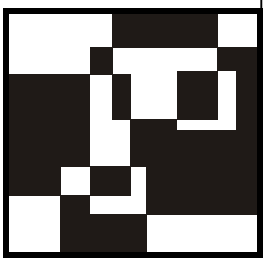
ГЛАВА 9



Работа с базами данных и применение объектной модели ADO

9.1. Зачем нужно работать с базами данных

Потребность в автоматизации (например, в создании программ VBA) возникает тогда, когда данных много. А если данных много, то они, скорее всего, будут храниться в базе данных — просто потому, что более удобного способа пока не придумано. Это относится к любым данным (в том числе к документам, графическим данным, архивам и т. п.).

Приложения Office и созданные на их основе программы при работе с данными могут быть очень полезными и сами по себе, но эффективность работы увеличивается многократно при сопряжении их с базами данных. Чаще всего в реальных приложениях Word используется для генерации отчетов на основе информации из базы данных, Excel — для анализа данных из базы, а Access — это сама по себе система управления базами данных (которая очень часто используется для построения клиентского интерфейса для внесения информации в клиент-серверные базы данных, такие как SQL Server и Oracle).

Потребности в обращении из приложений Office к базам данных возникают практически на любом предприятии. Очень часто приложение, которое изначально предназначалось для работы с данными в самом приложении (листе Excel, таблицы Word), по мере увеличения объема данных приходится переделывать под работу с клиент-серверными источниками. Поэтому в этой главе рассказывается о том, как можно подключаться к базам данных, скачивать и отображать в программе информацию из базы данных, вставлять новые записи, изменять или удалять существующие. Если у вас нет никакого опыта работы с базами данных, не пугайтесь. Как показывает опыт многих учебных

групп, знаний на уровне опытного администратора баз данных совсем не требуется. Освоить основные приемы работы с базами данных за несколько дней вполне способен каждый.

В этой главе речь пойдет об универсальных приемах работы с базами данных. Освоив их, вы сможете работать из приложений Office (и не только из них) с любыми базами данных: клиент-серверными, такими как Microsoft SQL Server, Oracle, IBM DB2, настольными, такими как Access, FoxPro, DBase, Paradox, и даже источниками данных, которые сами по себе базами не являются (например, иногда очень удобно подключиться к файлу Excel как к базе данных).

9.2. Что такое ADO

ADO расшифровывается как *ActiveX Data Objects* — набор программных объектов, построенных по технологии ActiveX (COM), которые позволяют получать данные из самых разных источников и управлять ими. Другие наборы программных объектов для доступа к источникам данных, которые часто используются в приложениях Office — это DAO и RDO, но эти программные объекты устарели и к использованию в современных приложениях не рекомендуются. В настоящее время появилась новая версия ADO — ADO.NET, которая сильно отличается от обычной ADO и предназначена для работы в .NET Framework. Однако по причине того, что ADO.NET:

- обязательно требует установленной .NET Framework (чего на многих старых компьютерах нет);
- обычными средствами с ADO.NET из редактора Visual Basic работать нельзя, требуется Visual Studio;
- отличается повышенной ресурсоемкостью.

ADO.NET в этой книге рассматриваться не будет.

ADO умеет работать с самыми разными драйверами для подключения к базам данных, например, с драйверами OLE DB и ODBC. Поскольку ADO построен по технологии COM, его можно использовать в любых COM-совместимых языках программирования (VC++, Visual Basic, Delphi, VBA, VBScript, JScript, ActivePerl и т. п.).

Сами программные объекты поставляются в наборе драйверов для подключения к базам данных, которые называются MDAC (Microsoft Data Access Components, компоненты доступа к данным Microsoft). Этот набор есть на любом компьютере под управлением Windows 2000, XP, 2003 (обычно сразу несколько версий). Самую свежую версию MDAC можно бесплатно скачать с Web-сайта фирмы Microsoft (**www.microsoft.com/download**). Настоятельно

рекомендуется отслеживать появление новых версий MDAC и устанавливать их на компьютерах пользователей.

К справке по ADO проще всего обращаться из Microsoft Access. Для этого нужно открыть Microsoft Access, создать в нем новую пустую базу данных, переключиться в окно редактора кода (при помощи клавиш <Alt>+<F11>) и нажать клавишу <F1>. Вторая строка в оглавлении справки — это справка по ADO. В нее входят **ADO Programmer's Guide** (рассказ о том, как можно выполнять различные действия), и **ADO Programmer's Reference** (справка по объектам ADO, их свойствам, методам и событиям) (рис. 9.1).

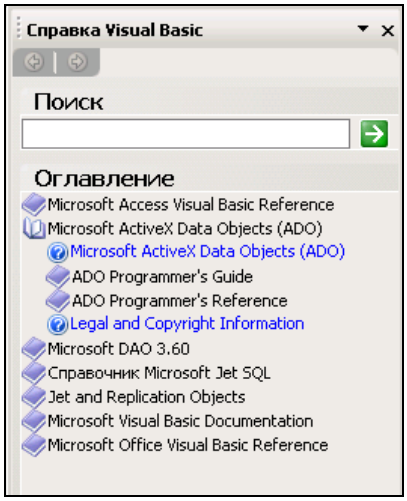


Рис. 9.1. Справка по ADO в Microsoft Access

Сама по себе объектная модель ADO очень проста и понятна. В нее включены всего три главных объекта.

- `Connection` — позволяет установить соединение с источником данных и управлять им. Все ошибки, которые возникают в ходе работы соединения, помещаются в сопутствующую коллекцию `Errors`.
- `Command` — представляет собой команду, при помощи которой производится выполнение определенной операции на источнике данных (выполнение запроса, хранимой процедуры, создание или изменение объекта, изменение данных и т. п.). Если источник данных SQL-совместимый, то объект `Command`, скорее всего, будет представлять команду SQL. Объекту `Command` сопутствует коллекция `Parameters` — параметры, которые передаются запросу или хранимой процедуре.
- `Recordset` — является набором записей, полученных с источника или сгенерированных другим способом. Этому объекту сопутствует коллекция

Fields, представляющая информацию о столбцах в этом наборе записей (имя, тип, размерность данных и т. п.), а также сами данные.

Для каждого из этих трех объектов предусмотрена также коллекция Properties, которая определяет свойства соединения, команды или набора записей соответственно.

Все объекты явно создавать необязательно. Так, например, при создании объекта Recordset можно в автоматическом режиме создать объект Connection.

Прежде чем приступить к работе с объектами ADO, необходимо добавить в ваш проект ссылку на необходимую библиотеку. Для этого в меню **Tools** выберите пункт **References** и установите флажок напротив строки **Microsoft ActiveX Data Objects** с нужным номером (в зависимости от того, какая версия библиотеки установлена на клиентских компьютерах, на которых будет работать ваше приложение). На момент создания этого курса последней версией MDAC была версия 2.8. Однако на практике рекомендуется выбирать более старую версию, например, версию 2.1, которая поставляется со всеми версиями Windows, начиная с Windows 2000. Функциональность разных версий практически одинакова, но при использовании этой версии вы сможете переносить свою программу на VBA с одного компьютера на другой, не беспокоясь о том, что на него нужно что-то доустанавливать.

9.3. Объект *Connection* и коллекция *Errors*

Создание объекта Connection выполняется очень просто. Например, чтобы подключиться к базе данных Northwind на сервере SQL Server с именем LONDON, можно использовать код типа:

```
Dim cn As New ADODB.Connection
cn.ConnectionString = "Provider=SQLOLEDB.1;Integrated
Security=SSPI;Initial Catalog=Northwind;Data Source=LONDON"
cn.Open
```

В принципе, этого вполне достаточно, чтобы подключиться к базе данных и создать объект соединения, который можно будет потом использовать. Однако у большинства пользователей возникает вопрос: а что написано в свойстве ConnectionString и как значение этого свойства можно написать самому?

Самый простой вариант — вообще ничего самому не писать. Значение для этого свойства можно сгенерировать в автоматическом режиме. Выглядит это очень просто:

1. Создаем любой пустой файл (например, текстовый). Для этого нужно щелкнуть правой кнопкой мыши на пустом месте в окне проводника Windows и в контекстном меню выбрать **New | Text Document** — рис. 9.2. И затем задать создавшемуся файлу любое название.

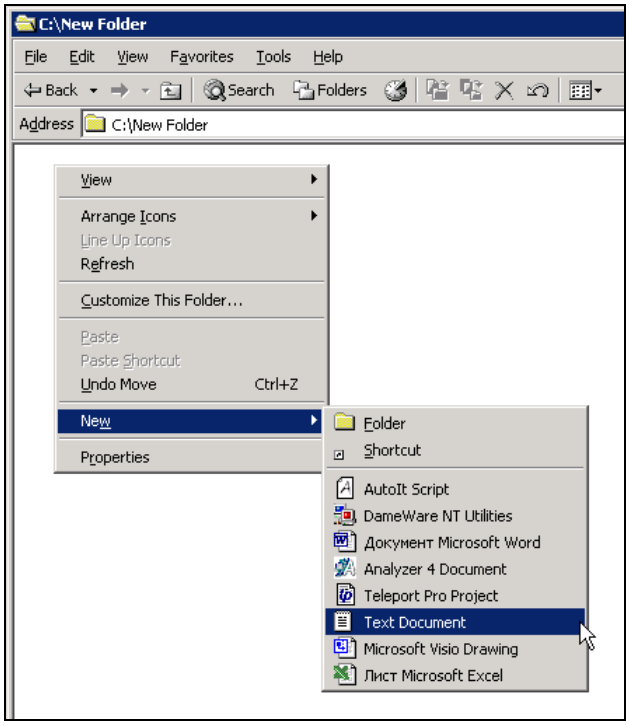


Рис. 9.2. Создание пустого текстового файла в Windows Explorer

2. Переименовываем файл так, чтобы у него было расширение udl (от *User Data Link* — пользовательское подключение к данным). После переименования убедитесь, что иконка для него изменилась (рис. 9.3).

