ГЛАВА 19

Введение в LINQ to Entities

Листинг 19.1. Простой пример обновления контактного имени заказчика в базе данных Northwind

```
// Создать ObjectContext.
NorthwindEntities context = new NorthwindEntities();

// Извлечь заказчика LAZYK
Customer cust = (from c in context.Customers
                 where c.CustomerID == "LAZYK"
                 select c).Single<Customer>();

// Изменить контактное имя.
cust.ContactName = "Ned Plimpton";

// Сохранить изменения.
try {
   context.SaveChanges();
} catch (OptimisticConcurrencyException) {
   context.Refresh(RefreshMode.ClientWins,
      context.Customers);
   context.SaveChanges();
}
```

На заметку! Этот пример требует генерации сущностной модели данных, о которой речь пойдет далее в этой главе.

В листинге 19.1 с помощью LINQ to Entities был произведен запрос записи, значение поля `CustomerID` которой равно `"LAZYK"`, из таблицы `Customers` базы данных Northwind и возврат объекта `Customer`, представляющего эту запись. Затем было обновлено свойство `ContactName` объекта `Customer` с сохранением изменения в базе данных, для чего был вызван метод `SaveChanges`. Нажмите <Ctrl+F5> для запуска программы из листинга 19.1. Консольного вывода нет, но если вы заглянете в базу данных, то увидите там, что поле `ContactName` для заказчика `"LAZYK"` теперь содержит значение `"Ned Plimpton"`.

На заметку! Код в этом примере вносит изменение в базу данных, не восстанавливая старого значения. Исходным значением `ContactName` заказчика `LAZYK` было `"John Steel"`. Его понадобится восстановить, чтобы последующие примеры работали правильно. Это можно сделать вручную или же соответствующим образом изменить исходный код и запустить пример снова.

В этой книге используется расширенная версия базы данных Northwind. Подробности ищите в разделе "Получение соответствующей версии базы данных Northwind" далее в главе.

Введение

В главе 12 объяснялось, что LINQ to SQL — это система объектно-реляционного отображения начального уровня. LINQ to Entities — это часть платформы ADO.NET Entity Framework, предоставляющая более высокую гибкость и больше средств, чем LINQ to SQL, но следующая за LINQ to SQL в отношении адаптации, из-за повышенной сложности и ранних выпусков, которым пока недостает ключевых средств.

Код в листинге 19.1 делает то же самое, что и код в листинге 12.1, который использовался для представления LINQ to SQL. Не поленитесь сравнить листинги 19.1 и 12.1; вы увидите, что они довольно похожи.

API-интерфейс Entity Framework спроектирован для работы с любыми базами данных, поддерживающими ADO (а не только с SQL Server), и даже включает собственный диалект независимого от поставщика языка SQL, который можно применять в качестве альтернативы LINQ. Фактически Entity Framework обладает настолько широким набором средств, что для их описания понадобилась бы отдельная книга. Здесь будет показано, как запустить и использовать только важнейшие части Entity Framework, относящиеся к LINQ to Entities.

Имеется некоторая путаница с терминами. В конце концов, разве мы не потратили последние несколько глав на обсуждение сущностных классов как части LINQ to SQL? Да, действительно. API-интерфейсы LINQ to SQL и Entity Framework делают нечто схожее, а потому не удивительно, что в них используется общая терминология.

Подобно LINQ to SQL, LINQ to Entities позволяет работать с объектами, которые представляют информацию из базы данных — выполнять LINQ-запросы, изменять значения, добавлять и удалять объекты. И так же, как LINQ to SQL, первый шаг в направлении использования этих средств предусматривает генерацию классов, отображающих содержимое базы данных на объекты — то, что делается при создании сущностной модели данных (entity data model — EDM). Модель EDM состоит из набора объектов и свойств, которые используются для взаимодействия с данными.

В листинге 19.1 сначала создается экземпляр класса `NorthwindEntities`. Этот класс унаследован от класса `System.Data.Objects.ObjectContext`, который подробно рассматривается в последующих главах. Это точка входа в EDM, похожая на класс `DataContext` для LINQ to SQL. Класс `NorthwindEntities` устанавливает соединение с базой данных при создании его нового экземпляра и берет на себя ответственность за сохранение изменений при вызове метода `SaveChanges`.

