Урок 1: Введение в ADO.NET

Содержание

1. Что такое ADO.NET

Вы уже познакомились с базами данных и оценили по достоинству удобство хранения в них данных. Но согласитесь – пользователю не очень удобно работать напрямую с базой данных. Кроме того, такой способ ра-

боты не отвечает требованиям безопасности. И не важно, с чего началась разработка приложения (приложение разрабатывается как обертка существующей базы данных, или новая база данных выступает хранилищем информации вашего приложения), первое, с чем сталкивается разработчик, – это необходимость построить связь между приложением и базой данных.

Для того чтобы осуществлять связь с базами данных **в NET. Framework**, разработана технология **ADO.NET**, которую мы и рассмотрим в данном уроке.

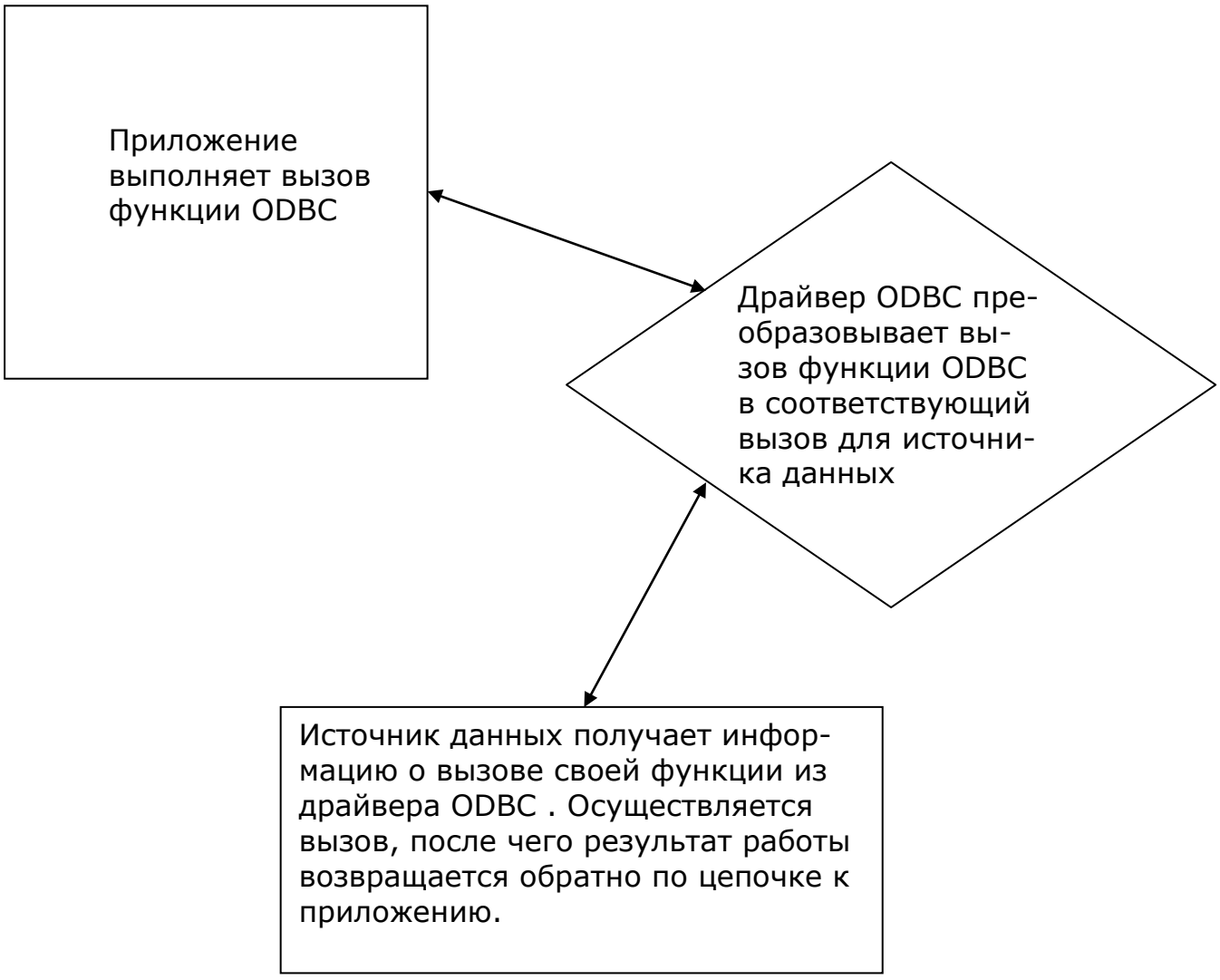
Но прежде, чем приступить к детальному рассмотрению технологии **ADO.NET**, узнаем, как развивались технологии доступа к данным до появления **ADO.NET** и в чем отличия существующих технологий.