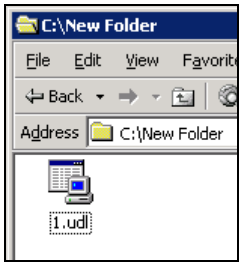


Рис. 9.3. Созданный файл 1.udl

Если она осталась такой же, как была для текстового документа, это значит, что реальное расширение для этого файла — txt, а не udl. В этом случае нужно в окне Windows Explorer в меню **Tools** выбрать **Folder Options**, в окне **Folder Options** перейти на вкладку **View** и снять флажок **Hide file**

extensions for known file types (скрывать расширения для известных типов файлов). После того, как вы выполните эту операцию, измените расширение файла на udl.

3. После того, как UDL-файл будет создан, просто щелкните по нему два раза левой кнопкой мыши. Откроется окно **Data Link Properties** с четырьмя вкладками.
4. На первой вкладке **Provider** выберите нужный тип базы данных (например, для базы данных SQL Server выберите **Microsoft OLE DB Provider for SQL Server**, для подключения к базе Oracle — **Microsoft OLE DB Provider for Oracle**, для подключения к базе Access — **Microsoft JET 4.0 OLE DB Provider**). Про подключение к листу Excel как к базе данных мы поговорим отдельно в *разд. 9.4*.
5. Далее перейдите на вкладку **Connection**. Для каждого типа базы данных эта вкладка выглядит по-своему. Например, для SQL Server она выглядит так, как представлено на рис. 9.4, а для Access — на рис. 9.5.

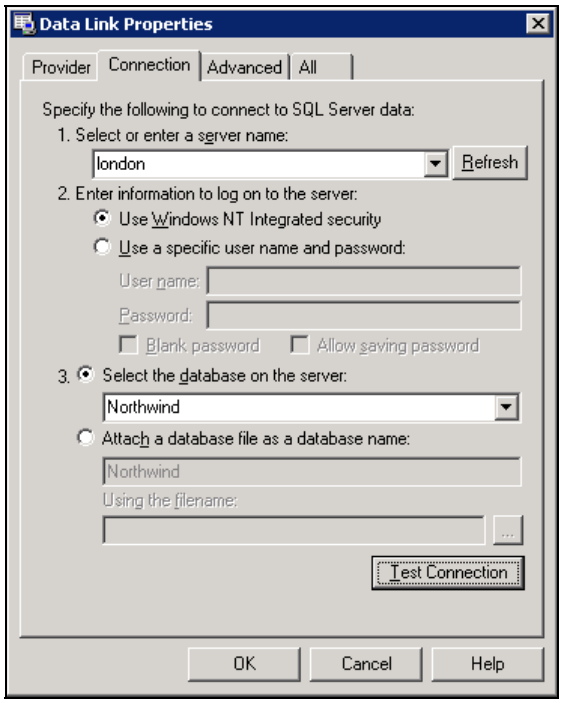


Рис. 9.4. Окно свойств соединения для подключения к SQL Server

Если вы не знаете, какие параметры вам нужно вводить в этом окне, спросите у вашего администратора базы данных. После того, как все парамет-

ры введены, рекомендуется нажать на кнопку **Test Connection**, чтобы проверить возможность подключения к базе данных. Если возникла ошибка, то нужно исправить введенные вами параметры свойств соединения. После этого нужно нажать кнопку **OK**, чтобы закрыть окно свойств соединения.

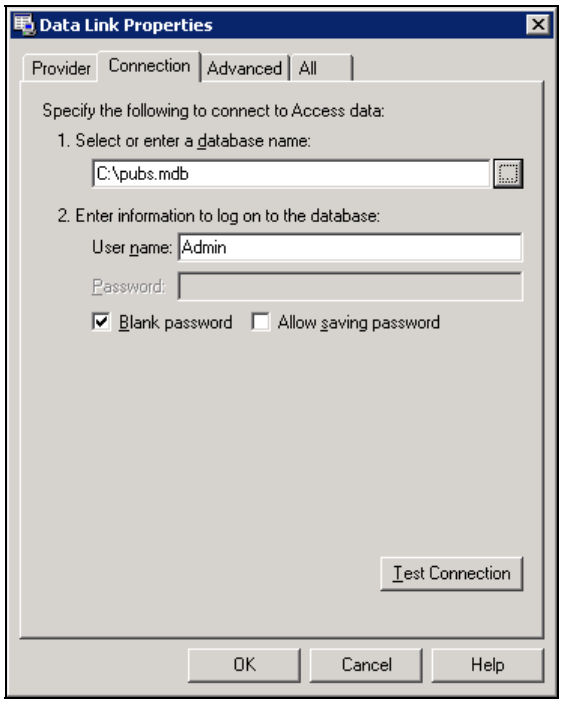


Рис. 9.5. Окно свойств соединения для подключения к базе данных Access

6. Последнее действие, которое нам потребуется сделать — щелкнуть правой кнопкой мыши по созданному UDL-файлу, в контекстном меню выбрать **Open With | Choose Program** и в появившемся списке выбрать **Notepad** (Блокнот) и нажать кнопку **OK**. Созданный вами файл откроется в блокноте. В меню **Format** в блокноте снимите флажок **Word wrap** и скопируйте в буфер обмена последнюю строку этого файла (рис. 9.6).

Внимание!

Если этой строке у вас есть русские буквы, то рекомендуется производить копирование при включенной русской раскладке клавиатуры, иначе русские символы будут заменены на знаки вопроса.

7. Осталось вставить скопированное значение в окно редактора кода как значение свойства `ConnectionString` и взять его в кавычки.

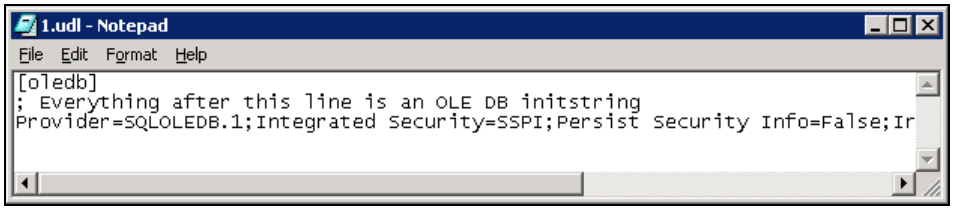


Рис. 9.6. Сгенерированная строка подключения

Конечно, вы вполне можете написать строку подключения *(connection string)* и вручную. Это просто набор параметров вида "*свойство=значение*", разделенных точкой с запятой (для значений в строке подключения кавычки не используются). Каждый из параметров означает следующее.

- `Provider` — драйвер для подключения к источнику данных. Для каждого типа источника данных (SQL Server, Access, Oracle) он свой. Мы с вами использовали драйвер OLE DB. Альтернатива ему — драйверы ODBC. Они работают медленнее и в основном используются для обеспечения обратной совместимости, но иногда без них не обойтись (например, когда подходящего драйвера OLE DB просто нет). Мы будем использовать подключение по ODBC к таблице на листе Excel.

  В принципе, в строке подключения значение для `Provider` можно не указывать вообще. Но тогда придется определить его как значение отдельного свойства объекта `Connection`.

- `Integrated Security` — в данном случае это свойство используется, поскольку мы применяем для подключения к SQL Server аутентификацию Windows. Если бы мы использовали аутентификацию SQL Server, то это свойство нам бы было не нужно, но вместо него нам потребовалось бы использовать два других свойства: `User ID` — идентификатор пользователя и `Password` — пароль. Если вы далеки от мира баз данных, просто узнайте необходимые значения у вашего администратора.

- `Data Source` — имя источника данных. В нашем случае это имя компьютера, на котором работает SQL Server. В других случаях оно могло бы быть именем экземпляра Oracle или файла базы данных Microsoft Access, все зависит от того, к какой базе данных вы подключаетесь.

- `Initial Catalog` — имя базы данных на этом сервере. В нашем случае это `Northwind`.

Когда вы генерируете строку подключения автоматически при помощи UDL-файла, в первый раз это может показаться долгим процессом. На самом деле это занимает не более минуты. Кроме того, при этом вы гарантируете, что в строке подключения не будет ошибок, и у вас есть возможность проверить

подключение к базе данных прямо из свойств UDL-файла (кнопка **Test Connection**).

Фактически все, что нужно для открытия соединения с базой данных, мы сделали: создали объект `Connection`, настроили для него свойство `ConnectionString` и вызвали метод `Open()`. Однако для справки приведем информацию об основных свойствах и методах этого объекта.

- `Provider` — это свойство позволяет определить драйвер, который будет использован для подключения к базе данных. Мы с вами определили такой драйвер внутри значения `ConnectionString`, но можно для этой цели использовать и отдельное свойство. Значения свойств `Provider` для подключения к разным источникам данных могут выглядеть так:
  - `"Microsoft.Jet.OLEDB.4.0"` — для подключений к файлам Access;
  - `"SQLOLEDB.1"` — для подключений к SQL Server (как в примере);
  - `"MSDAORA.1"` — для подключений к серверу Oracle;
  - `"ADsDSOObject"` — для подключения к базе данных службы каталогов Windows.
- `ConnectionString` — это главное свойство объекта `Connection`. Оно определяет параметры подключения к источнику. Как именно работать с ним, мы уже рассмотрели.
- `Open()` — этот метод позволяет открыть соединение с базой данных. Строку подключения можно не настраивать отдельно как свойство объекта `Connection`, а просто передавать ее этому методу как параметр.
- `Close()` — позволяет закрыть соединение. Учтите, что объект соединения при этом из памяти не удаляется. Чтобы полностью избавиться от этого объекта, можно использовать код:

  ```
  cn.Close
  Set cn = Nothing
  ```

  или просто:

  ```
  Set cn = Nothing
  ```

  а разрыв соединения произойдет автоматически.

