Платформа ADO.NET Entity Framework

1 Общие сведения о локальных данных

При использовании локальных данных приложение подключается к файлу базы данных на локальном компьютере, а не к базе данных на отдельном сервере. Например, можно подключить приложение, которое разрабатывается в Visual Studio, к следующим файлам локальной базы данных:

- Файлы базы данных SQL Server Express (MDF)
- Файлы базы данных Microsoft Access (MDB)

SQL Server Express LocalDB и SQL Server Express

Можно добавить файл базы данных, основанной на службе, (MDF) в любой проект в Visual Studio. Для разработки таблиц и других объектов базы данных, а также выполнения запросов можно использовать конструкторы в Visual Studio.

При создании в Visual Studio базы данных, основанной на службах, для доступа к файлу базы данных (MDF) она использует подсистему SQL Server Express LocalDB, где в более ранних версиях Visual Studio использовался обработчик экспресс-выпуска SQL Server.

SQL Server Express LocalDB — это упрощенная версия SQL Server, которая обладает множеством возможностей по программированию баз данных, аналогичных SQL Server. SQL Server Express LocalDB выполняется в пользовательском режиме, и его можно установить более быстро, с меньшим числом требований и без конфигурации.

Примечание. Дополнительные сведения о SQL Server Express LocalDB см. в статьях Вводные сведения о LocalDB, усовершенствованной версии SQL Express и LocalDB: где моя база данных? на веб-сайте Microsoft.

В Visual Studio (в Visual Studio 2010 и более ранних версиях) можно использовать SQL Server Express по умолчанию вместо SQL Server Express LocalDB. В строке меню выберите **Сервис**,**Параметры**. В узле **Инструменты базы данных** и выберите **Подключения данных**. В поле **Имя экземпляра SQL Server** введите**SQLEXPRESS**. В качестве альтернативы можно ввести другие значения для имени экземпляра SQL Server (например, **SQL2008**).

В следующей таблице описаны различия между командами SQL Server Express LocalDB и SQL Server Express.

Таблица 1 - Описание команд SQL Server Express LocalDB и SQL Server Express

Описание	**SQL Server Express LocalDB**	**SQL Server Express**
Тип базы данных при создании базы данных, основанной на службах	В Visual Studio 2012 и Visual Studio 2013 — SQL Server Express LocalDB	В Visual Studio 2010 и более ранних версиях — SQL Server Express
Имя экземпляра SQL Server в средствах и параметрах	(LocalDB)\v11.0	SQLEXPRESS
Значение источника данных в строке подключения	(LocalDB)\v11.0	.\SQLEXPRESS
Значение AttachDbFilename в строке подключения	путь к файлу	путь к файлу
Требуется пользовательский экземпляр ("User Instance=True" в строке подключения)	Нет	Да
Расширение файла базы данных	MDF	MDF

Преимущества SQL Server Express LocalDB

- SQL Server Express LocalDB совместима с основанными на службах выпусками SQL Server для функций, включаемых SQL Server Express LocalDB. В SQL Server можно переместить все базы данных или код Transact-SQL из SQL Server Express LocalDB в SQL Server или SQL Azure, не выполняя никаких действий по обновлению. Поэтому можно использовать SQL Server Express LocalDB для разработки приложений, ориентированных на все выпуски SQL Server.
- SQL Server Express LocalDB поддерживает те же оптимизатор запросов и средство обработки запросов, что и более высокие выпуски SQL Server.

Способы взаимодействия с БД

Entity Framework предполагает три возможных способа взаимодействия с базой данных:

- **Database first**: Entity Framework создает набор классов, которые отражают модель конкретной базы данных
- **Model first**: сначала разработчик создает модель базы данных, по которой затем Entity Framework создает реальную базу данных на сервере.

- **Code first**: разработчик создает класс модели данных, которые будут храниться в бд, а затем Entity Framework по этой модели генерирует базу данных и ее таблицы

2 Технология Entity Framework

2.1 Общие сведения о платформе Entity Framework.

Entity Framework (EF) — это рекомендуемая Майкрософт технология доступа к данным для новых приложений.

Entity Framework (EF) — это объектно-реляционный модуль сопоставления, позволяющий разработчикам .NET работать с реляционными данными с помощью объектов, специализированных для доменов. Это устраняет необходимость в написания большей части кода для доступа к данным, который обычно требуется разработчикам.

Представление Entity Framework

Entity Framework позволяет создать модель, написав код или используя поля и строки в конструкторе Entity Framework. Оба этих подхода можно использовать для работы с существующей базой данных или создания новой базы данных.

2.1 Применение моделей на практике

Многолетним и общим подходом к разработке является подход, при котором построение приложения или службы представляет собой его разделение на три части: модель домена, логическую модель и физическую модель.

Модель домена определяет сущности и связи в моделируемой системе.

Логическая модель для реляционной базы данных обеспечивает нормализацию сущностей и связей в целях создания таблиц с ограничениями внешнего ключа.

В физической модели учитываются возможности конкретной системы обработки данных путем определения зависящих от ядра базы данных подробных сведений о хранении данных, которые касаются секционирования и индексирования.

Физическая модель совершенствуется администраторами базы данных в целях повышения производительности, но программисты, которые разрабатывают код приложения, в основном вынуждены ограничиваться работой с логической моделью, подготавливая SQL-запросы и вызывая хранимые процедуры.

Модели домена в основном используются как инструмент для представления и обмена мнениями о требованиях к приложению, поэтому чаще всего служат в качестве практически не изменяющихся схем, которые рассматриваются и обсуждаются на ранних стадиях проекта, после чего выходят из сферы внимания. Во многих коллективах разработчиков принято

пропускать этап создания концептуальной модели и начинать с определения таблиц, столбцов и ключей в реляционной базе данных.

Платформа Entity Framework придает значимость *моделям*, позволяя разработчикам выполнять запросы к сущностям и связям в *модели домена* (которая называется *концептуальной моделью* в Entity Framework), при этом для перевода этих операций в команды, определяемые источником данных, используется сама платформа Entity Framework.

Это позволяет отказаться от применения в приложениях жестко заданных зависимостей от конкретного источника данных.

При работе в режиме Code First концептуальная модель сопоставлена с моделью хранения в коде. Entity Framework может вывести концептуальную модель, основанную на типах объектов и дополнительных конфигурациях, которые можно задать. Метаданные сопоставления формируются во время выполнения на основе сочетания определений типов домена и дополнительной информации о конфигурации, которая указана в коде. Entity Framework при необходимости создает базу данных на основе метаданных. Дополнительные сведения см. в разделе Создание и сопоставление концептуальной модели.