2. Исторический экскурс в технологии доступа к данным

Существует большой выбор баз данных, которыми можно воспользоваться для реализации потребностей вашего приложения. Изначально, каждый производитель баз данных предлагал свой набор функций для работы с ней, причем никакого стандарта этих функций не было. Из-за того, что для связи с каждой базой данных использовались разные функции, каждый разработчик тратил большое количество времени на знакомство и погружение в тонкости **API** конкретной базы данных. Естественно такая ситуация не могла устраивать программистов так как фактически приходилось тратить большие сегменты времени на получение однотипных знаний. В силу этого началась разработка некоторой унифицированной технологии для доступа к различным базам данных. Авангард движения возглавила компания **Microsoft** (ныне малоизвестная в широких кругах ☺). К ней примкнули другие компании.

В результате активной работы на суд публики была представлена технология **ODBC**. **ODBC** (Open DataBase Connectivity) – программный интерфейс (**API**), который позволяет единообразно обращаться к различным источникам данных не задумываясь о тонкостях и особенностях данного

источника. Что это означает на практике? Это значит, что у вас есть единый набор функций, которые используются для доступа к любым источникам данных. Технология **ODBC** предоставила прослойку, которая скрывает все сложности. Важным понятием технологии **ODBC** является драйвер. Драйвер – набор библиотек для доступа к тому или иному источнику данных **ODBC**. Например, если вам нужен, доступен через **ODBC** к СУБД **MS SQL Server** следует проинсталлировать соответствующий драйвер. Фактически драйвер и является прослойкой, которая скрывает аспекты работы того или иного источника данных. Продемонстрируем механизм использования драйвера на схеме:



Следующей технологией доступа к данным, появившейся на свет, стала технология **DAO**.

DAO (Data Access Objects) - основана на технологии баз данных **Microsoft Jet**. **Microsoft Jet** - процессор баз данных, предназначенный для **Microsoft Access**. Данный процессор является первым объектно-ориентированным интерфейсом для связи с **Microsoft Access** (в отличие от технологии **ODBC**, где доступ реализован в процедурном стиле).

Программы, взаимодействующие с **Access**, могут использовать **DAO** для прямого доступа к данным. Использование **DAO** для работы с **Access** является самым правильным и эффективным решением, так как это основная цель **DAO**. При этом нужно отметить, что **DAO** также может обращаться и к другим СУБД, например таким как

MS SQL Server, **Firebird**, **MySQL** и т.д. При возникновении такой ситуации **DAO** будет использовать **ODBC**, то есть будет производиться преобразование вызовов **DAO** в соответствующие вызовы **ODBC**, что естественно замедлит работу приложения. Важно отметить, что сейчас проблемы связанные с замедлением доступа к СУБД отличным от **MS Jet** снижены до минимума.