Для объекта `Connection` предусмотрено множество других свойств и методов, однако здесь они рассматриваться не будут (за дополнительной информацией можно обратиться к документации или к учебным курсам Microsoft). Единственное свойство, которое обязательно нужно знать, — это свойство `Errors`, которое возвращает коллекцию объектов `Error` — ошибок. Ошибки при установке или работе соединения встречаются очень часто (неверно введен пароль или имя пользователя, у пользователя недостаточно прав на подключе-

ние, невозможно обратиться к компьютеру по сети и т. п.), поэтому настоятельно рекомендуется реализовывать в программе обработку ошибок. Самый простой вариант реализации обработчика ошибок может выглядеть так:

```
Dim cn As ADODB.Connection
Set cn = CreateObject("ADODB.Connection")
cn.Provider = "SQLOLEDB"
cn.ConnectionString = "User ID=SA;Password=password;" _
    & "Data Source = LONDON1;Initial Catalog = Northwind"
On Error GoTo CnErrorHandler
cn.Open
Exit Sub
CnErrorHandler:
    For Each ADOErr In cn.Errors
        Debug.Print ADOErr.Number
        Debug.Print ADOErr.Description
    Next
```

На практике при возникновении ошибки пользователю предлагается ее исправить и еще раз произвести подключение.

Для получения информации о том, почему возникла ошибка, используется специальный объект `ADOError` (при возникновении ошибки он создается автоматически). Далее представлены самые важные свойства этого объекта.

- `Description` — описание ошибки. Обычно наиболее важная информация содержится именно в описании.
- `Number` — номер ошибки. По номеру удобно производить поиск в базе знаний Microsoft (**www.microsoft.com/support**) и в Интернете.
- `Source` — источник ошибки. Эта информация полезна только в том случае, если в коллекции `Errors` могут оказаться ошибки из разных источников.
- `SQLState` и `NativeError` — информация о возникшей ошибке, которая пришла с SQL-совместимого источника данных.

9.4. Подключение к таблице на листе Excel

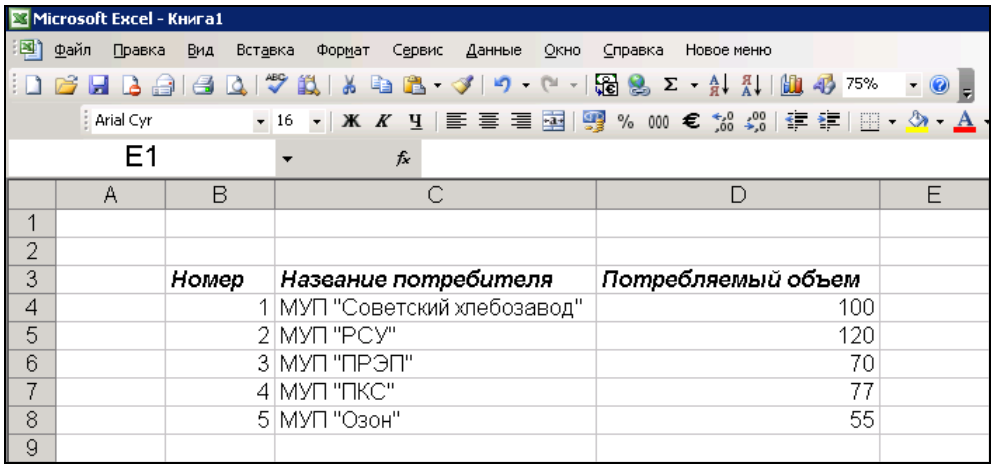
Очень часто в практической работе возникает необходимость подключиться к таблице на листе Excel как к базе данных. Конечно, можно работать и с объектной моделью Excel *(см. гл. 11)*, но использование объектов ADO дает значительные преимущества:

- намного проще и удобнее производить поиск записи, вставку новых записей в таблицу, изменение существующих записей. Объекты ADO изначально проектировались именно для этих целей;

- объектную модель Excel можно использовать только в Excel, а объекты ADO универсальны и могут применяться для подключения к любым источникам данных. Если вы используете объекты ADO, то можете использовать фактически одно и то же приложение как для работы с данными в Excel, так и для работы с информацией в "большой" базе данных, например, в SQL Server или Oracle. Ситуация, когда часть информации находится в базе данных, а другая часть — в книге Excel, встречается на практике очень часто.

Подключиться к таблице на листе Excel совсем не сложно, но самостоятельно догадаться до всей последовательности действий бывает трудно. Поэтому далее приведена пошаговая последовательность действий.

Предположим, что у нас есть книга Excel, которая называется Fact.xls и лежит в корневом каталоге диска C:. На первом листе этой книге есть несложная таблица, представленная на рис. 9.7.



	A	B	C	D	E
1					
2					
3		*Номер*	*Название потребителя*	*Потребляемый объем*	
4		1	МУП "Советский хлебозавод"	100	
5		2	МУП "РСУ"	120	
6		3	МУП "ПРЭП"	70	
7		4	МУП "ПКС"	77	
8		5	МУП "Озон"	55	
9					

Рис. 9.7. Таблица в Excel, к которой нужно обратиться средствами ADO

Нам необходимо подключиться к этой таблице, как к базе данных.

Первый этап — это подготовка. Иногда можно обойтись и без нее (если лист Excel — это одна таблица). На практике же часто бывает так, что на листе у нас несколько таблиц, или таблица с комментариями, или под таблицей посчитаны итоги и т. п. Чтобы не смущать Excel, лучше явно указать нашу таблицу. Сделать это очень просто: нужно ее выделить (в нашем случае — диапазон с B3 по D8) и присвоить выделенному диапазону имя. Для этого в Excel в меню **Вставка** нужно выбрать **Имя | Присвоить** и ввести имя. В нашем случае мы присвоили имя **Volumes** (рис. 9.8).

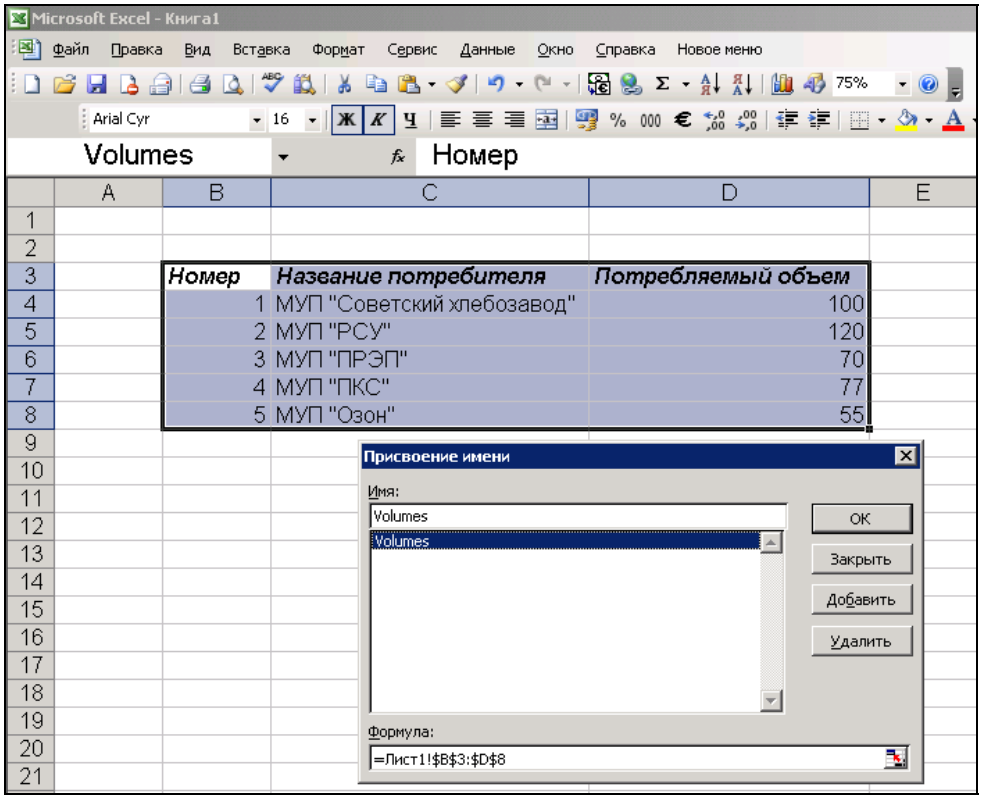


Рис. 9.8. Присвоение имени диапазону

Обратите внимание, что диапазон нужно выбирать вместе с названиями столбцов.

После того, как имя присвоено, документ Excel нужно сохранить и можно закрывать — он больше нам не нужен.

Далее по плану нужно было бы создать UDL-файл и настроить в нем подключение к нашему файлу C:\Fact.xls. Однако напрямую из UDL-файла можно работать только с драйверами OLE DB, а нужного нам драйвера, к сожалению, нет (Microsoft JET 4.0 OLE DB Provider хочет работать только с файлами mdb). Поэтому делаем еще один подготовительный шаг — создаем источник данных ODBC (поскольку драйвер ODBC для подключения к Excel есть). Первое действие — в **Панели управления** Windows открываем **Administrative Tools** (Средства администрирования) и два раза щелкаем по иконке **Data Sources (ODBC)** (Источники данных ODBC). Откроется окно, представленное на рис. 9.9.