При работе со средствами работы с моделью EDM концептуальная модель, модель хранения и сопоставление между ними выражены в схемах на основе XML и определены в файлах с именами с соответствующими расширениями.

2.3 Введение в разработку Database First

Модели и БД. Все сущности в приложении принято выделять в отдельные модели. В зависимости от поставленной задачи и сложности приложения можно выделить различное количество моделей. Так, в тестовом приложении из второй главы использовались две модели - класс для книги и класс для покупки книги.

Модели представляют собой простые классы и располагаются в проекте в каталоге *Models*. Модели описывают логику данных. Например, модель представляющая книгу и ее покупку:

```
public class Book
{
   // ID книги
   public int Id { get; set; }
   // название книги
   public string Name { get; set; }
   // автор книги
   public string Author { get; set; }
   // цена
   public int Price { get; set; }
}

public class Purchase
{
   public int PurchaseId { get; set; }
```

```
    public string Person { get; set; }
    public string Address { get; set; }
    public int BookId { get; set; }
    public DateTime Date { get; set; }
}
```

Модель необязательно состоит только из свойств, кроме того, она может иметь конструктор, какие-нибудь вспомогательные методы. Но главное не перегружать класс модели и помнить, что его предназначение - описывать данные.

Манипуляции с данными и бизнес-логика - это больше сфера контроллера.

Данные моделей, как правило, хранятся в базе данных. Для работы с базой данных очень удобно пользоваться фреймворком **Entity Framework**, который позволяет абстрагироваться от написания sql-запросов, от строения базы данных и полностью сосредоточиться на логике приложения.

Если при создании проекта MVC 5 вы выберете в качестве типа аутентификации "No Authentication", то после создания проекта в его надо будет подключить EntityFramework через пакетный менеджер NuGet.

В качестве альтернативы NuGet можно использовать консоль пакетного менеджера. Для этого в меню Visual Studio выберем View -> Other Windows -> Package Manager Console. После этого внизу студии откроется консоль пакетного менеджера. В ней введем такую команду:

```
PM> Install-Package EntityFramework -Version 6.0.2
```

После ввода команды будет загружен и установлен пакет Entity Framework. Иногда этой консолью предпочтительнее пользоваться при установке пакетов, чем менеджером NuGet, так как менеджер NuGet может немного опаздывать за выпуском последних версий пакетов. Либо наоборот, нам надо установить пакеты более ранней версии, а NuGet может предложить только текущую версию.

Entity Framework поддерживает подход "Code first", который предполагает сохранение или извлечение информации из БД на SQL Server без создания схемы базы данных или использования дизайнера в Visual Studo. Наоборот, мы создаем обычные классы, а Entity Framework уже сам определяет, как и где сохранять объекты этих классов.

Для подключения к базе данных через Entity Framework, нам нужен посредник - **контекст данных**. Контекст данных представляет собой класс, производный от класса **DbContext**. Контекст данных содержит одно или несколько свойств типа DbSet<T>, где T представляет тип объекта, хранящегося в базе данных. Например, создадим контекст данных для работы с вышеприведенными моделями:

Для хранения данных приложению нужна база данных. Мы можем использовать различные СУБД, но, как правило, в качестве базы данных в связке с ASP.NET MVC используется база данных MS SQL Server, на

примере которого мы и посмотрим весь процесс создания БД и подключения к ней.

Мы можем создать базу данных прямо в проекте, либо же создать ее на сервере MS SQL. Для хранения баз данных проекте предназначена папка **App_Data**. Для этого нажмем на папку App_Data правой кнопкой мыши и в появившемся контекстном меню выберем **Add-> New Item...**. В появившемся окне добавления нового элемента необходимо выбрать **SQL Server Database** и указать название новой базы данных:

После этого в папку папку App_Data будет добавлена база данных, и мы можем начинать с ней работать - добавлять таблицы и данные.

2.4 Database First был первым подходом, который появился в Entity Framework. Данный подход во многом похож на Model First и подходит для тех случаев, когда разработчик уже имеет готовую базу данных.

Чтобы Entity Framework мог получить доступ к базе данных, в системе должен быть установлен соответствующий провайдер. Так, Visual Studio уже поддерживает соответствующую инфраструктуру для СУБД MS SQL Server. Для остальных СУБД, например, MySQL, Oracle и других надо устанавливать соответствующие провайдеры. Список провайдеров для наиболее распространенных СУБД можно найти на странице ADO.NET Data Providers.

Итак, создадим новый проект. Его функциональность будет той же самой, что и у предыдущих проектов, только подход к использованию Entity Framework будет отличаться.

После создания нового проекта, чтобы задействовать базу данных, нам надо ее иметь. Создадим новую бд или возьмем уже имеющуюся.

В Visual Studio в окне Solution Explorer нажмем на проект правой кнопкой мыши и выберем в **Add - > New Item**. Далее в появившемся окне добавления нового элемента выберем **ADO.NET Entity Data Model**. Дадим новому компоненту какое-либо название (рис.1)