Затем из базы данных извлекается одиночный заказчик, который помещается в объект `Customer`. Объект `Customer` — это экземпляр класса `Customer`, являющегося частью сущностной модели данных. Далее в этой главе будет показано, как генерировать EDM для базы данных Northwind. После извлечения `Customer` обновляется одно из его свойств и вызывается метод `SaveChanges` для сохранения изменений в базе данных. Вызов метода `SaveChanges` помещен в блок `try/catch`, чтобы можно было разрешить любые потенциальные конфликты, связанные с параллелизмом. Более подробно о разрешении этих конфликтов будет рассказываться в главе 20.

Прежде чем получится запустить этот пример, равно как и любой другой пример в этой главе, понадобится создать сущностную модель данных для базы Northwind. Читайте подробности в разделе "Предварительные условия для запуска примеров" далее в главе.

Как это делалось с LINQ to SQL, начнем с обзора ключевых частей LINQ to Entities. Кое-что из того, что будет рассказано о LINQ to Entities, излагается в форме сравнения с LINQ to SQL, так что если вы не читали этих глав, сделайте это перед тем, как двигаться дальше. В первом примере, приведенном в начале этой главы, использовался класс-наследник `ObjectContext` по имени `NorthwindEntities`, сущностный класс `Customer`, средства обнаружения и разрешения конфликтов, а также обновление базы данных через метод `SaveChanges`. Для начала рассмотрим некоторые основы этих компонентов, чтобы обеспечить базовое понимание основ LINQ to Entities и ADO.NET Entity Framework в целом.

ObjectContext

Класс `ObjectContext` — это ключ для доступа к сущностной модели данных и эквивалент класса `DataContext` из LINQ to SQL. Класс `ObjectContext` отвечает за создание и управление соединением с базой данных, отслеживает изменения и управляет постоянством. Подробности будут изложены позднее, а пока достаточно знать, что именно класс `ObjectContext` соединяет с базой данных, когда создается новый экземпляр `NorthwindEntities`, и этот же класс отслеживает изменения, которые вносятся в объект `Customer`, а также транслирует их в оператор SQL, сохраняющий изменения по вызову метода `SaveChanges`.

Обычно используется класс, унаследованный от `ObjectContext`, который создается при генерации EDM из базы данных. Далее в этой главе будет показано, как это делается для базы данных Northwind. Имя класса выбирается на основе имени базы данных — в форме `[База_данных]Entities`. В листинге 19.1 для базы данных Northwind предусмотрен класс `NorthwindEntities`.

Производный класс `[База_данных]Entities` для каждой таблицы базы, выбранной при создании EDM, будет иметь свойство `ObjectSet<T>`, представляющее таблицу, причем `T` здесь — это тип сущностного класса, созданного для представления записи в таблице. Например, класс `NorthwindEntities`, использованный в листинге 19.1, имеет общедоступное свойство `Customers` типа `OrderSet<Customer>`. Оно применялось для выполнения запроса LINQ к множеству заказчиков.

Сущностные классы

Сущностные классы в Entity Framework имеют много общего с классами, описанными в главе о LINQ to SQL. Это типы .NET, которые представляют собой отображение на реляционную структуру базы данных. Платформа Entity Framework позволяет выполнять очень сложные отображения между сущностными классами и реляционными данными, которые могут охватывать разные базы данных и быть абстрагированы различными интересными способами. Здесь все эти тонкости не рассматриваются, потому что внимание сосредоточено на аспектах LINQ, но если нужны развитые средства ORM, то определенно следует обратиться к Entity Framework.

Сущностные классы в примерах обнаруживаются по наличию классов и объектов, имеющих имена таблиц базы данных в форме существительного единственного числа. Например, в листинге 19.1 используется класс по имени `Customer`. Поскольку `Customer` — форма единственного числа от `Customers`, и в базе данных Northwind есть таблица `Customers`, это намек на то, что `Customer` является сущностным классом, который представляет записи из таблицы `Customer` базы данных Northwind.

Мастер построения сущностной модели, с которым вы вскоре ознакомитесь, имеет опцию поддержки имен таблиц во множественном числе при создании сущностных классов, поэтому, когда он находит таблицу базы под названием `Customers`, то создает сущностный класс по имени `Customer` для представления элемента таблицы. Тот же

подход обеспечивается опцией /pluralize утилиты SQLMetal, которая была показана в главе 12, и это обеспечивает значительное повышение читабельности кода.

Ассоциации

Ассоциация — термин, применяемый для обозначения отношения первичного ключа к внешнему ключу между двумя сущностными классами. При отношении "один ко многим" результатом ассоциации является то, что родительский класс, включающий в себя первичный ключ, содержит коллекцию дочерних классов, имеющих внешний ключ.