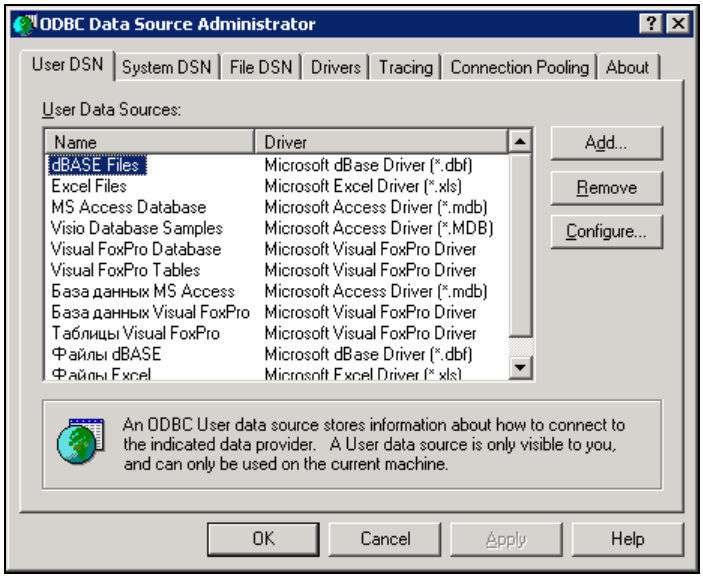


Рис. 9.9. Окно управления источниками данных ODBC

В вашем распоряжении три вкладки с DSN (*Data Source Name* — источников данных ODBC):

- **User DSN** (пользовательские источники данных) — информация об этих источниках данных хранится в части реестра, специфической для пользователя, поэтому эти источники данных доступны только тому пользователю, который их создал;
- **System DSN** (системные источники данных) — информация об этих источниках данных хранится в общей части реестра и доступна для всех пользователей на этом компьютере;
- **File DSN** (файловые источники данных) — информация об этих источниках данных записывается в файл в файловой системе.

Чаще всего используются системные источники данных, поэтому переходим на вкладку **System DSN** и нажимаем кнопку **Add**.

На первом экране мастера создания нового подключения нам потребуется ввести информацию о типе драйвера, который мы хотим использовать. Выбираем, конечно, **Microsoft Excel Driver** и нажимаем кнопку **Finish**. Но создание источника данных на этом далеко не кончилось.

На следующем экране мастера нам потребуется:

- в поле **Data Source Name** ввести имя источника данных. Можно ввести любое имя — главное, чтобы вы его не забыли. Введите имя **ExcelVolumes**;

- нажать кнопку **Select Workbook** и выбрать нужную рабочую книгу (в нашем случае это C:\Fact.xls);
- нажать кнопку **Options**, чтобы открылись дополнительные параметры подключения, и, если вы собираетесь изменять таблицу Excel из программы, снять флажок **Read Only**.

В итоге окно может выглядеть так, как представлено на рис. 9.10.

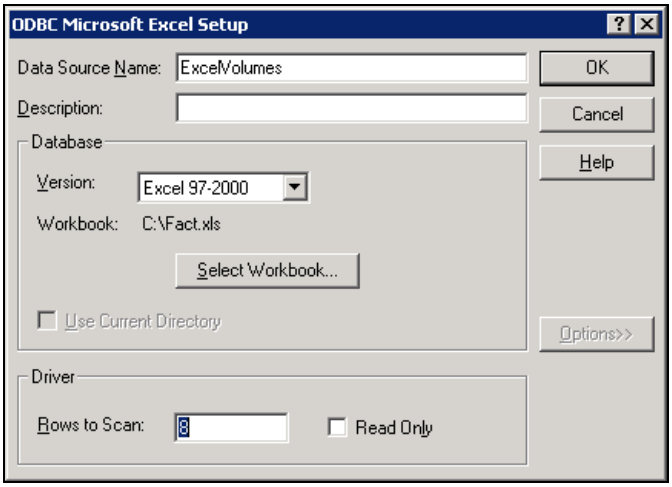


Рис. 9.10. Настроенный источник ODBC для подключения к файлу Excel

Осталось нажать две кнопки **OK**, чтобы закрыть окно создания источника данных ODBC.

В принципе, в коде программы можно написать значение свойства `ConnectionString` вручную, воспользовавшись документацией по ADO. Выглядеть соответствующая строка может, например, так:

```
cn.ConnectionString = "Provider=MSDASQL.1;DSN=FactExcel;" _
    & "DBQ=C:\Fact.xls;"
```

Но зачем писать что-то руками, когда очень просто можно сгенерировать нужное значение автоматически:

- так же, как было описано в предыдущем разделе, создаем UDL-файл (можно воспользоваться уже готовым);
- щелкаем по нему два раза мышью, переходим на вкладку **Provider** и выбираем **Microsoft OLE DB Provider for ODBC Drivers**;
- переходим на вкладку **Connection** и в списке **Use Data Source Name** выбираем созданный нами источник данных **ExcelVolumes**. Остальные поля

можно не заполнять (рис. 9.11). Для проверки можно нажать кнопку **Test Connection**, а затем **OK**.

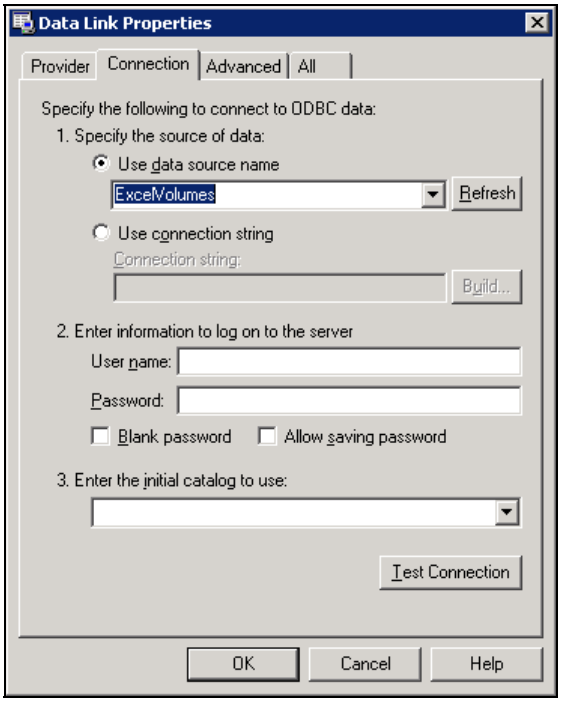


Рис. 9.11. Настройка параметров подключения к созданному источнику ODBC

- последнее действие — открываем созданный UDL-файл в блокноте, копируем из него строку подключения и используем в нашей программе.

Итоговый код процедуры для подключения к Excel может выглядеть так:

```
Public Sub ConnectToExcel()
    Dim cn As New ADODB.Connection
    cn.ConnectionString = "Provider=MSDASQL.1;" _
        & "Data Source=ExcelVolumes"
    cn.Open

    'Про Recordset мы будем говорить в следующем разделе
    'Этот код помещен для наглядной проверки
    Dim rs As New ADODB.Recordset
    rs.Open "Volumes", cn
    MsgBox rs.GetString

End Sub
```

Чтобы подключиться к файлу Excel, нам потребовалось:

- создать именованный диапазон в книге Excel;
- создать источник данных ODBC с именем **ExcelVolumes**;
- написать три строки кода, начиная с создания объекта `Connection` до вызова его метода `Open`.

9.5. Объект *Recordset* и коллекция *Fields*

9.5.1. Открытие *Recordset*

Обычно следующий после установки соединения этап — это создание объекта `Recordset` и работа с ним.

Что такое объект `Recordset`? Само слово `Recordset` расшифровывается как *Set of Records*, т. е. набор записей. Проще всего представить его как таблицу (аналогичную таблицам в Excel), которая находится в оперативной памяти компьютера. Однако у `Recordset` есть принципиальные отличия от таблиц Excel:

- Excel не следит за "строгостью" таблиц. На предприятиях часто можно встретить таблицы, в середину которых вставлены, например, промежуточные итоги по группам или заметки. `Recordset` — это "строгая" таблица. В ней четко определены столбцы и строки и разрывов она не допускает (хотя какие-то значения на пересечении строк и столбцов вполне могут быть пустыми);
- в таблице Excel в одном столбце без проблем можно использовать самые разные значения — числовые, даты и времени, строковые, формулы и т. п. В `Recordset` для каждого столбца определяется тип данных и значения в этом столбце должны соответствовать этому типу данных.

`Recordset` обычно создается на основе данных, полученных с источника (но может быть создан и заполнен вручную), в которых предусмотрены столбцы *(Fields)* и строки *(Rows)*. Создание объекта `Recordset` и заполнение его данными с источника в самом простом варианте выглядит так (подразумевается, что мы открыли при помощи объекта `cn` соединение с учебной базой данных `Northwind` на SQL Server):

```
Dim rs As New ADODB.Recordset
rs.Open "customers", cn
```

Убедиться, что `Recordset` действительно создан и существует, можно, например, при помощи строки:

```
MsgBox rs.GetString
```

При открытии `Recordset` вполне могут возникнуть ошибки, поэтому рекомендуется использовать обработчик ошибок. Специальной коллекции `Errors` в `Recordset` не предусмотрено, а значит, придется обойтись стандартным объектом `Err`.

В нашем примере мы открыли таблицу `Customers` целиком. Однако это не единственный (и не лучший) способ извлечения данных с источника. Для метода `Open()` рекомендуется использовать запрос на языке SQL. Например, в нашем случае можно было бы использовать такой код:

```
rs.Open "select * from dbo.customers", cn
```

Запрос использовать лучше, потому что:

- есть возможность указать фильтр `Where` (условие обязательно заключать в одинарные кавычки) и скачать в `Recordset` не все записи, а только удовлетворяющие вашему условию;
- есть возможность точно так же ограничить число возвращаемых столбцов — снова сокращение объема передаваемых данных и уменьшение расхода памяти;
- есть возможность использовать функции SQL, сортировку на источнике данных и множество полезных дополнительных возможностей.

Очень часто в реальных приложениях текст запроса "склеивается" из кусочков, которые поступают из разных мест. Например, пользователь выбрал в раскрывающемся списке имя заказчика, и для события `Change` этого списка тут же сработала процедура, которая выполнила запрос на SQL Server, получив данные только для этого заказчика, и присвоила полученные значения другим элементам управления. В нашем случае соответствующая строка кода может выглядеть так:

```
rs.Open "select * from dbo.customers Where CompanyName = " & "'" _
        & ComboBox1.Value & "'" , cn
```

Набор символов `"'"` — это одинарная кавычка внутри двух двойных. Такая конструкция нужна, чтобы текст запроса мог выглядеть, например, так:

```
select * from dbo.customers Where CompanyName = 'Alfreds Futterkiste'
```

Причина проста — в языке SQL строковые значения нужно заключать в одинарные кавычки.