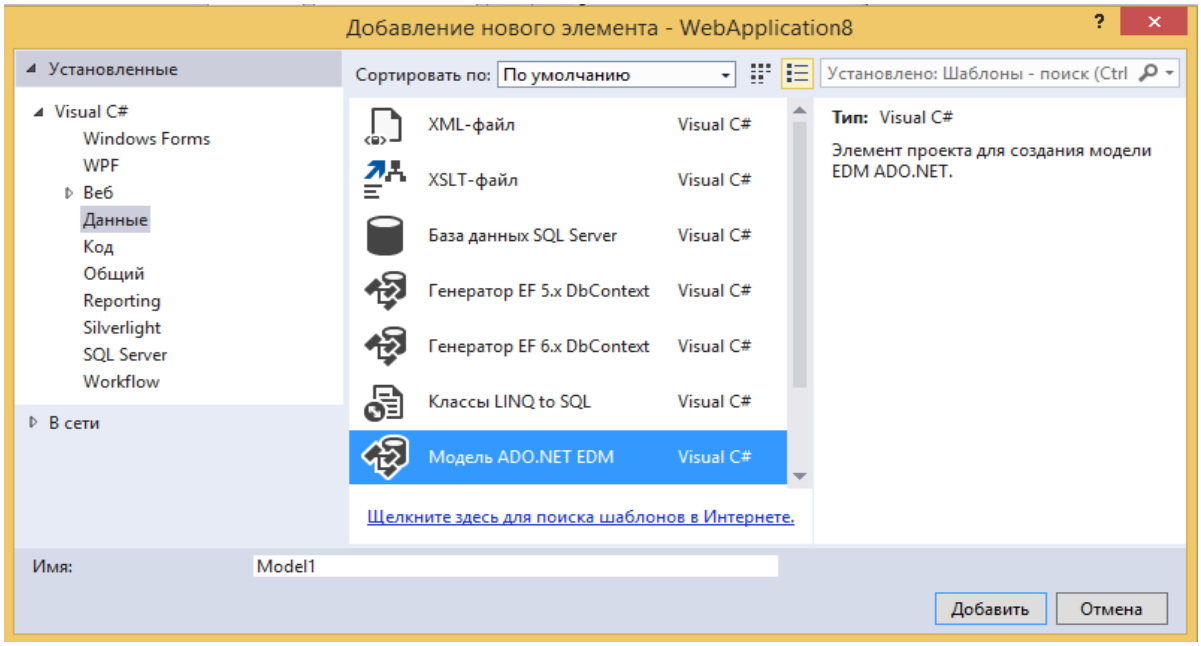

Рисунок 1 – Пример добавления компонента модели

После этого нам откроется окно мастера создания модели. Если вы работаете с Visual Studio 2013 с пакетом обновления SP2, SP3, то откроется следующее окно мастера модели (Generate from database):

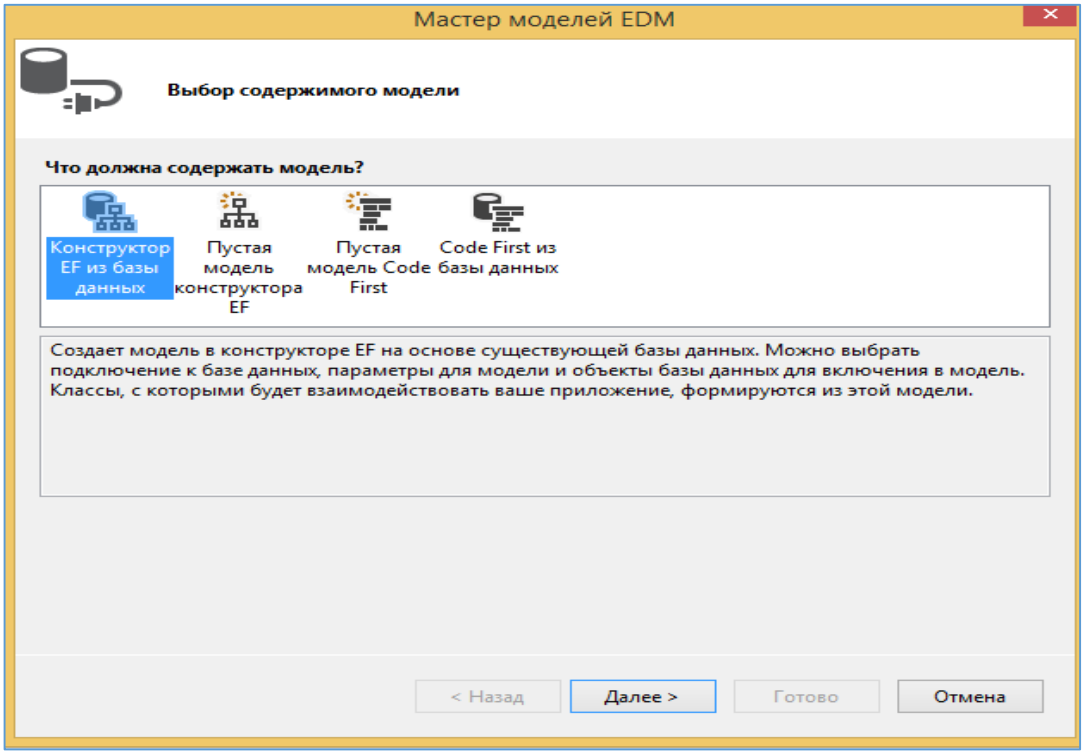

Рисунок 2 – Мастер модели EDM

В качестве подхода взаимодействия с БД выбирается Database First.

Затем откроется окно следующего шага по созданию модели, на котором надо будет установить подключение к базе данных:

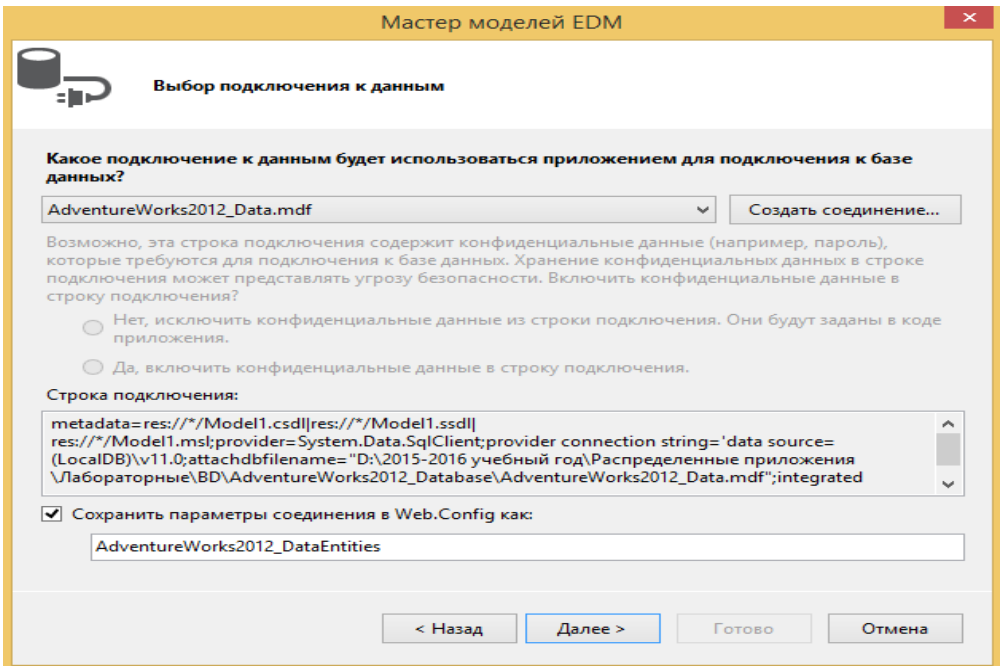

Рисунок 3 – Окно подключения к базе данных

В выпадающем списке выберем одно из доступных подключений. Если в списке нет предпочтительных подключений, то можно нажать на кнопку **New Connection** и установить новое подключение.

Также внизу указывается название **контекста данных**, который будет использоваться для доступа к данным. По умолчанию у меня контекст имеет название **userstoredbEntities**. Можно изменить, а можно и оставить.