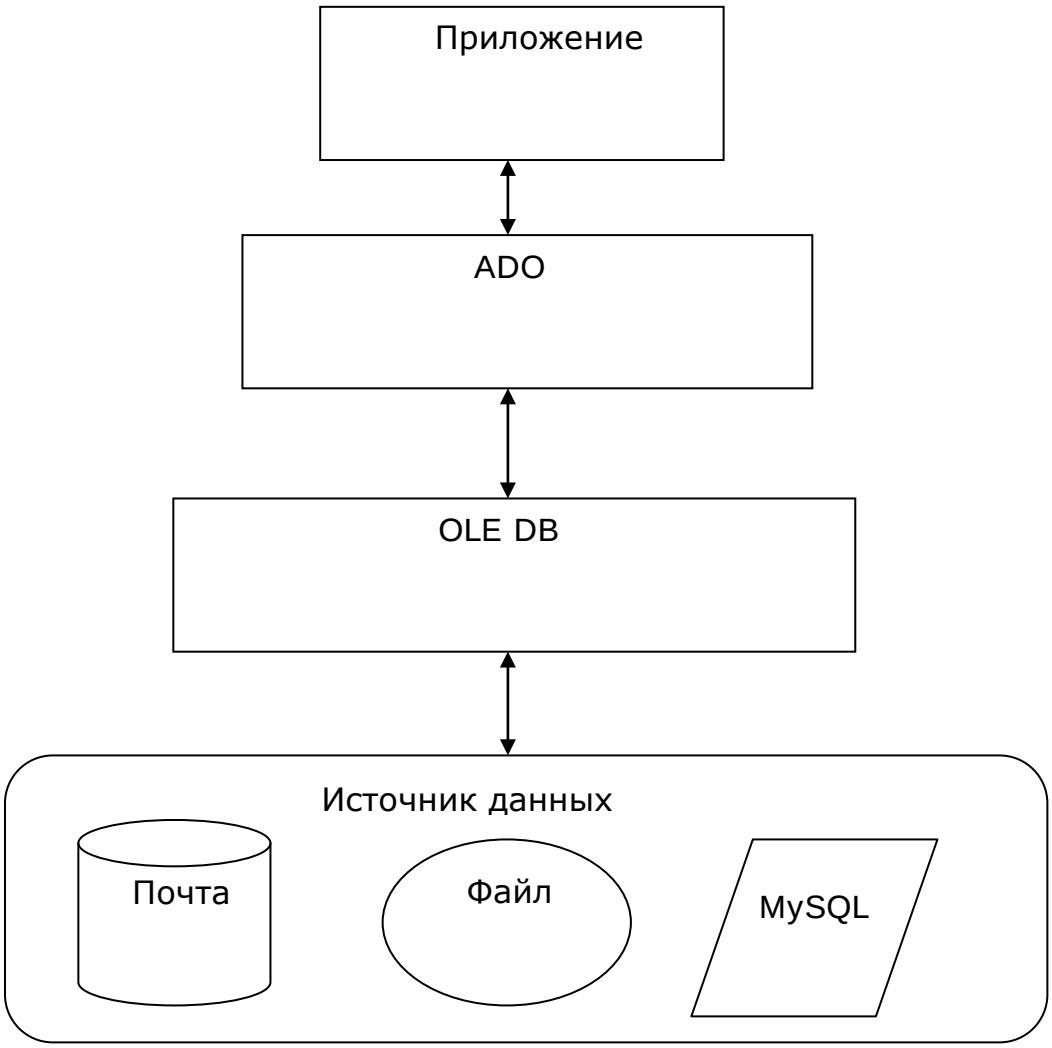
Технологии **ODBC** и **DAO** предназначены для взаимодействия с реляционными источниками данных. А что делать с другими источниками? Например, такими как почта, файловое хранилище и так далее? Для решения этой задачи была разработана технология **OLE DB**.