Если вы ответственны не только за создание клиентского приложения, но и за проектирование базы данных, бывает очень удобно предусмотреть запрос данных только через представления. Это позволит более гибко управлять системой безопасности и в случае необходимости перестройки базы данных

(например, разбиения главной таблицы на текущую и архивную) сэкономить множество времени.

И еще один практический момент. Конечно, для работы с базами данных знать язык запросов SQL очень полезно. Литературы по нему очень много, и его вполне реально освоить за несколько дней. Однако, если вы — обычный пользователь и не имеете об языке SQL никакого представления, ничего страшного. Просто открывайте таблицы целиком без всяких запросов, а все остальное можно будет сделать средствами VBA.

9.5.2. Настройки курсора и другие параметры открытия *Recordset*

При открытии объекта `Recordset` можно определить еще несколько важных его свойств (их можно определить как напрямую перед открытием, так и передать как дополнительные параметры методу `Open()`).

- Первое свойство — `CursorType`, тип курсора. Это свойство определяется только перед открытием `Recordset` (после открытия оно доступно только для чтения). Курсор можно представить себе как указатель на записи в `Recordset`. В зависимости от типа курсора мы определяем возможности работы с `Recordset` и производительность выполняемых операций (чем больше возможностей, тем меньше производительность, и наоборот). Можно задавать следующие значения:
  - `adOpenForwardOnly` — это значение используется по умолчанию. Оно оптимизировано для достижения максимальной производительности (его возможности будут минимальными). Курсор может двигаться только вперед, а изменения, вносимые другими пользователями, видны не будут;
  - `adOpenStatic` — то же самое, что и предыдущее значение, за исключением того, что курсор может двигаться во всех направлениях;
  - `adOpenKeyset` — позволяет двигаться курсору в любом направлении, видны только изменения существующих записей другими пользователями (удаление старых записей и добавление новых не видны);
  - `adOpenDynamic` — обеспечивает максимальные возможности: позволяет двигаться в любых направлениях, видны любые изменения в записях, производимые другими пользователями. К сожалению, провайдер `Microsoft.Jet.OLEDB.4.0` этот тип курсора не поддерживает, поэтому с Access и Excel его использовать не получится.

  Свойство `Recordset.RecordCount` нормально функционирует только для курсоров типа `adOpenStatic` и `adOpenKeyset`. Для курсоров типа

`adOpenForwardOnly` и `adOpenDynamic` оно возвращает –1, поскольку драйвер подключения не может определить количество записей.

- Второе важное свойство — `CursorLocation`. Оно определяет, где будет создан курсор — на сервере или на клиенте. По умолчанию используется значение `adUseServer` (создавать на сервере). Второе значение — `adUseClient` (создавать на клиенте). В целом, практически во всех ситуациях удобнее и производительнее использовать серверные курсоры, за одним исключением — в реализации серверных курсоров на разных источниках данных много отличий, поэтому если вы планируете работать с разными источниками, имеет смысл подумать о клиентских курсорах.
- Третье важное свойство — `LockType`. Это свойство определяет тип блокировок, которые будут наложены на записи на источнике, помещенные в `Recordset`. Можно использовать следующие значения:
  - `adLockReadOnly` — записи в `Recordset` будут доступны только для чтения, вы не сможете их изменять. Это значение используется по умолчанию;
  - `adLockPessimistic` — наиболее надежный, с точки зрения целостности данных, вид блокировки. Вы можете изменять записи в `Recordset`, но при начале изменения записи она блокируется на источнике таким образом, что другие пользователи не смогут обратиться к ней ни на чтение, ни на запись, пока вы не вызовете методы `Update()` или `CancelUpdate()`;
  - `adLockOptimistic` — это значение позволяет выиграть в производительности за счет проигрыша в надежности обеспечения целостности данных. Запись на источнике блокируется только на время выполнения метода `Update()`. Остальные пользователи могут одновременно с вами читать и изменять данные на источнике;
  - `adLockBatchOptimistic` — то же самое, что и `adLockOptimistic`, но вместо немедленного обновления по одной записи используется пакетное обновление. В ситуации, когда изменяется большое число записей, такое решение позволяет выиграть в производительности.

Первый параметр метода `Open()` в наших примерах был как именем таблицы, так и командой SQL (могут использоваться и другие варианты). Поскольку драйвер OLE DB не знает, чем может быть передаваемый текст, он взаимодействует с сервером баз данных, чтобы определить это. На практике такое выяснение может сильно тормозить работу приложения, поэтому имеет смысл перед открытием `Recordset` явно указать тип передаваемых параметров. Это делается при помощи параметра `Options`, который передается этому методу.

Наиболее часто используемые значения такие:

- `adCmdText` — передается команда SQL;
- `adCmdTable` — передается имя таблицы (равносильно указанию сгенерировать команду SQL, которая вернет все записи из таблицы);
- `adCmdTableDirect` — также передается имя таблицы для того, чтобы получить все ее записи напрямую (без выполнения SQL-запроса), если источник поддерживает такую операцию;
- `adCmdStoredProc` — передается имя хранимой процедуры для ее выполнения, а то, что она вернет, используется для заполнения `Recordset`.

С практической точки зрения важно запомнить следующее. Если вам нужно обеспечить себе возможность перемещения по `Recordset` в любом направлении и изменять в нем записи, код на открытие `Recordset` должен быть таким:

```
Dim rs As New ADODB.Recordset
rs.CursorType = adOpenStatic
rs.LockType = adLockOptimistic
rs.Open ... 'Пишем, что именно мы открываем
```

На практике я пишу строки:

```
rs.CursorType = adOpenStatic
rs.LockType = adLockOptimistic
```

совершенно автоматически, поскольку они нужны в подавляющем большинстве случаев. Какие-то другие значения этих свойств нужно использовать только в специальных ситуациях (например, когда важнее всего производительность).

9.5.3. Перемещение по *Recordset*

После того как объект `Recordset` создан, нам необходимо выполнять с ним различные операции. Самое простое действие, с которого мы начнем, — перемещение по объекту `Recordset`.

В `Recordset` всегда имеется ограниченное количество записей (столько, сколько мы получили с источника). Изначально курсор устанавливается на первую запись в `Recordset` (убедиться в этом можно при помощи свойства `AbsolutePosition`). Однако если мы выполним команду `MovePrevious()` (но только один раз), ошибки не произойдет, а если мы попробуем выполнить команду `MovePrevious()` второй раз, то возникнет ошибка. `AbsolutePosition` вернет загадочное значение –2. Связано это с тем, что в `Recordset` перед первой записью, полученной с источника, помещается специальная запись `BOF`

(от англ. *Begin Of File*, хотя никаких файлов, конечно же, нет). Проверить, находимся ли мы на этой специальной записи, можно при помощи свойства `BOF`. Например, такой код (если он выполнен сразу после открытия `Recordset`):

```
Debug.Print rs.BOF
rs.MovePrevious
Debug.Print rs.BOF
```

вернет нам вначале `False`, а затем `True`.

Точно так же после последней записи в `Recordset` находится специальная запись `EOF` (от *End Of File*). Проверить, не находится ли курсор на ней, можно при помощи аналогичного одноименного свойства `EOF`.

Иногда бывает так, что сразу после открытия `Recordset` и `BOF`, и `EOF` одновременно возвращают `True`. Объяснение такой ситуации очень простое — в `Recordset` с источника по каким-то причинам не вернулось ни одной записи. Рекомендуется во избежание неожиданностей предусматривать сразу после открытия `Recordset` проверку на наличие в нем записей.

После того, как мы определились с текущей позицией в `Recordset`, необходимо разобраться с тем, как можно по нему перемещаться. Проще всего это делать при помощи методов с префиксом `Move...`

- `Move()` — этот метод принимает два параметра: `NumRecords` — на сколько записей необходимо переместиться (это число может быть и отрицательным, что означает переместиться назад) и второй параметр (необязательный) — имя закладки, с которой нужно начать перемещение. Можно использовать три встроенные закладки: для текущей, первой и последней записи. Если имя закладки не указано, то перемещение начинается с текущей позиции.
- `MoveFirst()`, `MoveLast()`, `MoveNext()` и `MovePrevious()` — назначения этих методов понятны из названий: перемещение на первую, последнюю, следующую и предыдущую запись соответственно.

Необходимо отметить, что перемещение назад (при помощи `MovePrevious()` или `Move()` с отрицательным значением) для курсора, открытого как `adOpenForwardOnly`, может привести к совершенно непредсказуемому результату (в зависимости от источника данных) — от ошибки до перехода на случайную запись.

Чаще всего для прохода по всем записям используется такой нехитрый алгоритм:

```
rs.MoveFirst
Do Until rs.EOF
```

```
    'Что-то делаем с каждой записью
    'Например, получаем значение нужного поля
    rs.MoveNext
Loop
```

Если необходимо напрямую перепрыгнуть на нужную запись, можно использовать методы `Find()` и `Seek()`.

- `Find()` — предназначен для поиска по значению одного столбца. Он принимает в качестве параметра критерий поиска, насколько нужно отступить от исходной позиции, направление поиска и откуда нужно начать поиск. Очень удобно, что при определении критерия поиска можно использовать оператор `Like` с подстановочными символами. При обнаружении нужной записи метод `Find()` переставляет курсор на найденную запись, если же запись не обнаружена, то курсор устанавливается на `EOF` (или `BOF`, если поиск был в обратном порядке). Например, чтобы найти все немецкие фирмы в нашем `Recordset` для таблицы `Customers`, можно использовать код вида:

  ```
  rs.Find "country = 'Germany'"
  Do While Not rs.EOF
      Debug.Print "Название фирмы: " & rs.Fields("CompanyName")
      mark = rs.Bookmark
      rs.Find "country = 'Germany'", 1, adSearchForward, mark
  Loop
  ```

  В этом примере используются еще незнакомые нам объекты `Fields` и `Bookmark`, но их назначение понятно: объект `Field` нужен, чтобы вывести название фирмы, а объект `Bookmark` — чтобы продолжить поиск, начиная со следующей записи по отношению к последней найденной.