Коллекция сохраняется в EntityCollection<T>, где T — тип дочернего сущностного класса. В отношении "многие ко многим" каждый сущностный класс поддерживает EntityCollection<T>, где T — тип противоположной сущности в отношении.

Коллекции сущностей доступны через общедоступные свойства по имени внешнего ключа. Поэтому, например, чтобы добраться к заказам, ассоциированным с заказчиком в базе данных Northwind, следует обратиться к свойству Customer.Orders, которое вернет EntityCollection<Order>.

Преимущество ассоциаций между сущностными типами заключается в том, что они позволяют прозрачно выполнять навигацию по данным, не принимая во внимание того факта, что они могут быть распределены среди множества таблиц или даже множества баз данных.

Предварительные условия для запуска примеров

В этой и последующей главах, посвященных LINQ to Entities, используется одна и та же расширенная база данных Northwind, которую применялась также и в главах по LINQ to SQL. Для базы Northwind понадобится сгенерировать сущностную модель данных.

Получение соответствующей версии базы данных Northwind

Для согласованности будет использоваться та же самая расширенная версия базы данных примеров Microsoft под названием Northwind, которая применялась в главах, посвященных LINQ to SQL. Она входит в состав загружаемых примеров для книги.

Генерация сущностной модели данных Northwind

Сгенерировать модель EDM можно либо с помощью инструмента командной строки EdmGen, либо в среде Visual Studio 2010. Далее будет показано, как это делать с использованием графического мастера Visual Studio. Сначала щелкните правой кнопкой мыши на проекте, выберите в контекстном меню пункт Add⇨New Irtem (Добавить⇨Новый элемент) и затем укажите в списке вариант ADO.NET Entity Data Model (Сущностная модель данных ADO.NET). Измените имя модели данных. Поскольку используется база данных Northwind, в качестве имени имеет смысл указать NorthwindDataModel.edmx. Щелкните на кнопке Add (Добавить), после чего запустится мастер создания сущностной модели данных (Entity Data Model Wizard), окно которого показано на рис. 19.1.

Сущностную модель данных можно создать с нуля или же сгенерировать ее на основе имеющейся базы данных. Здесь необходимо сгенерировать EDM из базы данных Northwind, поэтому выберите значок Generate from database (Генерировать из базы данных) и щелкните на кнопке Next (Далее) для перехода на экран подключения к данным, который показан на рис. 19.2.

Этот экран используется для выбора базы данных, на основе которой будет сгенерирована EDM-модель. На рис. 19.2 выбрана существующая база данных Northwind, которая ранее была подключена к SQL Server 2008.