OLE DB (Object Linking and Embedding Database) – набор **COM** интерфейсов (когда мы говорим о **COM**, имеется в виду **COM** технология), которые позволяют приложениям обращаться к различным источникам данных (как реляционным, так и другим). Схема взаимодействия **OLE DB** и приложения схожа с **ODBC**. Однако вместо понятия драйвера используется понятие провайдер. Провайдер – набор объектов, реализующих множество **COM** интерфейсов для доступа к источнику данных. Например, провайдер для доступа к файловому хранилищу, провайдер для доступа к локальной папке, провайдер для взаимодействия с **MySQL**.

При обращении к реляционным источникам, соответствующий провайдер будет использовать **ODBC API**.

OLE DB была разработана для использования в рамках **C** и **C++**, а также для языков с **C**-подобными вызовами функций. Например, программисты **Visual Basic** были отрезаны от **OLE DB**. Решением этой проблемы занимается технология **ADO**.

Технология **ADO** (ActiveX Data Objects) предоставляет набор объектов для высокоуровневого доступа к источникам данных вне зависимости от их типа. Технологию **ADO** могут использовать как программисты **VC++**, так и программисты **VB** и **JScript**, и так далее. Фактически **ADO** является надстройкой над **OLE DB**, упрощающей особенности использования **OLE DB**. Естественно из-за этого есть некоторые потери в скорости взаимодействия по сравнению с чистым **OLE DB**.



Все вышеперечисленные технологии были разработаны до появления платформы **.Net Framework**. Они не могли учитывать особенности **.Net**. Именно поэтому и была разработана технология доступа к данным **ADO.NET**. Наличие корня **ADO** в названии технологии **ADO.NET** больше маркетинговый ход, чем указание на схожесть принципов использования с технологией **ADO**.

3. Сравнительный анализ технологий доступа к данным

Давайте теперь проанализируем, когда какую из перечисленных выше технологий стоит применять. Начнем по порядку. **ODBC –** наиболее старая технология доступа к данным из перечисленных выше. Она предоставляет большой и гибкий функционал для работы с реляционными источниками данных. Минусами данной технологии являются сложность использования, невозможность применения из ряда языков (например, **JScript**). **ODBC** стоит применять, когда вам необходима вся мощь, заложенная внутри неё, и при этом у вас есть необходимый драйвер ☺ **DAO** – технология, предназначенная для работы с процессором баз данных **Microsoft Jet** (**Microsoft Access**). Она позволяет просто и быстро получить необходимые данные и настроить нужные свойства базы данных **Jet**. К сожалению, на данный момент времени **DAO** является устаревшей и замороженной технологией. Её разработка закончилась в 2001 году.
OLE DB – технология, позволяющая работать как с реляционными, так и нереляционными источниками данных. Минусами данной технологии является сложность её использования, а также привязка к C и C++. Область применения данной технологии вполне очевидна: вы разработчик на C++, у вас есть провайдер для источника данных (это не обязательно реляционный источник данных).**ADO** – технология, представляющая набор объектов для высокоуровневого доступа к источникам данных вне зависимости от их типа. Как мы уже говорили, она является надстройкой

над **OLE DB**. Сфера применения такая же, как и у **OLE DB**, существенным отличием является то, что вы можете программировать на любом языке (технологии) поддерживающем использование **ActiveX**, например на **JScript**, **VB**, **Delphi**, и т.д.

ADO.NET – технология доступа к данным в рамках платформы **.Net Framework**. Она содержит набор пространств для доступа к технологиям **OLE DB** и **ODBC,** СУБД **MS SQL Server** и **Oracle**. Если вы программируете для платформы **.Net** то **ADO.NET** ваш очевидный выбор.

4. Сравнительный анализ понятий драйвер и провайдер

При обсуждении технологий доступа к данным мы несколько раз затрагивали понятия: драйвер и провайдер. Давайте немного подытожим всё вышесказанное. Драйвер – понятие характерное для технологии **ODBC**, обычно поставляется как набор бинарных библиотек, реализует **ODBC API** для доступа к конкретной базе, предоставляет доступ в процедурном стиле. Провайдер – понятие характерное для технологий **OLEDB**, **ADO**, **ADO.NET**, обычно поставляется в виде бинарных библиотек, реализует **COM**(или **.Net**) интерфейсы для доступа к конкретному источнику данных, доступ предоставляется в объектно-ориентированном виде. Основная цель и драйвера и провайдера - предоставить унифицированный доступ к источнику данных.

