
  - Не имеет публичных конструкторов. Автоматически конвертируется из строки
    - XNamespace aw = "http://www.adventure-works.com";

```
XNamespace aw = "http://www.adventure-works.com";
XElement root = new XElement(aw + "Root",
            new XAttribute("xmlns", "http://www.adventure-works.com"),
            new XElement(aw + "Child", "content"));
 Console.WriteLine(root);
```

```
<Root xmlns="http://www.adventure-works.com">
      <Child>content</Child>
</Root>
```

Демонстрация

Создание XML

Ввод / вывод XML

- Статический метод **Load()**
  - Загружает XML из файла / потока
  - Реализован у XDocumet и XElement

```
XDocument document = XDocument.Load("students.xml");
```

- Статический метод **Parse()**
  - Загружает XML из строки
  - Реализован у XDocumet и XElement

```
XElement students = XElement.Parse("<Students><Student Year=\"3\"/></Students>");
```

- **Save()**
  - Сохраняет XML в файл / поток
  - Реализован у XDocumet и XElement

```
students.Save("students.xml");
```

- **ToString()**
  - Переопределен у всех XObject
  - Возвращает форматированное содержание XML

```
Console.WriteLine(students);
```

Обход деревьев XML

- XContainer.**Element()** – возвращает первый дочерний элемент с указанным именем
- XContainer.**Elements()** – возвращает все дочерние элементы
- XContainer.**Nodes()** – возвращает все дочерние узлы

- XNode.**NextNode** / XNode.**PreviousNode** – свойства содержат предыдущий / следующий узел в дереве

- XObject.**Document** – ссылка на документ
- XObject.**Parent** – ссылка родителя в XML

- XNode.**Ancestors()** – возвращает все родительские элементы (рекурсивный обход XML вверх)
- XElement.**AncestorsAndSelf()** – возвращает все родительские элементы (рекурсивный обход XML вверх) и себя

- XContainer.**Descendants()** – рекурсивно возвращает все дочерние элементы
- XElement.**DescendantsAndSelf()** – рекурсивно возвращает все дочерние элементы и сам элемент

- XNode.**NodesAfterSelf()** / XNode.**NodesBeforeSelf()** – возвращает все узлы после / перед текущим
- XNode.**ElementsAfterSelf()** / XNode.**ElementsBeforeSelf()** – возвращает все элементы после / перед текущим

Изменения XML

- Добавление узлов
  - XContainer.**Add()** – добавляет узел в конец документа или элемента
  - XContainer.**AddFirst()** – добавляет узел в начало документа или элемента
  - XNode.**AddBeforeSelf()** – добавляет узел перед текущим узлом
  - XNode.**AddAfterSelf()** – добавляет узел после текущего узла
- Удаление узлов
  - XNode.**Remove()** – удаление текущего узла
  - **IEnumerable<T>.Remove()** – удаление нескольких узла
  - XElement.**RemoveAll()** – удаление всего содержимого элемента, но не сам элемент
- Обновление узлов
  - XElement.**Value** – задает текстовое значение элемента
  - XElement.**SetElementValue()** – обновляет указанный дочерний элемент
    - создает если такого нет
    - изменяет, если такой есть
    - удаляет, если значение установлено в null
  - XElement.**ReplaceAll()** – заменяет все поддерево элемента на новое

Атрибуты

- Обход атрибутов:
  - XElement.FirstAttribute, XElement.LastAttribute – первый / последний атрибуты элемента
  - XElement.NextAttribute, XElement.PreviousAttribute – следующий / предыдущий атрибуты элемента
  - XElement.**Attribute()** – возвращает атрибут с указанным именем
  - XElement.**Attributes()** – возвращает все атрибуты элемента
- Добавления атрибутов
  - Также как и добавление узлов
- Удаление атрибутов
  - XAttribute.Remove() – удаление текущего атрибута
  - IEnumerable<T>.Remove() – удаление коллекции атрибутов
- Изменения атрибутов
  - XAttribute.Value – значение атрибута