- `Seek()` — отличается от метода `Find()` тем, что он ищет значение по индексу (объект `Index` для `Recordset` создается либо программным способом, либо автоматически, если на таблицу, на основе который был создан `Recordset`, было наложено ограничение первичного ключа *(Primary Key)*). Этот метод работает только для серверных курсоров с типом команды `TableDirect`, и поэтому к использованию не рекомендуется.

Хочется упомянуть о еще одном свойстве, которое может сильно помочь в перемещении по `Recordset` (и которое уже встречалось в наших примерах) — свойстве `Bookmark`. Это свойство очень простое — достаточно присвоить его значение переменной типа `Variant`, когда указатель стоит в нужном месте `Recordset`, а затем присвоить этому свойству значение этой переменной, чтобы опять на него вернуться, как в нашем примере с поиском.

Вообще говоря, значение, которое возвращает это свойство, изначально совпадает с номером записи в `Recordset`, однако Microsoft честно предупреждает, что таким способом пользоваться закладкой очень не рекомендуется — если курсор стоит в одном и том же месте, свойство `Bookmark` может возвращать разные значения.

9.5.4. Коллекция *Fields* и объекты *Field*

Главное содержание `Recordset` — это то, что лежит в ячейках на пересечении строк (в `Recordset` они называются записями *(records)* и представлены объектами `Record`) и столбцов. В `Recordset` столбцы называются полями и представляются объектами `Field`, которые сведены в коллекцию `Fields`. Объекты `Record` используются нечасто, поскольку имен у них нет, а переходить между записями проще при помощи свойств и методов самого объекта `Recordset` — `AbsolutePosition`, `Find()`, `Move()` и т. п. Коллекция же `Fields` и объекты `Field` используются практически в каждой программе.

У коллекции `Fields` все свойства стандартные, как у каждого объекта `Collection`.

- `Count` — возвращает, сколько всего столбцов в `Recordset`.
- `Item` — позволяет вернуть нужный столбец (объект `Field`) по имени или номеру. Поскольку это свойство является свойством по умолчанию, то можно использовать код, как в нашем примере:

  ```
  rs.Fields("CompanyName")
  ```

  Есть еще один вариант синтаксиса для обращения к этому свойству:

  ```
  rs!CompanyName
  ```

Методы у этой коллекции есть как стандартные, так и специфические.

- `Append()` — добавляет новый столбец в `Recordset`. `Delete()` — удаляет столбец. Обе команды разрешено выполнять только на закрытом `Recordset` (пока не был вызван метод `Open()` или не установлено свойство `ActiveConnection`).
- `Update()` — сохраняет изменения, внесенные в `Recordset`. Метод `CancelUpdate()` — отменяет изменения, внесенные в `Recordset`.
- `Refresh()` — загадочный метод, который ничего не делает (о чем честно написано в документации). Обновить структуру `Recordset` данными с источника можно только методами самого объекта `Recordset`.
- `Resync()` — работает только для коллекции `Fields` объекта `Record` (не `Recordset`), обновляя значения в строке.

Намного больше интересных свойств у объекта Field.

- ActualSize — реальный размер данных для текущей записи, DefinedSize — номинальный размер данных для столбца (в байтах), в соответствии с полученной с источника информацией.
- Attributes — определяет битовую маску для атрибутов столбца (допускает ли пустые значения, можно ли использовать отрицательные значения, можно ли обновлять, используется ли тип данных фиксированной длины и т. п.)
- Name — просто строковое имя столбца. Для столбцов, полученных с источника, это свойство доступно только для чтения.
- NumericScale и Precision — значения, которые определяют соответственно допустимое количество знаков после запятой и общее максимальное количество цифр, которое можно использовать для представления значения.
- Value — самое важное свойство объекта Field. Определяет значение, которое находится в столбце (если мы пришли через коллекцию Fields объекта Record, то значение этой записи Record; если через Fields объекта Recordset — текущей записи). Доступно и для чтения, и для записи (в зависимости от типа указателя). ADO позволяет работать с большими двоичными данными (изображения, документы, архивы), что очень удобно. Свойство OriginalValue возвращает значение, которое было в этом столбце до начала изменений, UnderlyingValue — значение, которое находится на источнике данных (пока мы работали с Recordset, оно могло быть изменено другим пользователем, и поэтому OriginalValue и UnderlyingValue могут не совпадать). Свойство Value — это свойство объекта Field по умолчанию (т. е. то свойство, значение которого будет возвращаться, если не указывать, к какому свойству объекта мы обращаемся), поэтому следующие две строки равноценны:

```
Debug.Print rs.Fields("CompanyName")
Debug.Print rs.Fields("CompanyName").Value
```

- Status — значение этого свойства, отличное от adFieldOK (значение 0), означает, что поле было недавно программно добавлено в Recordset.
- Type — определяет тип данных столбца в соответствии с приведенной в документации таблицей. Например, для типа данных nvarchar возвращается 202.

У объекта Field есть только два метода: AppendChunk() и GetChunk(). Оба эти метода используются только для работы с большими двоичными типами данных (изображениями, документами и т. п.), когда использовать обычными способами свойство Value не получается.

9.5.5. Сортировка и фильтрация данных

Данные в `Recordset` помещаются в том порядке, как они пришли из источника. Если специальный порядок сортировки в запросе не указан, то данные возвращаются в соответствии с параметрами источника данных (например, на SQL Server они будут по умолчанию упорядочены по кластерному индексу, который по умолчанию создается для первичного ключа). Можно произвести сортировку на сервере, указав в запросе выражение `ORDER BY`, а можно выполнить сортировку на клиенте при помощи свойства `Sort` объекта `Recordset`.

Общее правило выглядит так: если есть возможность, всегда нужно выполнять сортировку на сервере. Сервер намного лучше оптимизирован для выполнения подобных операций, на нем обычно больше оперативной памяти и процессорных ресурсов. На клиенте сортировку выполнять следует только тогда, когда это невозможно сделать на сервере (например, вы получаете данные от хранимой процедуры, в тексте которой сортировка не предусмотрена). Однако, в принципе, сортировка в `Recordset` менее ресурсоемка, чем могла бы быть (данные физически не перемещаются, только создается новый индекс для поля, по которому производится сортировка), поэтому ее вполне можно использовать в программах даже для большого количества данных.

Применение свойства `Sort` связано с двумя серьезными ограничениями:

- его можно использовать только тогда, когда курсор открыт на клиенте (т. е. для свойства `CursorLocation` должно быть установлено значение `adUseClient`, по умолчанию он открывается на сервере);
- это свойство нельзя использовать с некоторыми драйверами, например, нельзя сортировать данные при подключении к Excel.

Применение этого свойства выглядит очень просто:

```
rs.Sort = "CompanyName"
```

При этом то, что передается этому свойству, должно быть названием столбца (т. е. именем объекта `Field`) в `Recordset`.

Можно передавать несколько названий столбцов и разные порядки сортировки:

```
rs.Sort = "Country ASC, CompanyName DESC"
```

`Recordset` вначале будет отсортирован по стране (`Country`), а потом — по имени компании (`CompanyName`) в убывающем порядке `DESC` (по умолчанию используется `ASC` — по возрастанию, поэтому в нашем примере это слово можно опустить).

Чтобы отменить сортировку (и вернуться к записям в том порядке, в котором они были возвращены с источника), достаточно присвоить этому свойству пустое значение:

```
rs.Sort = ""
```

В Recordset записи можно фильтровать. Отфильтрованные записи остаются в Recordset, но являются невидимыми при выполнении операций перемещения и поиска, и курсор на отфильтрованных записях установить нельзя.

Для фильтрации используется свойство Filter объекта Recordset. Оно может принимать три типа значений:

- строковое значение, по синтаксису аналогичное передаваемому методу Find():

  ```
  rs.Filter = "LastName = 'Smith' AND FirstName = 'John'"
  ```

  Можно использовать оператор Like с подстановочными символами (только '*' и '%'). Отличие от Find() заключается в том, что в Find() просто устанавливается курсор на первую найденную подходящую запись, а в Filter все неподходящие становятся невидимыми;
- массив закладок;
- несколько специальных значений — все конфликтующие записи, записи, ожидающие сохранения на источнике, и т. п.

Снять фильтрацию можно точно так же, как и сортировку:

```
rs.Filter = ""
```

9.5.6. Изменение записей на источнике при помощи объекта *Recordset*

Очень часто возникает необходимость из приложения не только получить информацию о записях с источника, но и внести на источник изменения. При этом обычно возникает множество сложностей, связанных с решением вопроса о том, в какой таблице данные изменять (если у нас набор таблиц), с блокировками, производительностью, разрешениями, каскадными обновлениями, возможностью отката внесенных пользователем изменений и т. п. Многие проблемы решаются намного проще, если вы изначально будете следовать правилу: любые изменения можно проводить только при помощи хранимых процедур (и, соответственно, при помощи объекта Command). Далее будут рассмотрены возможности внесения изменений через объект Recordset, которые следует использовать только в самых простых случаях.

Необходимо также помнить, что значение свойства `LockType` при открытии `Recordset` по умолчанию устанавливается в `adLockReadOnly`, что не позволяет вносить изменения в `Recordset`. Вам потребуется изменить значение этого свойства перед открытием `Recordset`.

Общая схема внесений изменений через `Recordset` выглядит так: вначале вызывается один из нужных нам методов (`AddNew()`, `Edit()`, `Delete()`) и производится внесение изменений в оперативной памяти на клиенте. Следующая операция — вызов метода `Update()` для `Recordset`, который и производит запись внесенных изменений на источник данных (после вызова `Delete()` вызывать метод `Update()` не нужно). Далее приведена информация о каждом из этих методов.