5. Пространства ADO.NET

Основным пространством имен для работы с архитектурой **ADO.NET** является **System.Data**, которое содержит набор классов и пространств, реализующих работу с **ADO.NET**. Рассмотрим пространства, входящие в **System.Data.**

Пространство имен	Описание
System.Data.Common	В этом пространстве имен содержатся классы для обобщенной работы с данными. Используя методы данного пространства, не нужно задумываться каким образом произошло подключение к данным. Но в случае, если нужно использовать какие-либо особенности поставщика данных (ODBC, OLE DB или какого-то другого) это пространство имен уже не подойдет. В частности в данном пространстве находятся базовые абстрактные классы (например, такие как DbConnection, DbCommand, DbDataAdapter, и так далее), которые переопределяются в специфических подпространствах, например, System.Data.OleDb
System.Data.OleDb	Используется для подключения к данным при помощи технологии OLE DB и работе с ними.
System.Data.SqlClient и System.Data.Sql	Используются для работы с базами данных SQL Server.
System.Data.Odbc	Используется для подключения к данным при помощи технологии ODBC и работы с ними.
System.Data.OracleClient	Используется для работы с базами данных Oracle.

6. Модели работы ADO.NET

В **ADO.NET** существует две модели работы с данными: **присоединенная** и **отсоединенная** модель. Традиционной моделью работы с данными является **присоединенная модель**. Работа присоединенной модели заключается в том, что приложение, открывая связь с поставщиком данных, остается на связи с ним на протяжении всего времени работы. Плюсом этой концепции является скорость работы с данными и их актуальность, а основным минусом – постоянное выделение ресурсов под соединение, что в случае с большим количеством подключений может вызвать серьезную перегрузку поставщика данных.

Для того чтобы избежать минусов работы с присоединенной моделью в **ADO.NET** существует и другая модель работы с данными - **отсоединенная модель**. Работа в отсоединенной модели происходит следующим образом: сначала подключаемся к поставщику данных, далее данные загружаются в объект **DataTable** (если результаты выборки можно представить в виде одной таблицы) или **DataSet** (если выборка более сложная), после чего можно отключиться и работать только с уже полученными данными. Для того чтобы внести изменения в данные, необходимо снова подключиться к поставщику. Эта модель снижает нагрузку на поставщика данных, но нужно помнить, что данные за время работы с ними в отсоединенном режиме могут устареть.

7. Концепция интерфейсов и базовых классов ADO.NET

Рассмотрим основные классы и интерфейсы **ADO.NET**

Интерфейс	Описание
IDbConnection	Представляет собой открытое подключение к источнику данных и реализуется поставщиками данных .NET Framework, которые имеют доступ к реляционным базам данных.
IDbCommand	Представляет собой оператор SQL, исполняемый при подключении к источнику данных и реализуемый поставщиками данных .NET Framework, которые имеют доступ к реляционным базам данных.
IDataReader	Позволяет читать один или несколько потоков результирующих наборов только в направлении вперед, выполняя команду над источником данных. Он реализуется поставщиками данных .NET Framework, обращающимися к реляционным базам данных.
IDataRecord	Предоставляет доступ к значениям столбцов в каждой строке для средства чтения DataReader, и реализуется поставщиками данных .NET Framework, которые имеют доступ к реляционным базам данных.
IDataAdapter	Позволяет объекту реализовать DataAdapter и представляет набор мето-