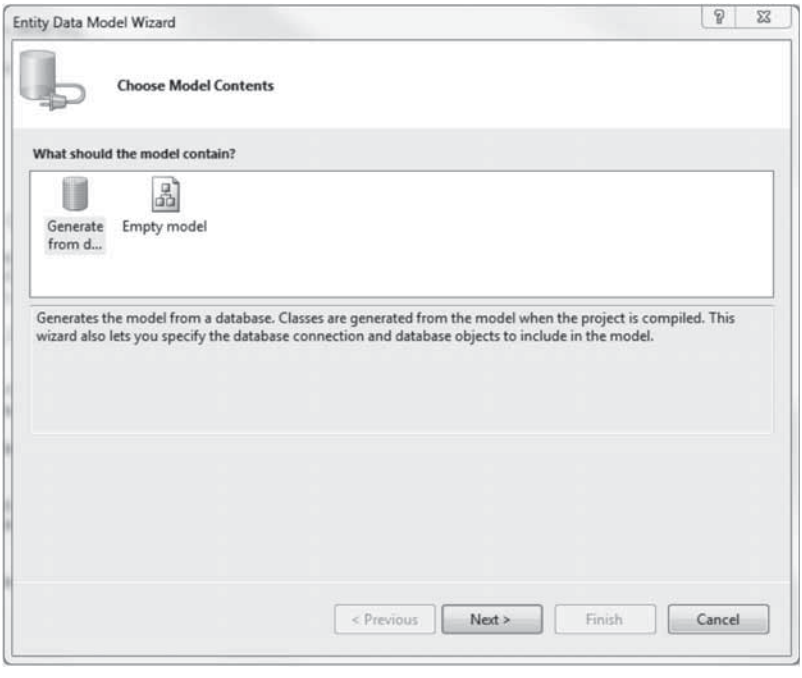

Рис. 19.1. Первый экран мастера Entity Data Model Wizard

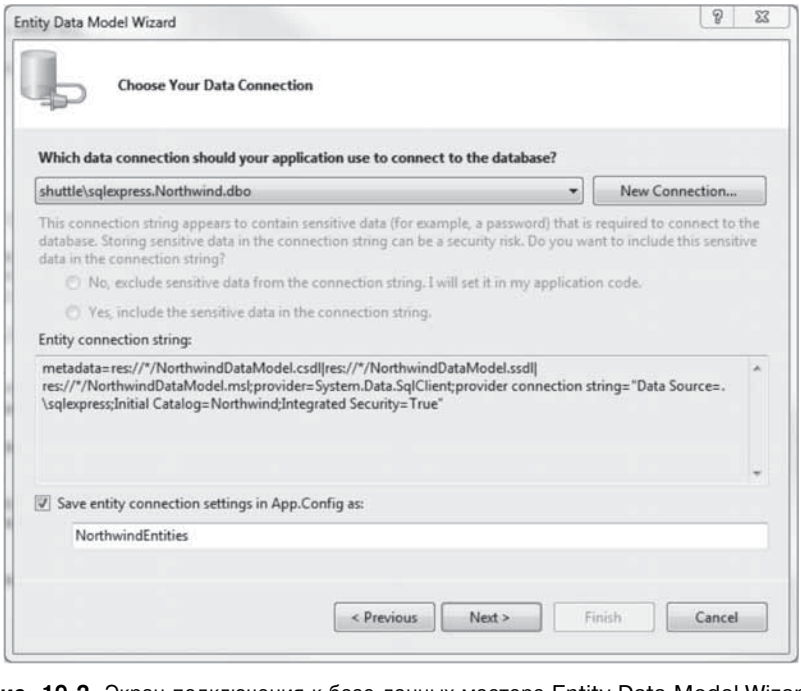

Рис. 19.2. Экран подключения к базе данных мастера Entity Data Model Wizard

Содержимое окна может отличаться в зависимости от местоположения базы данных. По крайней мере, имя сервера будет отличаться от `shuttle`. Выберите нужное соединение и щелкните на кнопке Next для перехода к следующему экрану мастера, показанному на рис. 19.3.

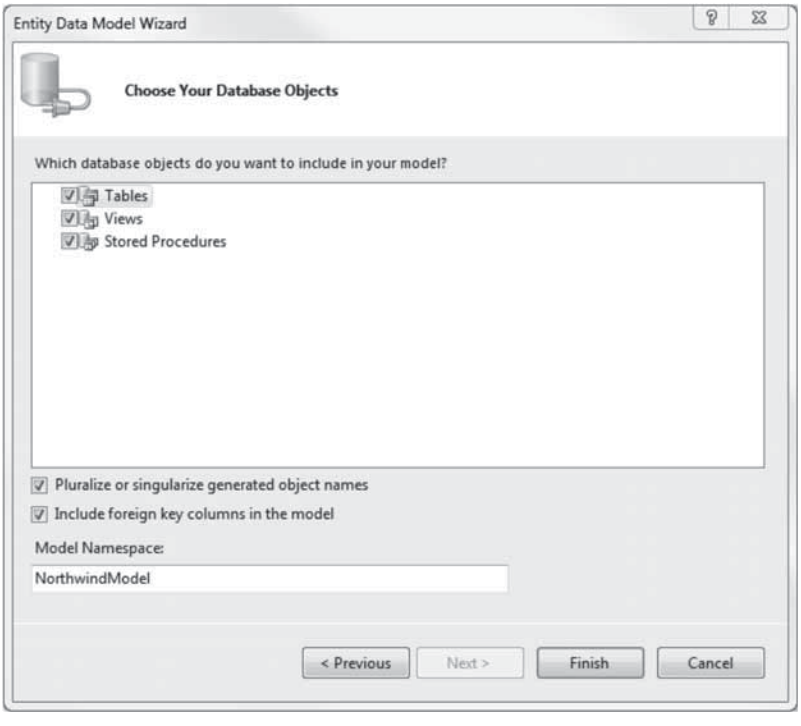

Рис. 19.3. Экран объектов сущностной модели базы данных мастера Entity Data Model Wizard

На этом экране мастера выбираются таблицы, представления и хранимые процедуры из базы данных, которые будут включены в EDM-модель. Можно также выбрать множественную или единственную форму имен объектов (например, чтобы объекты, сгенерированные из таблицы `Customers` назывались `Customer`) и добавить внешние ключи. Для текущих целей понадобится включить в модель все содержимое базы данных, поэтому отметьте все флажки на экране, показанном на рис. 19.3. Щелкните на кнопке Finish (Готово) для закрытия окна мастера и генерации модели. Когда процесс завершится, среда Visual Studio должна выглядеть примерно так, как показано на рис. 19.4.