Выбрав подключение, переходим к следующему шагу. Если у нас Visual Studio 2013 без пакетов обновления, то будет предложено также выбрать версию Entity Framework. Выберем шестую версию:

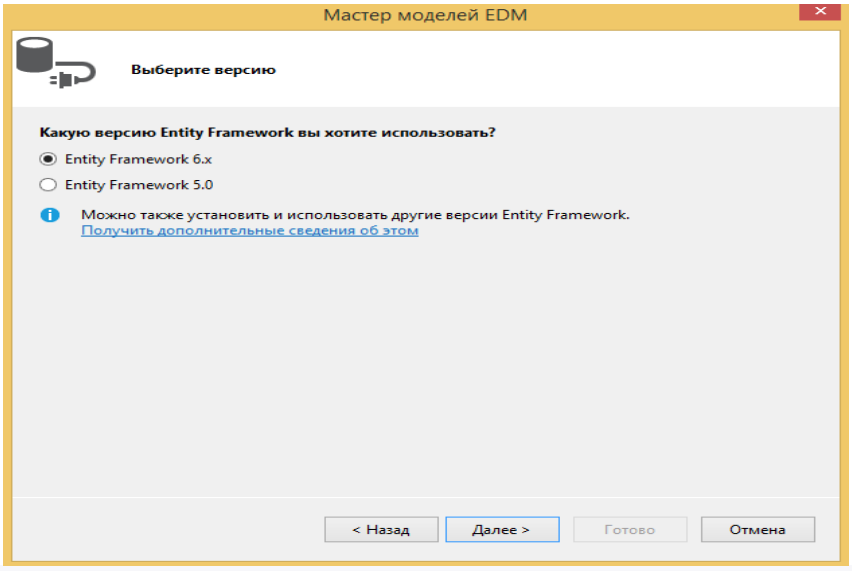

Рисунок 4 – Окно выбора версии Entity Framework

В версиях Visual Studio 2013 SP2, SP3 по умолчанию используется EF 6, поэтому этот шаг пропускается.

Далее Visual Studio извлекает всю информацию о базе данных:

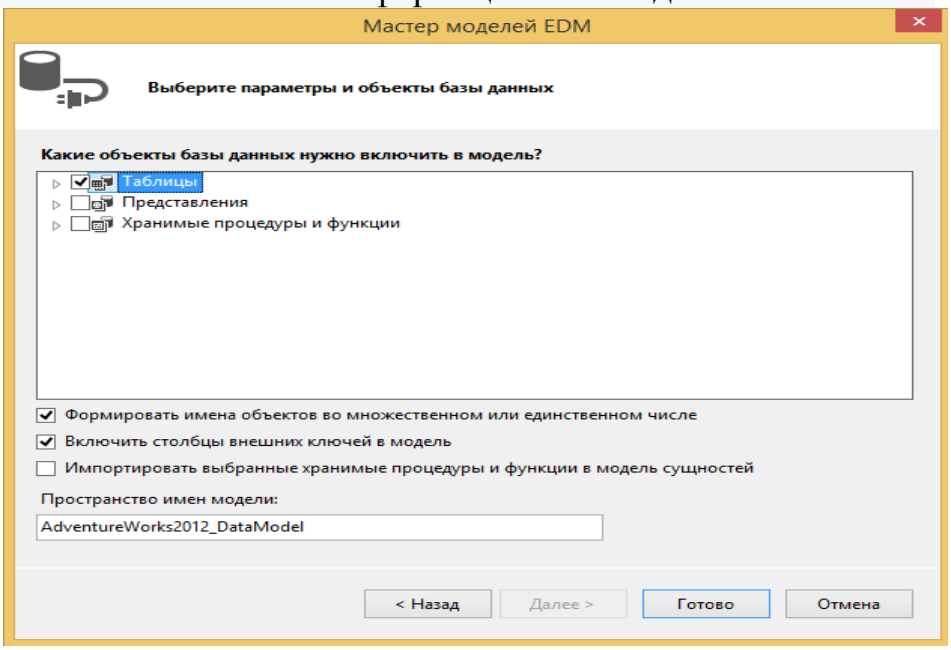

Рисунок 5 – Окно выбора объектов БД

Раскроем узел Tables. Он отображает все таблицы, имеющиеся в базе данных. В моем случае имеется только одна таблица Users. Отметим все подузлы в ветке Tables.

В поле **Model Namespace** установим предпочтительное имя модели и нажмем Finish.

После этого Entity Framework сгенерирует модель по базе данных и добавит ее в проект. Visiual Studio отобразит нам схему модели, отображаются все сущности.

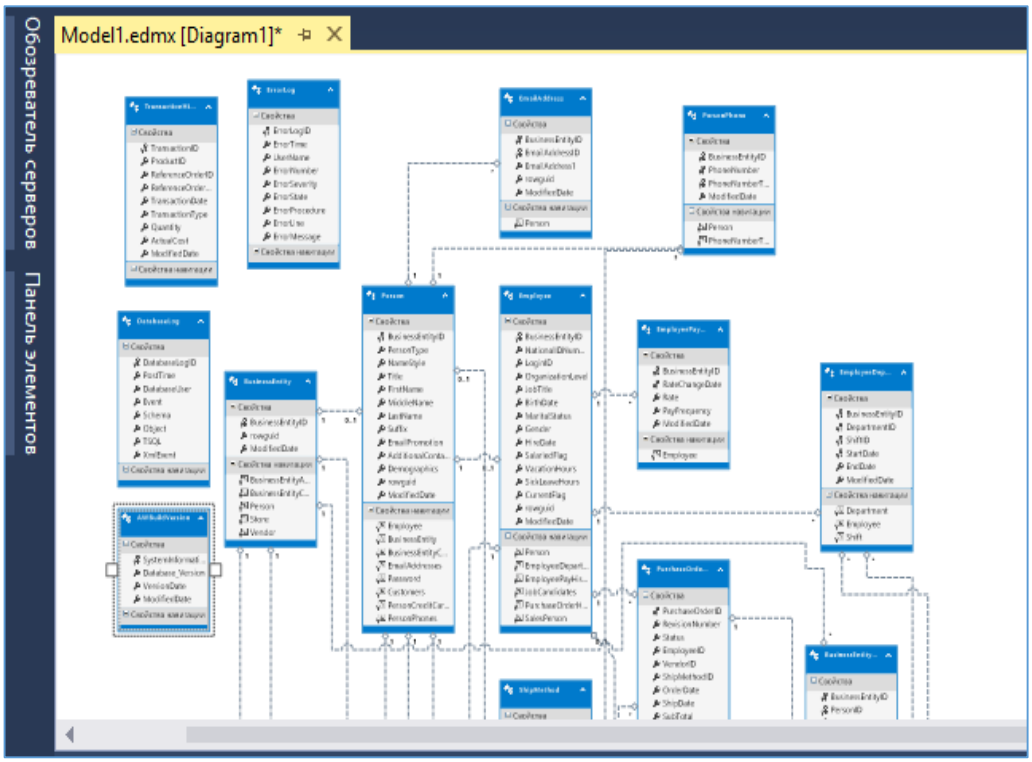

Рисунок 6 – Окно схемы сгенерированной модели БД

После выделения сущности в правом нижнем углу Visual Studio мы увидим свойства для этой сущности:

2.5 База первых позволяет перепроектировать модели из существующей базы данных. Модель хранится в EDMX файла (EDMX-расширения) и могут быть просмотрены и отредактированы в Entity Framework Designer. Классы, которые вы взаимодействуете с в вашем приложении автоматически генерируются из файла EDMX.

