18

Введение в технологию ADO

Основным методом доступа к разделяемым файлам и базам данных клиент/сервер становятся технология ActiveX Data Objects (ADO) и OLE DB, пришедшие на смену прикладному интерфейсу программирования Open Database Connectivity (ODBC). Технология ADO является еще одной моделью доступа к базам данных и представляет собой объектно-ориентированный интерфейс для технологии доступа к данным OLE DB. ADO поддерживает ключевые возможности для построения клиент/серверных и Web-приложений, а также обеспечивает функции Remote Data Service (RDS), посредством которого можно перемещать данные с сервера в клиентское приложение или на Web-страницу, манипулировать данными «на стороне клиента» и возвращать обновленные данные серверу.

Технология OLE DB является стратегической технологией компании Microsoft для доступа к данным. Технология предназначена для локальных баз данных, клиент/серверных, для размещения данных в хранилищах сообщений и системах каталогов. Microsoft приняла решение заменить технологию ODBC набором компонентов, классифицируемых как провайдеры[1] (поставщики) данных OLE DB, провайдеры (поставщики) служб и потребители данных.

Провайдеры данных OLE DB возвращают основанные на объектной модели COM (Component Object Model) наборы строк из табличных и иерархических источников данных. Объект ADO является автоматическим конвертером объектов OLE DB и преобразует наборы строк **`Rowset`** в наборы записей **`ADO.Recordsets`**.

Основными поставщиками данных OLE DB являются:

- *Microsoft OLE Provider for ODBC Drivers* — провайдер, заменяющий диспетчер драверов ODBC и позволяющий устанавливать соединение с базами даных, не имеющими собственного провайдера OLE DB.
- *Microsoft Jet 4.0 OLE DB Provider* — собственный провайдер OLE DB для баз данных Let 4.0 и более ранних версий. Формат баз данных Jet 4.0 используется в Microsoft Access 9 и 10.
- *Microsoft OLE Provider for SQL Server* — собственный провайдер OLE DB для SQL Server 6.x и более поздних версий.
- *Microsoft OLE DB Provider for Oracle* — предназначен для извлечения информации из баз данных Oracle (несмотря на свое название, он был разработан корпорацией Microsoft).
- *Microsoft OLE DB Provider for Microsoft Indexing Service* — предоставляет доступ в режиме «только для чтения» к файловой системе и Web-контенту, проиндексированным с помощью службы Microsoft Indexing Service. С его помощью можно осуществлять поиск по конкретным словам и фразам с использованием шаблонов и логических операторов. Кроме того, доступен режим свободного текста, когда объектом поиска является скорее смысл фразы, чем простое совпадение составляющих ее слов.
- *Microsoft OLE DB Provider for Internet Publishing* — используется для подключения к Microsoft Internet Information Server (IIS), а также серверным расширениям Microsoft FrontPage.
- *Microsoft OLE DB Provider for Microsoft Active Directory Service* — можно извлекать данные в режиме только для чтения из каталогов Windows NT 4 и Novell Directory Services, а также любых служб каталогов, совместимых с протоколом LDAP.

Список поставщиков OLE DB не ограничивается перечисленными. Например, при установке Microsoft SQL Server могут быть дополнительно установлены Microsoft OLE DB Provider for DTS Packages, Microsoft OLE DB Provider for OLAP Services и другие.

В настоящее время Microsoft поддерживаются следующие поставщики служб:

[1] Термин, к которому можно привыкать постепенно, сначала подразумевая под ним «драйвер».

- Microsoft Data Shaping Service for OLE DB
- Microsoft OLE DB Persistence Provider
- Microsoft OLE DB Remoting Provider
- Microsoft Cursor Service for OLE DB

Служба **Microsoft Data Shaping Service for OLE DB** поддерживает структурированные (иерархические) наборы записей. Первичная информация, извлеченная поставщиком данных, преобразуется ей во вложенную структуру, основанную на родительских и дочерних наборах записей, которые находятся в отношении подчинения (эта процедура получила название *формирования*).

Поставщик службы **Microsoft OLE DB Persistence Provider** позволяет ADO сохранять наборы записей в дисковых файлах. В них же сохраняется информация о месте, откуда поступили исходные данные. Например, вы можете создать набор записей, отредактировать его и затем сохранить на диске. Впоследствии все внесенные в него изменения могут быть зафиксированы в источнике данных.

Поставщик службы **Microsoft OLE DB Remoting Provider** позволяет обращаться к удаленным источникам данных (например, расположенным на серверах Internet) так, как если бы они были расположены локально. В настоящее время эта служба считается устаревшей. Для доступа к удаленным источникам данных Microsoft рекомендует использовать протокол SOAP — открытый стандарт, базирующийся на XML.

Служба **Microsoft Cursor Service for OLE DB** предназначена для расширенной поддержки курсоров, используемых многими поставщиками. Ее запуск происходит автоматически при создании курсора на стороне клиента.

Объектная модель ADO

На рис. 18. 1 представлена немного необычная (неиерархическая) модель объектов ADO.