Получение значений

- Для получение значений элементов и атрибутов достаточно просто привести элемент или атрибут к нужному типу

```
XElement firstName = new XElement("FirstName", "Василий");
Console.WriteLine(firstName);
Console.WriteLine((string)firstName));
```

```
<FirstName>Василий</FirstName>
Василий
```

```
XAttribute  year = new XAttribute("Year", 3);
Console.WriteLine(year);
Console.WriteLine((int)year);
```

```
Year=”3”
3
```

Расширения

- Многие методы работы с XML могут быть вызваны и на последовательности:
- **Elements**
  - IEnumerable<XElement> **Elements<T>**(this IEnumerable<T> source) where T : XContainer;
  - IEnumerable<XElement> **Elements<T>**(this IEnumerable<T> source, XName name) where T : XContainer;

```
IEnumerable<XElement> names = students.Elements("Student").Elements("FirstName");
foreach (XElement firstName in names) Console.WriteLine((string)firstName);
```

- **Nodes**
  - IEnumerable<XNode> **Nodes<T>**(this IEnumerable<T> source) where T : XContainer;
- **Attributes**
  - IEnumerable<XAttribute> **Attributes**(this IEnumerable<XElement> source);
  - IEnumerable<XAttribute> **Attributes**(this IEnumerable<XElement> source, XName name);

```
IEnumerable<XAttribute> years = students.Elements("Student").Attributes("Year");
foreach (XAttribute year in years) Console.WriteLine((int)year);
```

Расширения

- **Ancestors**
  - IEnumerable<XElement> Ancestors<T>(this IEnumerable<T> source) where T : XNode;
  - IEnumerable<XElement> Ancestors<T>(this IEnumerable<T> source, XName name) where T : XNode;
  - IEnumerable<XElement> AncestorsAndSelf(this IEnumerable<XElement> source);
  - IEnumerable<XElement> AncestorsAndSelf(this IEnumerable<XElement> source, XName name);
- **Descendants**
  - IEnumerable<XElement> Descendants<T>(this IEnumerable<T> source) where T : XContainer;
  - IEnumerable<XElement> Descendants<T>(this IEnumerable<T> source, XName name) where T : XContainer;

```
XElement sidorov = students.Descendants("Student")
                           .Where(e => (string)e.Element("LastName") == "Сидоров")
                           .FirstOrDefault();
 Console.WriteLine(sidorov);

IEnumerable<XElement> petrov = from s in students.Descendants("Student")
                               where (string)s.Element("LastName") == "Сидоров"
                               select s;
Console.WriteLine(petrov.Single());
```

  - IEnumerable<XElement> DescendantsAndSelf(this IEnumerable<XElement> source);
  - IEnumerable<XElement> DescendantsAndSelf(this IEnumerable<XElement> source, XName name);
  - IEnumerable<XNode> DescendantNodes<T>(this IEnumerable<T> source) where T : XContainer;
  - IEnumerable<XNode> DescendantNodesAndSelf(this IEnumerable<XElement> source);
- Удаление узлов и атрибутов
  - void Remove<T>(this IEnumerable<T> source) where T : XNode;
  - void Remove(this IEnumerable<XAttribute> source);
- Сортировка в порядке присутствия в документе
  - IEnumerable<T> InDocumentOrder<T>(this IEnumerable<T> source) where T : XNode;

Демонстрация

Запросы к XML

Дополнительные возможности

- Проверка достоверности XML документа
- Получение и работа с XSD схемой
- XSLT преобразование
- Использование XPath для навигации по XML

LINQ to DataSet

LINQ to DataSet

- Представляет дополнительные методы для работы с DataTable и с DataRow
- Имеет некоторые особенности при использовании стандартных LINQ запросов

Операции с множеством DataRow

- DataTable.AsEnumerable() – позволяет выполнять LINQ запросы с множеством строк
- Знакомые методы
  - Distinct()
  - Except()
  - Intersect()
  - Union()
  - SequenceEqual()
- Но по умолчанию сравнивают ссылки на строки, что приведет к неправильному результату.
- Для сравнения значений полей необходимо использовать специальный Comparer:
  - **System.Data.DataRowComparer.Default**
  - IEnumerable<DataRow> dictinctRows = dataTable.AsEnumerable().Distinct(DataRowComparer.Default);
- CopyToDataTable<DataRow> - копирует строки в новую таблицу