Большинство операций с данными представляют собой CRUD-операции (Create, Read, Update, Delete), то есть получение данных, создание, обновление и удаление. Entity Framework позволяет легко производить данные операции.

Приведем описание секции connectionStrings файла конфигурации Web.config:

```
<connectionStrings>
  <add name="TestData" connectionString="Data Source=(LocalDB)\v11.0;AttachDbFilename='D:\2015-2016 учебный год\Распределенные приложения\Лабораторные\BD\AdventureWorks2012_Database\AdventureWorks2012_Data.mdf'; Integrated Security=True" />
  <add name="AdventureWorks2012_DataEntities" connectionString="metadata=res://*/Model1.csdl|res://*/Model1.ssdl|res://*/Model1.msl;provider=System.Data.SqlClient;provider connection string=&quot;data source=(LocalDB)\v11.0;attachdbfilename=|DataDirectory|\AdventureWorks2012_Data.mdf;integrated security=True;connect timeout=30;MultipleActiveResultSets=True;App=EntityFramework&quot;" providerName="System.Data.EntityClient" />
  <add name="AdventureWorksConnection" connectionString="metadata=res://*/Model1.csdl|res://*/Model1.ssdl|res://*/Model1.msl;provider=System.Data.SqlClient;provider connection string='data source=(LocalDB)\v11.0;attachdbfilename=&quot;D:\2015-2016 учебный год\Распределенные приложения\Лабораторные\BD\AdventureWorks2012_Database\AdventureWorks2012_Data.mdf&quot;;integrated security=True;connect timeout=30;MultipleActiveResultSets=True;App=EntityFramework'"
```

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

Приведем класс контекста данных в проекте - Model1.Context.cs:

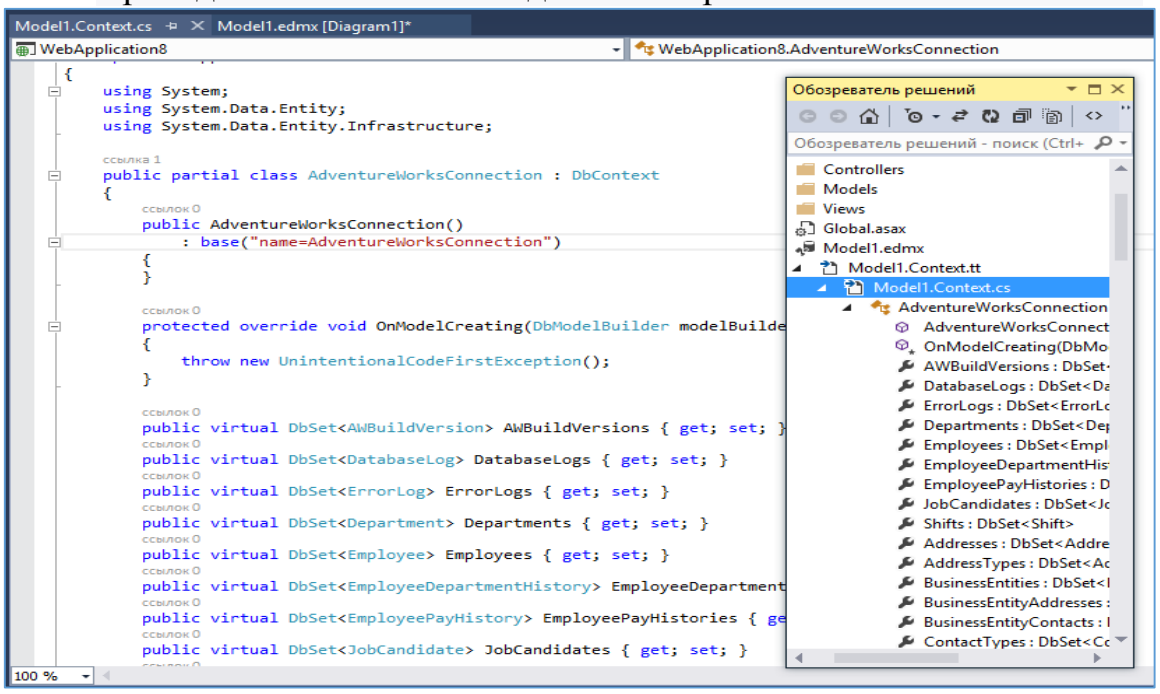

Рисунок 7 – Окно класс контекста данных

Он имеет два конструктора. Статический конструктор призван выполнить начальную инициализацию данных. В данном случае он ничего не выполняет.

Стандартный конструктор обращается к конструктору базового класса (то есть класса DbContext) и передает ему название строки подключения (*base("name=AdventureWorksConnection")*).

Для взаимодействия с таблицами класс контекста данных имеет одноименные **свойства,** например, *Departments*:

```
public virtual DbSet<Department> Departments { get; set; }
```

Приведем класс описания модели сущности Department:

```
public partial class Department
    {
        public Department()
        {
            this.EmployeeDepartmentHistories = new HashSet<EmployeeDepartmentHistory>();
        }

        public short DepartmentID { get; set; }
        public string Name { get; set; }
        public string GroupName { get; set; }
        public System.DateTime ModifiedDate { get; set; }

        public virtual ICollection<EmployeeDepartmentHistory> EmployeeDepartmentHistories { get; set; }
    }
```

Обычные свойств класса *Department: DepartmentID, Name, GroupName, ModifiedDate*. Свойство *DepartmentID* таблицы *Department* используется в качестве **первичного ключа**.