В основной части окна показана создаваемая сущностная модель. Здесь можно видеть свойства каждого сущностного класса и отношения между ними. Кроме того, также выводится множество предупреждений относительно модели данных; они вызваны тем, что с расширенной базой данных Northwind связан ряд недостатков. Пока эти ошибки игнорируются, но в реальном проекте они должны быть тщательно изучены. Наконец, обратите внимание, что мастер создания EDM добавил в проект ряд новых ссылок; они требуются для API-интерфейса Entity Framework и не должны удаляться. На этом все — сущностная модель данных для расширенной базы данных Northwind сгенерирована. В следующем разделе будет приведен очень краткий обзор ее использования.

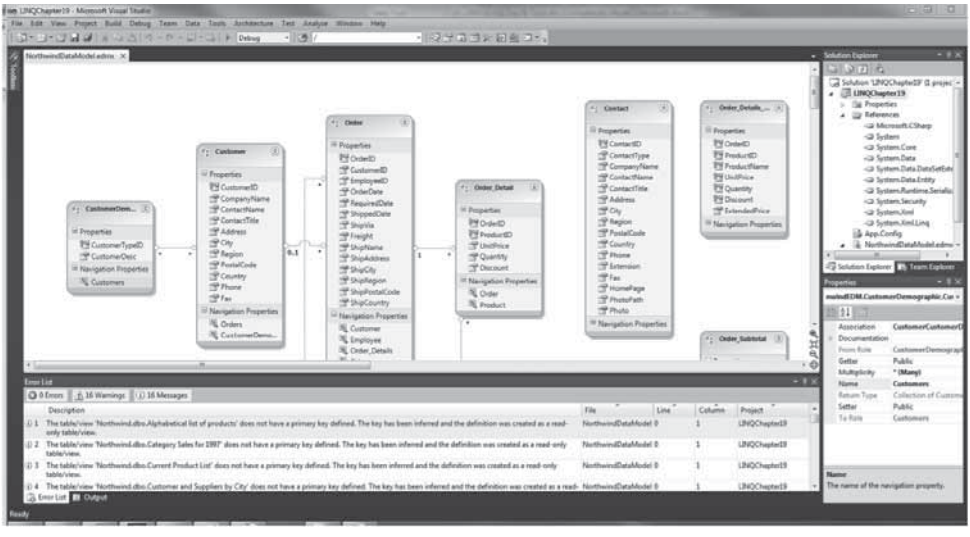

Рис. 19.4. EDM-модель Northwind в Visual Studio

Использование API-интерфейса LINQ to Entities

Сборки, которые понадобятся для использования LINQ to Entities, добавляются к проекту автоматически при генерации сущностной модели данных. В отличие от LINQ to SQL, импортировать пространство имен для использования сущностных классов не понадобится, т.к. мастер Entity Data Model генерирует сущностные модели данных в пространстве имен по умолчанию проекта.

IQueryable<T>

Во многих примерах настоящей и последующих глав, посвященных LINQ to Entities, работа будет проводиться с последовательностями типа `IQueryable<T>`, где `T` — тип сущностного класса. Этот тип последовательностей обычно возвращается запросами LINQ to Entities, как и в LINQ to SQL. Обычно работа с ними выглядит, как с последовательностью `IEnumerable<T>`, и это не случайно. Интерфейс `IQueryable<T>` реализует интерфейс `IEnumerable<T>`. Ниже приведено определение `IQueryable<T>`:

```
interface IQueryable<T> : IEnumerable<T>, IQueryable
```

Благодаря этому наследованию последовательность `IQueryable<T>` можно трактовать как последовательность `IQueryable<T>`.

Некоторые общие методы

В процессе демонстрации средств LINQ to Entities нужна возможность запрашивать или модифицировать базу данных, внешнюю по отношению к Entity Framework. Чтобы выделить код LINQ to Entities и исключить как можно больше тривиальных деталей (в то же время обеспечивая полезные примеры), были разработаны некоторые общие методы. Добавьте их к своему исходному коду при тестировании примеров.