- Работа с методом `AddNew()` — это всегда операция, которая состоит из трех частей:
  - вначале нужно вызвать этот метод, чтобы создать новую пустую запись (курсор будет установлен на нее автоматически);
  - занести значения в столбцы, используя свойство `Value` коллекции `Fields`;
  - вызвать метод `Update()` для записи изменений на источник.

  Пример использования этого метода к нашему `Recordset` может выглядеть так:

```
Dim rs As ADODB.Recordset
Set rs = CreateObject("ADODB.Recordset")
rs.CursorType = adOpenKeyset
rs.LockType = adLockOptimistic
rs.Open "select * from dbo.customers", cn
rs.AddNew
rs.Fields("CompanyName") = "Test rs company"
rs.Fields("Country") = "Germany"
rs.Fields("CustomerId") = "TESTR"
rs.Update
```

  В принципе, можно присваивать новые значения и в самом методе `AddNew()`, но с точки зрения синтаксиса это сложнее.

- Изменение существующей записи при помощи метода `Edit()` выглядит очень просто:

```
rs.Find "CustomerID='ALFKI'"
rs.Fields("ContactName") = "Маша"
rs.Update
```

- Удаление записи с помощью метода `Delete()` — еще проще:

```
rs.Find "CustomerID='TESTR'"
rs.Delete
```

Обратите внимание, что в этом случае метод `Update()` вызывать не нужно!

Конечно, при внесении изменений на источник ошибки могут возникнуть по множеству причин. Рекомендуется всегда в таких ситуациях устанавливать обработчик ошибок и анализировать свойства стандартного объекта `Err`.

9.5.7. Прочие свойства и методы объекта *Recordset*

Далее рассказывается о некоторых дополнительных свойствах и методах объекта `Recordset`, которые используются намного реже.

- `AbsolutePage`, `PageSize`, `PageCount` — эти свойства позволяют использовать группы записей (страницы) для перемещения по `Recordset`. По умолчанию размер страницы равен 10 записям.
- `ActiveCommand` — позволяет вернуть объект `Command`, представляющий команду, которая использовалась на источнике при создании `Recordset` и заполнении его записями. Подробнее про объект `Command` будет рассказано в *разд. 9.6*.
- `ActiveConnection` — возвращает объект `Connection`, который использовался для создания `Recordset`. Передать (или получить) строковое значение, на основе которого будет создан объект `Connection`, можно при помощи свойства `Source`.
- `CacheSize` — позволяет определить количество записей, которые будут находиться в оперативной памяти на клиенте (остальные записи будут подкачиваться по мере необходимости с источника). Используется тогда, когда количество записей в `Recordset` очень большое или приходится работать с записями очень большого размера, например, с большими двоичными объектами.
- `EditMode` — позволяет определить состояние текущей записи: не изменялась, изменялась, но изменения еще не переданы на источник, удалена и т. п.
- `InsertCommand`, `DeleteCommand`, `UpdateCommand` — позволяют определить объекты `Command`, представляющие команды, которые будут использоваться на источнике при создании, удалении и изменении записей в `Recordset` соответственно.

- `MarshalOptions` — позволяет определить, какие записи при изменении `Recordset` будут возвращаться с клиента на сервер: все (по умолчанию) или только измененные.
- `MaxRecords` — это свойство настоятельно рекомендуется указывать перед открытием для всех `Recordset`, для которых существует возможность получить с источника очень большое количество записей (что может привести к нехватке оперативной памяти на клиенте). Оно определяет максимальное количество записей, которые могут быть скачаны в `Recordset`. Вместо этого свойства можно использовать и `CacheSize`.
- `State` — позволяет определить, что в настоящее время происходит с `Recordset`. Используется одно из 5 значений: открыт, закрыт, соединяется, выполняет команду на источнике или получает оттуда данные.
- `Status` — позволяет определить результат последней операции обновления данных.

Методы объекта `Recordset` представлены в следующем списке.

- `Cancel()` — позволяет прервать открытие `Recordset` (например, если оно затянулось).
- `CancelBatch()` и `CancelUpdate()` — позволяют отменить внесенные в `Recordset` изменения (до вызова команды `Update()`) в разных режимах.
- `Clone()` — позволяет скопировать объект `Recordset` в другой объект `Recordset` со всеми закладками (обычно используется, когда нужно иметь более чем одну текущую запись).
- `Close()` — позволяет освободить память, занимаемую данными `Recordset` (но не удаляет сам объект). В случае необходимости вы можете опять вызвать метод `Open()`, чтобы воссоздать этот объект с ранее определенными параметрами (значения свойств объекта `Recordset` при вызове метода `Close()` сохраняются).
- `CompareBookmarks()` — позволяет сравнить две закладки и вернуть результат сравнения (указывают на одну и ту же запись, или первая запись выше, или первая запись ниже и т. п.).
- `GetRows()` — позволяет вернуть из `Recordset` двумерный массив типов `Variant`. В качестве необязательных параметров принимает стартовую позицию, количество строк, которые нужно поместить в массив, и те столбцы, которые нужно извлечь из `Recordset`. В нашем примере использование этого метода может выглядеть так:

```
Dim arr As Variant   'объявляем переменную для создаваемого массива
arr = rs.GetRows
Debug.Print arr(2, 3)
```

- `GetString()` — самая простая возможность получить из объекта `Recordset` строковое значение. По умолчанию разделителями между столбцами принимаются символы табуляции, между записями — перевод каретки. Можно использовать в отладочных целях, чтобы посмотреть на `Recordset`:

  ```
  Debug.Print rs.GetString
  ```

- `NextRecordSet()` — позволяет очистить текущий `Recordset` и выполнить следующую команду, указанную в методе `Open()`, если команды были указаны в формате:

  ```
  "select * from dbo.customers; select * from dbo.employees"
  ```

  Обычно такой подход используется для обработки наборов однотипных таблиц.

- `Requery()` — повторно выполняет запрос, который использовался для метода `Open()`, и заново заполняет `Recordset`.

- `Resync()` — обновляет значения уже полученных записей, скачав их заново с источника. Новые записи при этом видны не будут (в отличие от метода `Requery()`).

- `Save()` — сохраняет `Recordset` в файле на диске. Можно использовать формат ADTG (Microsoft Advanced Data TableGram), XML или родной формат провайдера. Например:

  ```
  rs.Save "C:\rscustomers.xml", adPersistXML
  ```

  При необходимости можно восстановить сохраненный `Recordset` из файла, указав соответствующие параметры методу `Open()`.

- `SetAllRowsStatus()` — позволяет изменить значение свойства `Status` для всех строк `Recordset`.

- `Supports()` — выполняет проверку того, что поддерживает данный `Recordset` (в каких направлениях можно двигаться, поддерживается ли поиск, закладки и т. п.). Те возможности, которые проверяются этим методом, определяются особенностями провайдера (т. е. драйвера для подключения к источнику).

Для объекта `Recordset` предусмотрен также набор событий (`EndOfRecordset`, `FetchComplete`, `MoveComplete`), но используются они редко, поэтому здесь рассматриваться не будут.

9.6. Объект *Command* и коллекция *Parameters*

В самых простых случаях, когда можно получать и изменять данные напрямую в таблицах, можно обойтись объектом `Recordset`. Однако во многих си-

туациях возможностей этого объекта недостаточно. Как уже говорилось ранее, предпочтительнее производить любое внесение изменений на источник данных при помощи хранимых процедур. Часто существует потребность в создании временных таблиц и других объектов на сервере. Бизнес-логика многих приложений (начисление процентов, абонентской платы, формирование специальных отчетов с вычислениями и т. п.) также реализована в виде хранимых процедур, поэтому в реальных приложениях одним объектом `Recordset` не обойтись.

Для выполнения команд SQL на сервере (в том числе для запуска хранимых процедур, команд DDL для создания объектов, выполнения служебных операций типа резервного копирования, восстановления, изменения параметров работы) необходимо использовать объект `Command`.

Создание этого объекта производится очень просто:

```
Dim cmd As ADODB.Command
Set cmd = CreateObject("ADODB.Command")
```

Следующее, что нужно сделать, — это назначить объекту `Command` объект подключения `Connection`. Для этой цели предназначено свойство `Command.ActiveConnection`. Ему можно передать готовый объект `Connection`, а можно сформировать этот объект неявно, используя в качестве значения свойства `ActiveConnection` строку подключения. Рекомендуется всегда предавать готовый объект подключения: во-первых, так для соединения можно настроить больше параметров, а во-вторых, если вы используете в приложении несколько объектов `Command`, можно использовать для каждого такого объекта одно-единственное подключение, что экономит ресурсы. В нашем примере мы используем созданный нами ранее объект `Connection`:

```
cmd.ActiveConnection = cn
```

Следующая наша задача — выбрать тип команды. В принципе, для многих источников можно этого не делать — модули ADO постараются сами выяснить у источника данных, что это за команда (хранимая процедура, SQL-запрос и т. п.), однако лучше всегда его определять: экономится время и системные ресурсы, уменьшается вероятность ошибок. Для выбора типа команды используется свойство `CommandType`. Значения, которые ему можно присвоить, аналогичны возможным значениям параметра `Options` метода `Open()` объекта `Recordset`, которое было рассмотрено в *разд. 9.5.2*. Например, если мы передаем команду на выполнение хранимой процедуры, то присвоить соответствующее значение можно так:

```
cmd.CommandType = adCmdStoredProc
```

Следующее действие — определить текст команды, которая будет выполняться. Делается это при помощи свойства `CommandText`. Например, если мы

хотим запустить на выполнение хранимую процедуру `CustOrderHist`, то соответствующий код может выглядеть так:

```
cmd.CommandText = "CustOrderHist"
```

Чаще всего хранимая процедура требует передачи ей одного или нескольких параметров. Делается это при помощи коллекции `Parameters` и объектов `Parameter`. Для определения параметров можно использовать два способа:

- вначале создать объекты `Parameter` автоматически путем запроса к серверу (используется метод `Refresh()` коллекции `Parameters`), а затем присвоить им значения:

```
cmd.Parameters.Refresh
cmd.Parameters(1) = "ALFKI"
```

- создать объекты `Parameter` вручную и вручную добавить их в коллекцию `Parameters`. Этот способ более экономичен (нет необходимости лишний раз обращаться на сервер), но требует предварительно знать точные свойства параметра и использовать код большего размера:

```
Dim Prm As ADODB.Parameter
Set Prm = cmd.CreateParameter("CustomerID", adVarWChar, _
          adParamInput, 5, "ALFKI")
cmd.Parameters.Append Prm
```

После определения параметров команду необходимо запустить на выполнение. Для этого используется метод `Execute()`. Самый простой способ его вызова выглядит так:

```
cmd.Execute
```

Этот метод принимает также три необязательных параметра, при помощи которых можно дополнительно определить параметры, тип вызываемой команды и т. п.