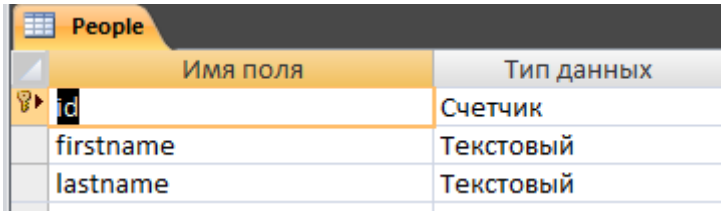
Интерфейс	Описание
	дов и соответствующие свойства, связанные с операциями, для заполнения и обновления объекта DataSet и обновления источника данных.
IDbDataAdapter	Представляет набор связанных с командой свойств, которые используются для заполнения DataSet и обновления источника данных, реализуемый поставщиками данных .NET Framework, осуществляющими доступ к реляционным базам данных.
IDataParameter	Представляет параметр для объекта Command, а также дополнительно — его сопоставление со столбцами DataSet. Реализуется поставщиками данных .NET Framework, которые осуществляют доступ к источникам данных.
IDbDataParameter	Используется в конструкторах данных Visual Basic .NET для представления параметра в объекте Command, а также дополнительно для его сопоставления со столбцами DataSet.
IDbTransaction	Представляет собой транзакцию, выполняемую в источнике данных и реализуемую поставщиками данных .NET Framework, осуществляющими доступ к реляционным базам данных.

Класс	Описание
DbConnection	Представляет подключение к базе данных.
DbCommand	Представляет оператор SQL или хранимую процедуру, выполняемую применительно к источнику данных. Предоставляет базовый класс для классов, зависящих от базы данных, представляющих команды.
DbDataReader	Считывает поток строк последовательного доступа из источника данных.
DbDataAdapter	Помогает реализации интерфейса IDbDataAdapter. Наследники DbDataAdapter реализуют набор функций, обеспечивающий строгую типизацию, но наследуют большинство функциональных возможностей, необходимых для полной реализации класса **DataAdapter**.
DbParameter	Предоставляет параметр для DbCommand и дополнительно его отображение для столбца DataSet.

Класс	Описание
DbTransaction	Базовый класс для транзакции.

8. Обзорный пример использования ADO.NET для доступа к источнику данных

Итак, настала пора немного приоткрыть завесу таинственности над **ADO.NET** и продемонстрировать пример использования данной технологии. Цель примера показать базовые принципы использования, более глубокое погружение в предметную область вас ожидает в дальнейших уроках. Как обычно пример будет располагаться в папке **Source** урока.
Название проекта **GetData**.
Исходные данные: база данных sample.mdb (формат **Microsoft Access**) с одной таблицей **people**. Структура таблицы:



Пример будет производить соединение с базой данных и отображение содержимого таблицы **people** на экран. Пример использует присоединенную модель. Код примера:

```
    static void Main(string[] args)
        {
            // создаём объект для соединения с базой
            OleDbConnection connection = new OleDbConnection();
            try
            {
                // Прописываем строку соединения
                // Указываем параметры необходимые для MS Access
                // Их два: имя провайдера (Provider) зарегистрированное в
```

```
системе
                // и путь к файлу базы данных (Data Source)
                connection.ConnectionString =
"Provider=Microsoft.Jet.OLEDB.4.0; Data Source=../../sample.mdb";
                // Открываем соединение
                // Если произойдет ошибка вылетит исключение
                connection.Open();
                // Создаём объект для исполнения запроса
                // Указываем ему в качестве параметра запрос и объект
открытого соединения
                OleDbCommand command = new OleDbCommand("SELECT * FROM
PEOPLE",connection);
                // Исполняем запрос и сохраняем ссылку на объект результата
                OleDbDataReader reader = command.ExecuteReader();
                // Проходимся по результатам работы запроса строка за строкой
                // Когда данные кончатся метод Read вернет false
                while (reader.Read() != false)
                {
                    // Так как в таблице три столбца в результате в строке
будет тоже три столбца
                    // Для обращения к столбцу используется индексатор
                    // Возможен доступ как по имени столбца так и по индексу
                    Console.WriteLine("{0,-10} {1,-10} {2,-
10}",reader["id"],reader["firstname"],reader["lastname"]);
                    /*
                      Console.WriteLine("{0,-10} {1,-10} {2,-10}", reader[0],
reader[1], reader[2]);
                     */
                }

            }
            catch(Exception ex){
                Console.WriteLine(ex.Message);
            }
            finally
            {
                // Закрываем соединение
                connection.Close();
            }
        }
```