Приведем класс описания модели сущности *EmployeeDepartmentHistor*:

```
public partial class EmployeeDepartmentHistory
    {
        public int BusinessEntityID { get; set; }
        public short DepartmentID { get; set; }
        public byte ShiftID { get; set; }
        public System.DateTime StartDate { get; set; }
        public Nullable<System.DateTime> EndDate { get; set; }
        public System.DateTime ModifiedDate { get; set; }

        public virtual Department Department { get; set; }
        public virtual Employee Employee { get; set; }
        public virtual Shift Shift { get; set; }
    }
```

Соответственно, приведем описание таблицы *EmployeeDepartmentHistor:*

```
CREATE TABLE [HumanResources].[EmployeeDepartmentHistory] (
    [BusinessEntityID] INT      NOT NULL,
    [DepartmentID]     SMALLINT NOT NULL,
    [ShiftID]          TINYINT  NOT NULL,
    [StartDate]        DATE     NOT NULL,
    [EndDate]          DATE     NULL,
    [ModifiedDate]     DATETIME CONSTRAINT [DF_EmployeeDepartmentHistory_ModifiedDate]
DEFAULT (getdate()) NOT NULL,
    CONSTRAINT [PK_EmployeeDepartmentHistory_BusinessEntityID_StartDate_DepartmentID] PRIMARY
KEY CLUSTERED ([BusinessEntityID] ASC, [StartDate] ASC, [DepartmentID] ASC, [ShiftID] ASC),
    CONSTRAINT [FK_EmployeeDepartmentHistory_Department_DepartmentID] FOREIGN KEY
([DepartmentID]) REFERENCES [HumanResources].[Department] ([DepartmentID]),
    CONSTRAINT [FK_EmployeeDepartmentHistory_Employee_BusinessEntityID] FOREIGN KEY
([BusinessEntityID]) REFERENCES [HumanResources].[Employee] ([BusinessEntityID]),
    CONSTRAINT [FK_EmployeeDepartmentHistory_Shift_ShiftID] FOREIGN KEY ([ShiftID])
REFERENCES [HumanResources].[Shift] ([ShiftID]),
    CONSTRAINT [CK_EmployeeDepartmentHistory_EndDate] CHECK ([EndDate]>=[StartDate] OR
[EndDate] IS NULL)
);
```

2.6 Внешний ключ. **Внешний ключ** определяется как обычное свойство, например, свойство *DepartmentID* таблицы *EmployeeDepartmentHistory*. Внешний ключ позволяет получать связанные данные.

При определении внешнего ключа нужно иметь в виду следующее. Если тип обычного свойства во внешнем ключе определяется как *int?*, то есть допускает значения null, то при создании базы данных соответствующее поле будет принимать значения NULL, например, [TeamId] INT NULL.

Для описания связи **один-ко-многим** (one-to-many) между таблицами используются навигационные свойства рис (222).

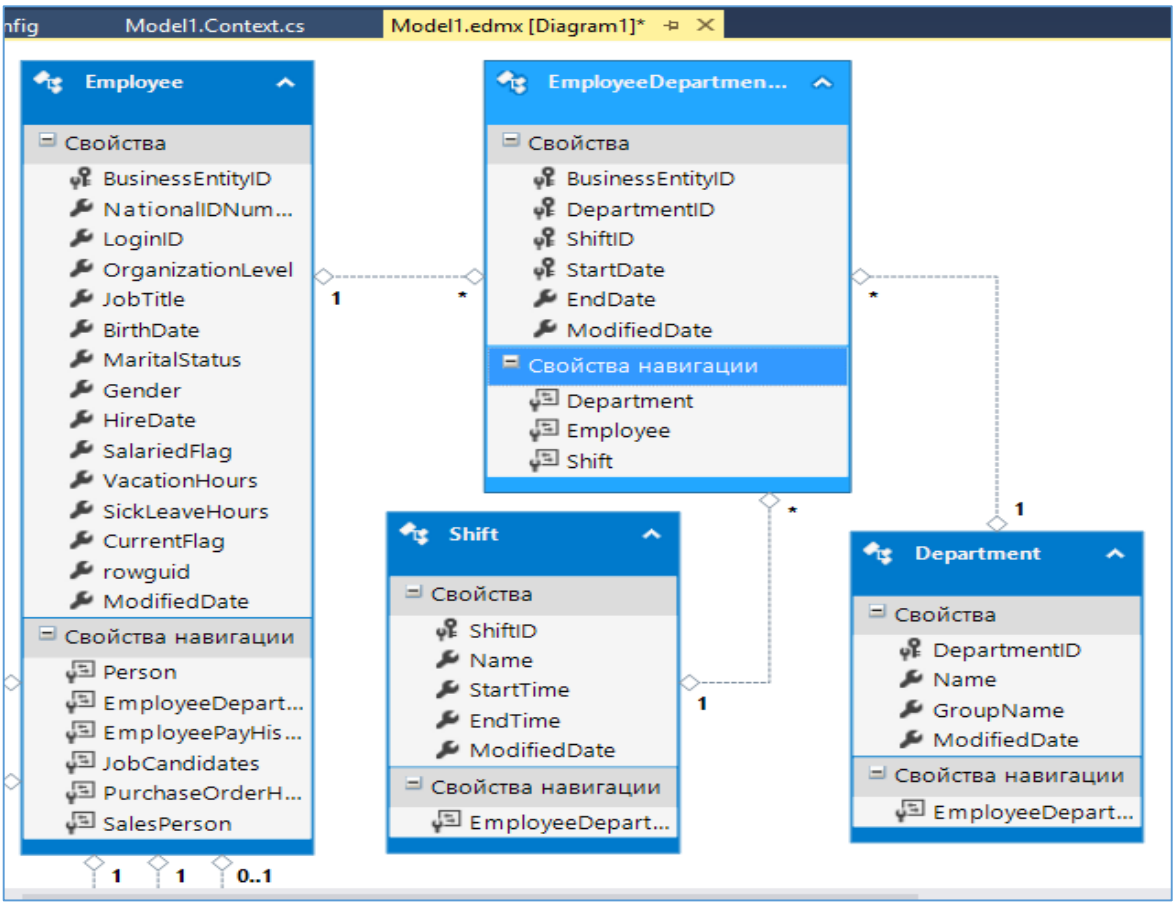

Рисунок 8 – Пример навигационных свойств модели

Например, свойство EmployeeDepartmentHistories в классе Department называется **навигационным свойством.**

```
public virtual ICollection<EmployeeDepartmentHistory>
EmployeeDepartmentHistories { get; set; }
```

Навигационное свойство обычно определяется как виртуальное свойство, используются для осуществления функциональности lazy loading (отложенная загрузка) Entity Framework.

Если навигационное свойство может содержать несколько объектов (экземпляров) (как в многие-ко-многим или один-ко-многим), то его тип должен быть определен списком, например, ICollection , где записи могут быть добавлены, удалены и обновлены.

3 Построение запросов

ADO.NET Entity Framework позволяет выполнять запросы к концептуальной модели:

- запросы и поиск сущностей.
- загрузка связанных сущностей
- неотслеживаемые запросы (No-Tracking Queries)
- построение sql запросов (raw sql queries)

Дополнительно ознакомится с материалом по ссылке:
http://professorweb.ru/my/entity-framework/6/level3/3_4.php

3.1 Запросы и поиск сущностей.