Некоторые хранимые процедуры и передаваемые команды не требуют возврата каких-либо значений (кроме кода ошибки), но это бывает редко. Как же принять значение, возвращаемое выполняемой командой?

Если возвращаемое значение официально зарегистрировано как возвращаемый параметр (например, оно помечено ключевым словом `OUT` в определении хранимой процедуры), то это значение будет присвоено соответствующему параметру объекта `Command`, и до него можно будет добраться обычным способом — при помощи свойства `Value`.

Если же, как в нашем примере с `CustOrderHist`, возвращаемое значение просто сбрасывается в поток вывода (в нашем случае возвращается набор записей), то можно использовать два способа:

- первый способ — использовать то, что метод `Execute()` возвращает объект `Recordset`, заполненный полученными при помощи команды записями:

```
Dim rs2 As ADODB.Recordset
Set rs2 = cmd.Execute()
Debug.Print rs2.GetString
```

- второй способ — воспользоваться тем, что метод `Open()` объекта `Recordset` может принимать в качестве параметра объект `Command` (в этом случае объект `Connection` передавать этому методу уже нельзя):

```
Dim rs2 As ADODB.Recordset
Set rs2 = CreateObject("ADODB.Recordset")
rs2.Open cmd
Debug.Print rs2.GetString
```

Рассмотрим некоторые другие свойства и методы объекта `Command`.

- `CommandStream` — позволяет вместо прямого назначения текста команды (через свойство `CommandText`) принять значение из потока ввода (например, из текстового файла или другой программы). Допустимый формат потока зависит от провайдера (драйвера для данного подключения). Использовать одновременно `CommandStream` и `CommandText` нельзя (второе свойство автоматически становится пустым).
- `CommandTimeout` — позволяет указать, сколько времени в секундах ждать результата выполнения команды на источнике, прежде чем вернуть ошибку.
- `Dialect` — это свойство позволяет указать особенности разбора *(parsing)* текста команды на провайдере.
- `NamedParameters` — определяет, будут ли передаваться провайдеру имена параметров (`True`) или будет использоваться простая передача значений по порядку (`False`, по умолчанию).
- `Prepared` — свойство, которое может влиять на производительность. Если установить его в `True` (по умолчанию `False`), то при первом выполнении команды провайдер создаст ее откомпилированную версию, которую и будет использовать при последующих выполнениях. Первый раз команда будет выполнять медленнее, чем обычно, зато в следующие разы — быстрее. Такое "приготовление команды" *(command preparation)* поддерживают далеко не все драйверы подключений.
- `State` — возвращает те же значения и используется в тех же целях, что и для объекта `Recordset`.
- `Cancel()` — этот метод позволяет прекратить выполнение команды (когда выполнение затянулось), если такую возможность поддерживает провайдер.

Задание для самостоятельной работы 9: Вставка по запросу записей из базы данных

Примечание:

В этом задании используется база данных Microsoft Access с именем Борей.mdb, которая при установке Microsoft Office 2003 по умолчанию помещается в каталог C:\Program Files\Microsoft Office\OFFICE11\SAMPLES. Перед началом этой работы рекомендуется провести поиск на диске, чтобы найти этот файл. Если он находится в другом каталоге — скорректируйте путь к этому файлу в ответе. Если вы его вообще не обнаружите, то доустановите Microsoft Office таким образом, чтобы он был установлен с полным набором компонентов.

Подготовка:

1. Создайте новый документ Word и сохраните его как C:\ InsertQueryResults.doc.
2. Щелкните правой кнопкой мыши по любой панели инструментов или меню и в открывшемся списке доступных панелей инструментов выберите **Элементы управления**.
3. Нажмите кнопку **Режим конструктора** на панели инструментов **Элементы управления** (верхняя левая кнопка) и в этом режиме поместите в документ Word новую кнопку. Для этого нужно щелкнуть по объекту **Кнопка** на панели инструментов **Элементы управления** и в документе определить местонахождение и размеры этой кнопки.
4. Щелкните по созданной вами кнопке правой кнопкой мыши и в контекстном меню выберите **Свойства**. Определите для кнопки свойства по вашему желанию. Выглядеть документ в итоге может, например, так, как показано на рис. 9.12.
5. При помощи меню **Вставка | Закладка** поместите под эту кнопку закладку с именем **Bookmark1**.
6. В режиме конструктора щелкните по кнопке правой кнопкой мыши и в контекстном меню выберите **Исходный текст**. Откроется редактор кода Visual Basic с созданной процедурой для события `Click` данной кнопки. Поместите в него следующий код:

```
Private Sub CommandButton1_Click()
    Dim nEmpId As Integer
    Dim sLastName As String
    Dim sFirstName As String
```

```
    Dim sTitle As String

    nEmpId = CInt(InputBox("Введите номер сотрудника:"))

    'Код, который нужно заменить
    sLastName = "Иванов"
    sFirstName = "Иван"
    sTitle = "Начальник"
    'Конец кода, который нужно заменить

    ThisDocument.Activate
    ThisDocument.Bookmarks("Bookmark1").Select
    Selection.TypeText CStr(nEmpId) & " " & sLastName & " " & _
        sFirstName & " " & vbTab & sTitle & vbCrLf

End Sub
```

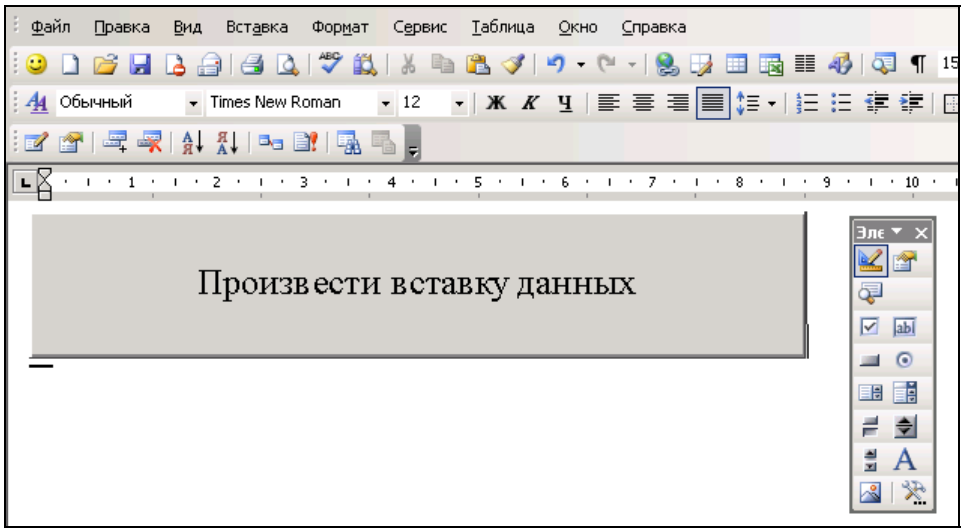


Рис. 9.12. Кнопка в документе Word

ЗАДАНИЕ:

Измените код события `Click` этой кнопки таким образом, чтобы вместо присвоения переменным явно определенных значений им присваивались значения из таблицы `Сотрудники` базы данных Борей.mdb:

- для переменной `sLastName` — значение из столбца `Фамилия`;
- для переменной `sFirstName` — значение из столбца `Имя`;
- для переменной `sTitle` — значение из столбца `Должность`.

Номер сотрудника (значение столбца `Код сотрудника`) нужной записи должен определяться пользователем при помощи функции `InputBox()`.

Ответ к заданию 9

Итоговый код для события `Click()` нашей кнопки может быть таким:

```
Private Sub CommandButton1_Click()
    Dim nEmpId As Integer
    Dim sLastName As String
    Dim sFirstName As String
    Dim sTitle As String

    'Получаем от пользователя номер сотрудника
    nEmpId = CInt(InputBox("Введите номер сотрудника:"))

    'Создаем и настраиваем объект Connection
    Dim cn As New ADODB.Connection
    'У вас путь к файлу Борей.mdb может быть другим
    cn.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;" & _
        "Data Source=C:\Program Files\Microsoft Office\OFFICE11\" _
        & "SAMPLES\Борей.mdb"
    cn.Open

    'Создаем и настраиваем объект Recordset
    Dim rs As New ADODB.Recordset
    'Обеспечиваем возможность передвижения в любом направлении
    rs.CursorType = adOpenStatic

    'Открываем Recordset на основе запроса
    rs.Open "SELECT [КодСотрудника], [Имя], [Фамилия], [Должность] " & _
        "FROM [Сотрудники] WHERE [КодСотрудника] = " & nEmpId, cn

    'Проверяем, не пустой ли Recordset
    If rs.EOF = True And rs.BOF = True Then
        MsgBox "Сотрудник с таким номером не обнаружен"
        Exit Sub
    End If

    'Присваиваем значения из найденной записи в таблице
    sLastName = rs.Fields("Фамилия")
    sFirstName = rs.Fields("Имя")
    sTitle = rs.Fields("Должность")

    ThisDocument.Activate
    ThisDocument.Bookmarks("Bookmark1").Select
    Selection.TypeText nEmpId & " " & sLastName & " " & sFirstName & _
        " " & vbTab & sTitle & vbCrLf

End Sub
```