Операции с полями DataRow

- При хранении значений в строке используется тип object.
  - Поэтому при сравнении row1[“Id”] == row2[“Id”] будет получен неверный результат, из-за boxing.
  - Необходимо сравнивать (int)row1[“Id”] == (int)row2[“Id”]
- В таблице могут хранится DBNull, которые также вызываю сложности при преобразовании
- Метод **Field<T>()** - создан для правильного преобразования
  - row1.Field<int>(“Id”) == row2.Field<int>(“Id”)
  - Выполняет преобразование типов
  - Контролирует и преобразует DBNull
- Метод **SetField<T>()** - создан для правильной установки значений полей DataRow. Преобразует null в DBNull, если необходимо.

Демонстрация

LINQ to DataSet

LINQ to SQL

LINQ to SQL

- Необходима генерация специальных сущностных классов для LINQ to SQL
  - Либо с использованием Visual Studio (шаблон LINQ to SQL)
  - Либо с использованием утилиты SQLMetal для генерации сущностных классов для всех таблиц базы данных
- DataContext – управляет подключением к базе данных
- Запросы к базе данных выполняются прозрачным образом
- Проявляется вся мощь отложенных запросов
  - Некоторые запросы могут вообще не выполнятся
- Выражение LINQ проецируется на SQL запрос
- Запросы оперируют IQueryable<T>. Представляют собой деревья выражений

Создание

- Шаблон LINQ to SQL classes
- Создание соединения
- Перетаскивание необходимых таблиц

- Вместо генеренных запросов на insert/delete/update можно использовать хранимые процедуры

- **DataContext** - Отвечает за соединение, слежение за изменениями и т.д.

Использование

```
NorthwindDataContext dc = new NorthwindDataContext ();
var customers = from o in dc.Orders
                where o.EmployeeID > 5
                select o.CustomerID;
foreach (var o in customers )
{
    Console.WriteLine(o);
}
```

- Автоматическое построение “оптимального” запроса

```
SELECT [to].[CustomerID]
FROM [dbo].[Orders] AS [to]
WHERE [to].[EmployeeID] > @p0
-- @p0: Input Int (Size = -1; Prec = 0; Scale = 0) [5]
-- Context: SqlProvider(Sql2008) Model: AttributedMetaModel Build: 4.0.30319.17929
```

- Отложенное выполнение запроса. Запрос будет выполняться только в момент использования
- Есть возможность отложенной загрузки или пред загрузки
- Возможности использования фильтрации, сортировки, join, группировок и агрегаций

Операции с записями

- Сохранение изменений в базе данных
  - На коллекции SubmitChanges()
  - dc. SubmitChanges()
- Добавление
  - На коллекции сущностей InsertOnSubmit()
  - dc.Orders.InsertOnSubmit(new Order{….})
  - Для сохранения изменений в базе необходимо вызвать SubmitChanges() на DataContext
- Удаление
  - На коллекции сущностей DeleteOnSubmit()
  - dc.Orders.DeleteOnSubmit(order)
  - Для сохранения изменений в базе необходимо вызвать SubmitChanges() на DataContext

Демонстрация

LINQ to SQL

Entity Framework

Entity Framework

- ORM (object relational mapping)
  - В коде работаем с объектами
  - В базе с таблицами, индексами, первичными и внешними ключами, и .т.д.
- Распространяется как NuGet Package.
- Текущая версия 6.2.0
- Open Source проект - https://github.com/aspnet/EntityFramework6

- Широкое управление классами (нарушение соответствия класс = таблица)
- Поддержка наследования (различных моделей хранения наследования)
- Возможность сначала создать модель, а потом по ней сгенерировать базу данных (Code First)

Entity Framework

- Entity – сущность, набор данных, ассоциированных с объектом
- Entity Data Model – модель, сопоставляет сущности с реальными таблицами БД
  - Концептуальный уровень. Определение сущностей (классов)
  - Уровень хранилища. Определяет структуру БД (таблицы, отношения между таблицами, типы данных и т.д.)
  - Mapping. Сопоставление межу сущностями, их свойствами и таблицами, столбцами в БД
- Entity Data Model позволяет, оперируя с объектами (сущностями), взаимодействовать с данными в таблицах БД