В этом разделе рассматриваются различные способы доступа к данным Entity Framework включающий в себя запросы LINQ и метод Find класса DbSet.

3.1.1 Поиск объектов/сущностей с помощью запросов. Класс DbSet и интерфейс IDbSet реализуют интерфейс IQueryable, поэтому могут быть использованы в запросах LINQ к базе данных. Пример:

```
using (var context = new BloggingContext())
{
   // Query for all blogs with names starting with B
   var blogs = from b in context.Blogs
               where b.Name.StartsWith("B")
               select b;

   // Query for the Blog named ADO.NET Blog
   var blog = context.Blogs
                .Where(b => b.Name == "ADO.NET Blog")                .FirstOrDefa
ult();
```

Когда результаты запрашиваются из базы данных, объекты, которые не существуют в контексте, присоединяются к контексту.

Если объект уже находится в контексте, существующий объект не возвращается (текущие и исходные значения свойства объекта не будут перезаписаны значениями из базы данных).

При выполнении запроса, объекты, которые были добавлены к контексту, но еще не были сохранены в базе данных. не будут возвращены как часть результирующего набора (прим. Существуют возможности просмотра данных из контекста).

Если запрос ничего не возвращает из базы данных, то результат будет пустой коллекцией, а не **null**.

3.1.2 Поиск объектов/сущностей с помощью первичных ключей

Метод Find класса DbSet обнаруживает сущность с указанными значениями первичного ключа. DbSet представляет коллекцию всех сущностей указанного типа, которые содержатся в контексте или могут быть запрошены из базы данных. Объекты DbSet создаются из DbContext с помощью метода DbContext.Set.

- Если сущность с указанными значениями первичного ключа содержится в контексте, она возвращается немедленно без выполнения запроса к хранилищу.

- В противном случае выполняется запрос к хранилищу в поисках сущности с указанными значениями первичного ключа. Если такая сущность обнаружена, она добавляется к контексту и возвращается вызывающей стороне.

- Если сущность не обнаружена в контексте или в хранилище, возвращается значение NULL.

Примечания. Порядок значений составного ключа соответствует порядку, определенному в модели EDM, который в свою очередь соответствует порядку, определенному в конструкторе с помощью быстрого API-интерфейса Code First или с помощью атрибута DataMember.

Пример метода Find класса DbSet<TEntity>

```
using (var context = new AdventureWorks2012_DataEntities()  )
          {
             var blog = context.Employees.Find(3);
             Console.WriteLine(blog.JobTitle.ToString());
             Console.Read();
          }
```

```
// Will return the same instance without hitting the database
   var blogAgain = context.Blogs.Find(3);

   context.Blogs.Add(new Blog { Id = -1 });

   // Will find the new blog even though it does not exist in the database
   var newBlog = context.Blogs.Find(-1);

 // Will find a User which has a string primary key
   var user = context.Users.Find("johndoe1987");
```

4.1.3 Поиск объектов с помощью составного первичного ключа

```
using (var context = new BloggingContext())
{ var settings = context.BlogSettings.Find(3, "johndoe1987"); }
```

3.2 Загрузка связанных сущностей

Entity Framework поддерживает три способа загрузки связанных данных:

- жадная загрузка (Eagerly Loading)
- отложенная загрузки (Lazy Loading)
- явная загрузки (Explicitly Loading)
- Использование запроса для подсчета связанных сущностей без необходимости загружать их

3.2.1 Жадная загрузка (Eagerly Loading). Жадная загрузка - это процесс, при котором посредством запроса для одного типа сущности также загружается связанные с ним сущности, как часть запроса. Жадная загрузка выполняется посредством метода Include. Например, запросы будут загружать блоги и все сообщения, связанные с каждым блогом.

```
using (var context = new BloggingContext())
{
   // Load all blogs and related posts
   var blogs1 = context.Blogs
                    .Include(b => b.Posts)
                    .ToList();

   // Load one blogs and its related posts
   var blog1 = context.Blogs
                   .Where(b => b.Name == "ADO.NET Blog")
                   .Include(b => b.Posts)
                   .FirstOrDefault();

   // Load all blogs and related posts
   // using a string to specify the relationship
   var blogs2 = context.Blogs
                    .Include("Posts")
                    .ToList();

   // Load one blog and its related posts
   // using a string to specify the relationship
   var blog2 = context.Blogs
                   .Where(b => b.Name == "ADO.NET Blog")
                   .Include("Posts")
                   .FirstOrDefault();
}
```

Многоуровневая жадная загрузка (Eagerly loading multiple levels). Возможно, также выполнить многоуровневую жадную загрузку связанных субъектов. Ниже приведены примеры запросов, использующие ссылочные навигационные свойства.

```
using (var context = new BloggingContext())
{
   // Load all blogs, all related posts, and all related comments
   var blogs1 = context.Blogs
                .Include(b => b.Posts.Select(p => p.Comments))
                .ToList();

   // Load all users their related profiles, and related avatar
   var users1 = context.Users
                .Include(u => u.Profile.Avatar)
                .ToList();

   // Load all blogs, all related posts, and all related comments
   // using a string to specify the relationships
   var blogs2 = context.Blogs
                .Include("Posts.Comments")
                .ToList();

   // Load all users their related profiles, and related avatar
   // using a string to specify the relationships
   var users2 = context.Users
                .Include("Profile.Avatar")
                .ToList();
}
```

3.2.2 Отложенная загрузка (Lazy Loading).

Во время загрузки этого типа связанные сущности загружаются из источника данных автоматически при доступе к свойству навигации. Используя загрузку этого типа, необходимо учитывать, что доступ к каждому свойству навигации приводит к выполнению отдельного запроса к источнику данных, если сущность еще не находится в **ObjectContext**.

Например, при использовании класса сущности Blog, определенный ниже, соответствующие экземпляры класса Posts будут загружены в первый раз, осуществляется навигационным свойством Posts:

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

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

Отключение отложенной загрузки для конкретных навигационных свойств (Turning off lazy loading for specific navigation properties).

Для коллекции Posts ленивая загрузка может быть отключена следующим образом (свойство Posts не виртуальное):

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

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

Загрузка коллекции *Posts* все еще может быть достигнуто с помощью жадной загрузки (см. жадную загрузку, Eagerly loading) или метод Load (см. Explicitly loading related entities).