GetStringFromDo()

Общий метод, который весьма пригодится — это метод для получения простой строки из базы данных с помощью стандартного механизма ADO.NET (листинг 19.2). Это позволит посмотреть, что на самом деле есть в базе данных, и сравнить с тем, что отображает LINQ to Entities.

Листинг 19.2. GetStringFromDb: метод для извлечения строки посредством ADO.NET

```
static private string GetStringFromDb(string sqlQuery) {

   string connection =
     @"Data Source=.\SQLEXPRESS;Initial Catalog=Northwind;Integrated Security=SSPI;";

   System.Data.SqlClient.SqlConnection sqlConn =
     new System.Data.SqlClient.SqlConnection(connection);

   if (sqlConn.State != ConnectionState.Open) {
      sqlConn.Open();
   }

   System.Data.SqlClient.SqlCommand sqlCommand =
     new System.Data.SqlClient.SqlCommand(sqlQuery, sqlConn);

   System.Data.SqlClient.SqlDataReader sqlDataReader = sqlCommand.ExecuteReader();
   string result = null;

   try {
      if (!sqlDataReader.Read()) {
        throw (new Exception(
          String.Format("Unexpected exception executing query [{0}].", sqlQuery)));
      } else {
        if (!sqlDataReader.IsDBNull(0)) {
          result = sqlDataReader.GetString(0);
        }
      }
   } finally {
      // Всегда вызывать Close, когда чтение завершено.
      sqlDataReader.Close();
      sqlConn.Close();
   }

   return (result);
}
```

При вызове методу `GetStringFromDb` передается строка запроса SQL. Метод создает и открывает новое соединение с базой данных.

Затем создается объект `SqlCommand`, конструктору которого передается запрос и соединение. Затем `SqlDataReader` получается вызовом метода `ExecuteReader` на `SqlCommand`. Объект `SqlDataReader` читает данные с помощью своего метода `Read`, и если данные были прочитаны, а значение возвращенного первого столбца не `null`, то это значение извлекается методом `GetString`. Наконец, `SqlDataReader` и `SqlConnection` закрываются, и значение первого столбца возвращается вызывающему методу.

ExecuteStatementInDb()

Иногда для изменения состояния базы данных вне Entity Framework возникает необходимость в выполнении оператора SQL, отличного от запроса вроде `insert`, `update` и `delete` в ADO.NET. Для этой цели был создан метод `ExecuteStatementInDb`, приведенный в листинге 19.3.

Листинг 19.3. `ExecuteStatementInDb`: метод для выполнения вставок, обновлений и удалений в ADO.NET

```
static private void ExecuteStatementInDb(string cmd) {
   string connection =
     @"Data Source=.\SQLEXPRESS;Initial Catalog=Northwind;Integrated Security=SSPI;";

   System.Data.SqlClient.SqlConnection sqlConn =
     new System.Data.SqlClient.SqlConnection(connection);
   if (sqlConn.State != ConnectionState.Open) {
      sqlConn.Open();
   }

   System.Data.SqlClient.SqlCommand sqlComm =
     new System.Data.SqlClient.SqlCommand(cmd);

   sqlComm.Connection = sqlConn;
   try {
     Console.WriteLine("Выполнение оператора SQL в базе данных с помощью ADO.NET...");
       sqlComm.ExecuteNonQuery();
       Console.WriteLine("База данных обновлена.");
   } finally {
       // Закрыть соединение.
       sqlComm.Connection.Close();
   }
}
```

При вызове методу `ExecuteStatementInDb` передается аргумент `string`, содержащий команду SQL. Создается экземпляр `SqlConnection`, за которым следует `SqlCommand`. Объект `SqlConnection` затем открывается и команда SQL выполняется посредством вызова метода `ExecuteNonQuery` объекта `SqlCommand`. Наконец, `SqlCommand` закрывается.

Резюме

В этой главе было дано введение в платформу Entity Framework и API-интерфейс LINQ to Entities. Также были рассмотрены некоторые его базовые элементы, такие как объекты `ObjectContext`, сущностные классы и ассоциации.

В главе было продемонстрировано, как генерируется сущностная модель данных для расширенной базы данных Northwind, содержащая сущностные классы, которые используются для работы с данными Northwind. Эти сущностные классы применяются во всех примерах LINQ to Entities. Также были представлены методы общего назначения, на которые полагаются некоторые примеры в последующих главах, посвященных LINQ to Entities. Следующая глава посвящена демонстрации использования LINQ to Entities и Entity Framework для выполнения общих операций с базами данных.