- Подходы при работе с БД
  - Database First
    - фактически устарел. Функциональность включена в Code First
  - Model First
    - фактически устарел. Функциональность включена в Code First
  - Code First
    - Сначала классы – по ним генерация БД
    - По готовой БД – генерация классов (“Code Second”)

Code First по готовой БД

- Шаблон в Visual Studio
  - ADO.NET Entity Data Model -> Code First from Database
- Генерируются классы сущностей. Сущность по каждой таблице*

```
public partial class Order
{
    public int OrderID { get; set; }
    public string CustomerID { get; set; }
....}
```

- В сущности добавляются навигационные свойства
  - public virtual Customer Customer { get; set; }
  - public virtual ICollection<Order_Detail> Order_Details { get; set; }
- Генерируется наследник от DbContext. Добавляются свойства DbSet<T> для определение коллекций сущностей (таблиц)

```
public partial class NortwindContext : DbContext
{
    public virtual DbSet<Order_Detail> Order_Details { get; set; }
    public virtual DbSet<Order> Orders { get; set; }
    public virtual DbSet<Product> Products { get; set; }
...}
```

- В методе OnModelCreating(DbModelBuilder modelBuilder) настраивает модель (маппинг, особенности свойств и т.д.)
- В конфигурационный файл добавляется строка подключения с именем класса, наследника от DbContext

Использование EF

- Подключение к БД – создание контекста.
  - Строка подключения возьмётся из конфигурационного файла
  - Dispose контекста – закрытие соединения с БД
- Обращение с БД в стиле LINQ.
- Можно оперировать навигационными свойствами.

```
using (UniversityContext context = new UniversityContext())
{
    context.People.Add(new Person() {Name = "Vasja", Age =
                                        DateTime.Now.AddYears(-1)});
    context.SaveChanges();
    foreach (var person in context.People)
    {
        Console.WriteLine($"{person.Id} - {person.Name} - {person.Age}");
    }
}
```

- SQL запросы будут строиться автоматически

Загрузка данных

- Lazy loading (по умолчанию)
  - Загрузка сущностей по мере требования.
  - Удобно, экономично, но могут быть проблемы с производительностью
- Eager loading
  - Include() – приводит к принудительной загрузке указанной сущности при первом использовании

```
using (NortwindContext context = new NortwindContext())
{
    foreach (var order in context.Orders.Take(10).Include(o => o.Customer))
    {
        Console.WriteLine($"{order.OrderID} - {order.Customer.CompanyName}");
    }
}
```

- Explicit loading
  - Load() – вызывает немедленную загрузку, выполение запроса. Как ToList(), только без сохранения коллекции, а только с заполнением кеша

Демонстрация

Code First по готовой БД

Изменение данных

- DbContext следит за изменениями данных
- Добавление данных Add(), AddRange(), AddOrUpdate()
  - В коллекцию сущностей
    - context.People.Add(new Person() {Name = "Vasja", Age = DateTime.Now.AddYears(-1)});
    - context.SaveChanges();
- Изменение данных
    - context.People.First(p => p.Id == 23).Name = "Sasha“;
    - context.SaveChanges();
- Удаление данных. Remove(), RemoveRange()
    - context.People.Remove(context.People.First(p => p.Id == 23));
    - context.SaveChanges();
- Сохранение изменений
    - context.SaveChanges();

Code First

- Шаблон в Visual Studio
  - ADO.NET Entity Data Model -> Empty Code First Model
- Нужно создать классы сущностей

```
public class Person
{
    public int Id { get; set; }
    public string Name { get; set; }
    public DateTime Age { get; set; }
}
```

- В сгенерированный наследник от DbContext нужно добавить свойства DbSet<T> для определение коллекций сущностей (таблиц)

```
public class UniversityContext : DbContext
{
    public University() : base("name=University"){}
    public DbSet<Person> People { get; set; }
}
```

- В конфигурационный файл нужно добавить строку подключения с именем класса или с именем, переданным в конструктор DbContext
- Далее можно использовать Context как и ранее.
- Если БД не существует при старта программы, при создании контекста БД будет создана автоматически

Демонстрация

Code First