Отключить отложенную загрузки для всех объектов (Turn off lazy loading for all entities)

Отложенная загрузка может быть отключена для всех объектов в контексте путем установки свойства в файле конфигурации. Например:

```
public class BloggingContext : DbContext
{
   public BloggingContext()
   {
      this.Configuration.LazyLoadingEnabled = false;
   }
}
```

Загрузка связанных сущностей все еще может быть доступны с помощью жадной загрузки (см. жадная загрузка связанных сущностей) или метода Load (см явная загрузки связанных сущностей ниже).

3.2.3 Явная загрузка (Explicitly Loading)

3.2.3.1 Последним вариантом загрузки данных в Entity Framework является **явная загрузка (explicit loading)** данных. Явная загрузка, как и отложенная загрузка, не приводит к загрузке всех связанных данных в первом запросе. Но при этом, в отличие от отложенной загрузки, при вызове навигационного свойства связанного класса, эта загрузка не приводит к автоматическому извлечению связанных данных, вы должны явно вызвать метод Load(), если хотите загрузить связанные данные. Такой тип загрузки может использоваться в следующих случаях:

- Этот тип загрузки устраняет необходимость отмечать навигационные свойства класса модели как виртуальные. Для вас это может показаться не значительным изменением, но тот факт, что технология доступа к связанным данным при отложенной загрузке требует изменять ваши классы POCO модели, далека от идеальной.
- Вы можете работать с существующей библиотекой классов, где навигационные свойства не помечены как виртуальные, и вы не можете изменить эту библиотеку.
- В конце концов явная загрузка позволяет быть уверенным, что вы точно знаете, когда запросы отправляются в базу данных. Отложенная загрузка имеет характерную особенность генерировать много запросов к базе данных, с явной загрузкой очевидно, когда и где запросы выполняются в настоящее время.

Явная загрузка использует *метод DbSet.Entry()* для доступа к сущностному объекту, а не свойство типа DbSet. Объект**DbEntityEntry**, возвращаемый этим методом, дает вам доступ ко всей информации о сущностном типе данных. Помимо обычных свойств модели, этот объект хранит большое количество расширенных настроек сущностных объектов, а также позволяет вызвать *метод Load()* для вызова явной загрузки. В примере ниже мы используем метод ExplicitLoading(), в котором мы реализовали самый первый пример в этой статье, в котором мы не могли загрузить данные заказов автоматически, т.к. навигационное свойство Customer.Orders не было виртуальным:

Пример №1

```
var context = new AdventureWorks2012_DataEntities();

// Вывести заказ по немору
SalesOrderHeader salesorderheader = context.SalesOrderHeaders.Where(c =>
c.SalesOrderID == 43659).FirstOrDefault();

// Загрузить клиентов с помощью явной загрузки
// Customer - свойство класса SalesOrderHeader
   context.Entry(salesorderheader).Reference(c => c.Customer).Load();

 if (salesorderheader != null && salesorderheader.Customer != null)
 {   Console.Write(salesorderheader.SalesOrderID.ToString());
     Console.Write("   " + salesorderheader.CustomerID.ToString());
     Console.WriteLine("   " + salesorderheader.Customer.ModifiedDate);
 }
. . .
```

Пример №2

```
var context = new AdventureWorks2012_DataEntities();
 // Загрузить запись по одному покупателю
 Customer customer = context.Customers.Where(c => c.CustomerID ==
```

```
29825).FirstOrDefault();

 // Загрузить связанные  заказы с помощью явной загрузки
// SalesOrderHeaders - свойство класса Customer
 context.Entry(customer).Collection(c => c.SalesOrderHeaders).Load();

if (customer != null && customer.SalesOrderHeaders != null)
      {
            foreach (var order in customer.SalesOrderHeaders)
            {
             Console.Write(customer.CustomerID.ToString());
             Console.WriteLine("   " + order.SalesOrderID.ToString());
            }
         }
. . .
```

3.2.3.2 Применение фильтров при явной загрузки связанных объектов (Applying filters when explicitly loading related entities).

Query метод обеспечивает построение запросов, используемые Entity Framework при загрузке связанных сущностей.

В запросах используются фильтры LINQ и методы LINQ, такие как ToList, Load, др.

Пример:

```
using (var context = new AdventureWorks2012_DataEntities())
        {
         // Загрузить запись по одному покупателю
          Customer customer = context.Customers.Where(c => c.CustomerID ==
29825).FirstOrDefault();

          // Загрузить заказы с помощью явной загрузки
          // bcgjkmpez свойство класса Customer SalesOrderHeaders
          //применение фильтров перед вызовом метода LINQ
      List<SalesOrderHeader>  spisok =    context.Entry(customer).Collection(c =>
c.SalesOrderHeaders).Query().Where(p => p.SubTotal > 30000).ToList() ;

       if (customer != null && spisok != null)
          {
             foreach (var order in spisok)
             {
               Console.WriteLine("{0}  {1}  {2:F2}" ,
order.SalesOrderID.ToString(), order.CustomerID,   order.SubTotal );
             }

          }
```

```
    }
        Console.ReadLine();
```

3.2.4 Использование запроса для подсчета связанных сущностей без необходимости загружать их. Пример:

```
using (var context = new AdventureWorks2012_DataEntities())
        {
            // Загрузить запись по одному покупателю
            Customer customer = context.Customers.Where(c => c.CustomerID ==
29825).FirstOrDefault();

            // Загрузить связанные  заказы с помощью явной загрузки
            // SalesOrderHeaders - свойство класса Customer
            var postCount = context.Entry(customer).Collection(c =>
c.SalesOrderHeaders).Query().Count();
            Console.WriteLine("{0}  ", postCount.ToString());
        }
        Console.ReadLine();
```

3.2.5 Entity Framework No-Tracking Queries

Возвращает новый запрос, возвращающий сущности, которые не будут кэшироваться в контексте DbContext илиObjectContext. Этот метод работает, вызывая метод AsNoTracking базового объекта запроса.

```
using (var context = new BloggingContext())
{
   // Query for all blogs without tracking them
   var blogs1 = context.Blogs.AsNoTracking();

   // Query for some blogs without tracking them
   var blogs2 = context.Blogs
                   .Where(b => b.Name.Contains(".NET"))
                   .AsNoTracking()
                   .ToList();
}
```

4 Задание для выполнения лабораторной работы

1) Подключить файлы базы данных к текущему вашему проекту (см. пункт 1-2)
2) Реализовать запросы и поиск сущностей, вывода представлений (см 3.1).
3) Реализовать способы загрузки связанных сущностей Entity Framework и вывода представлений связанных данных (см 3.2)..

Источники
https://msdn.microsoft.com/ru-ru/library/ms233817.aspx
http://professorweb.ru/my/entity-framework/6/level3/3_4.php

Источник ВД
http://msftdbprodsamples.codeplex.com/releases/view/93587