Прокомментируем пример. Мы выбрали в качестве пространства **ADO.NET** для доступа к источнику данных подпространство **System.Data.OleDb**. Это связано с тем, что не существует специального пространства для данной СУБД и выбор фактически стоит между пространствами для технологий **ODBC** и **OLE DB**. Для соединения с базой данных используется класс **OleDbConnection**. Его основная цель настроить и открыть соединение.

Обычно при открытии соединения указывается строка с параметрами для соединения. Она передаётся либо в конструктор класса, либо задаётся

через свойство **ConnectionString** данного класса. Для открытия соединения используется метод **Open** данного класса. Он генерирует исключение, если произошла ошибка при открытии соединения. Для получения данных нужно пройти ещё два этапа:

1. Сформулировать и послать запрос
2. Получить результат работы запроса и строка за строкой отобразить его на экран консоли

Для реализации пункта один нужно воспользоваться объектом класса **OleDbCommand**. Основная цель данного класса состоит в создании запроса и его исполнении. Запрос можно передать в качестве параметра конструктора, либо же настроить через свойство класса. Для исполнения запроса, который вернет 0 и более строк результата используется метод **ExecuteReader**. Он запускает запрос на исполнение и возвращает объект содержащий результат. Для получения конкретных данных необходимо вызвать метод **Read** – основная цель, которого переходить от одной строки результат к другой, пока конец данных не будет достигнут

Важно отметить, что в рамках данной концепции используется так называемое понятие **курсора**. Курсор – специальный объект, ссылающийся на текущую строку. Перед первым вызовом метода **Read** курсор находится на строке перед первой строкой, вызов **Read** переводит курсор на первую строку выборки, если такая имеется, следующий вызов на вторую и так до тех пор, пока данные не кончатся, то есть пока мы не дойдем до строки, находящейся за последней строкой данных. Хотелось бы отметить следующий факт, при успешном переходе на новую строку **Read** возвращает **true**, при отсутствии данных **Read** вернет **false**.

Важно понимать, что каждый вызов **Read** предоставляет доступ только к одной строке данных, то есть фактически мы имеем дело с механизмом ”только вперед” (forward only). Для получения данных находящихся в текущей строке используется перегруженный индексатор, который предоставляет доступ, как по индексу, так и по имени столбца. После проведения всех операций важно не забыть закрыть источник данных. Для

этого необходимо вызвать метод **Close** класса **OleDbConnection**. Данный пример демонстрирует лишь некоторые базовые фрагменты мозаики, в следующих уроках вы более подробно познакомитесь с методами и классами, используемыми в коде, приведенном выше.

Домашнее задание:

Создайте базу данных Продажи (в качестве формата базы данных можно использовать **Microsoft Access**). Создайте внутри базы данных три таблицы:

a. Таблица Покупатели. Столбцы: идентификатор, имя, фамилия;
b. Таблица Продавцы. Столбцы: идентификатор, имя, фамилия;
c. Таблица Продажи. Столбцы: идентификатор сделки, идентификатор покупателя, идентификатор продавца, сумма сделки, дата сделки.

Наполните таблицы данными, задайте правила ссылочной целостности. Реализуйте **WinForms** приложение, позволяющее пользователю выбрать название таблицы из базы данных sample.mdb (например, с помощью выпадающего списка), в результате выбора таблицы приложение должно отображать содержимое данной таблицы на форму.