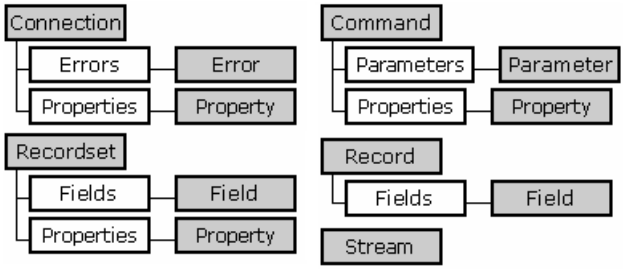

Рис. 18.1
Объекты и коллекции модели ADO

Как видно из модели ADO, вместо иерархии объектов здесь имеется независимый набор объектов (и коллекций). Причем объект **`Error`** и коллекция **`Errors`**, объекты **`Recordset`** и **`Fields`** (и их коллекции) имеют аналоги и одно и то же назначение в моделях DAO и ADO. Объект ADO Connection соответствует объекту DAO **`Workspace`**, а объект ADO **`Command`** подобен объекту DAO **`QueryDef`**. Однако свойства и методы объектов ADO явно отличаются от свойств и методов соотвествующих объектов модели DAO, хотя рассмотренные при изучении модели DAO понятия очень пригодятся при освоении модели ADO.

Объект Connection

Объект **`Connection`** (рис. 18.2) — основной объект ADO верхнего уровня. Именно с него начинается подключение к источнику данных, после чего можно использовать связанные с соединением объекты **`Command`** или **`Recordset`**.

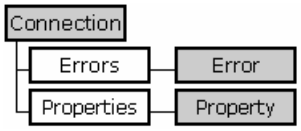

Рис. 18.2
Объекты и коллекции модели ADO

Объект **Connection** имеет следующие свойства:

Attributes	Определяет использование транзакций и может быть суммой XactAttributeEnum-значений: **adXactAbortRetaining** (новая транзакция запускается при вызове метода **RollbackTrans**), **adXactCommitRetaining** (новая транзакция запускается при вызове метода **CommitTrans**). Значение по умолчанию равно 0 — поддерживающие транзакции не используются.
CommandTimeout	Определяет время (в секундах), через которое прекращается неуспешное выполнение метода **Execute** соответствующего объекта **Command** (тип **Long**). Значение по умолчанию — 30.
ConnectionString	Строка с информацией, необходимой для провайдера, чтобы открыть соединение с источником данных.
ConnectionTimeout	Определяет время (в секундах), через которое прекращается неуспешная попытка установки соединения (тип **Long**).
CursorLocation	Определяет используемый механизм курсора (элемент базы данных, определяющий «перемещение» по записям) и может быть одним из CursorLocationEnum-значений: **adUseClient** [курсор на стороне клиента], **adUseNone** [не использовать механизм курсора], **adUseServer** [курсор на стороне сервера, значение по умолчанию] (тип **Long**).
DefaultDatabase	Определяет имя базы данных по умолчанию для объекта **Connection**, если оно не было указано в **ConnectionString** (тип **String**).
IsolationLevel	Определяет поведение транзакций, которые взаимодействуют с другими одновременно выполняющимися транзакциями, и может быть одним из IsolationLevelEnum-значений: **adXactUnspecified** [провайдер использует различные уровни транзакции, которые нельзя определить], **adXactChaos** [транзакция не будет переписывать изменения, выполненные транзакциями более высокого уровня изоляции], **adXactBrowse** [разрешается чтение незафиксированных изменений в других транзакциях], **adXactReadUncommitted** [то же назначение, что и у **adXactBrowse**], **adXactCursorStability** [разрешается чтение только зафиксированных изменений в других транзакциях], **adXactReadCommitted** [то же назначение, что и у **adXactCursorStability**], **adXactRepeatableRead** [разрешается чтение изменений в других транзакциях], **adXactIsolated** [все транзакции не зависят от других (изолированных) транзакций], **adXactSerializable** [то же назначение, что и у **adXactIsolated**] (тип **Long**).
Mode	Определяет режим доступа на чтение и запись для объектов **Connection**, **Record** или **Stream**[2] и может быть одним из ConnectModeEnum-значений: **adModeRead** [режим с разрешением только чтения], **adModeReadWrite** [режим с разрешением чтения и записи], **adModeRecursive**, **adModeShareDenyNone** [раз-

[2] Представляет поток двоичных или текстовых данных.

	решает другим пользователям открывать соединение с любым режимом доступа], **adModeShareDenyRead** [запрещает чтение данных другими пользователями], **adModeShareDenyWrite** [запрещает другим пользователям устанавливать соединение с доступом на запись], **adModeShareExclusive** [режим монопольного использования], **adModeUnknown** [режим по умолчанию; указывает на то, что не установлены никакие разрешения на соединение], **adModeWrite** [режим с разрешением только записи] (тип **Long**).
Provider	Строка, указывающая имя провайдера (если он не задан в строке **ConnectionString**). По умолчанию используется значение **MSDASQL** — Microsoft OLE DB Provider for ODBC.
State	Указывает состояние соединения: открыто или закрыто. Возвращает значение (типа **Long**) и может быть одним из ObjectStateEnum-значений: **adStateClosed** [объект закрыт — значение по умолчанию], **adStateOpen** [объект открыт], **adStateConnecting** [объект соединяется], **adStateExecuting** [вызван метод **Execute** объекта **Connection** или **Command**], **adStateFetching** [в объект **Recordset** возвращаются строки].
Version	Указывает номер версии объектной модели ADO (тип **String**).

Объект **Connection** имеет следующие методы:

BeginTrans	Запускает транзакцию.
Close	Закрывает открытый объект и любые зависимые объекты.
CommitTrans	Завершает транзакцию, закрепляя изменения в источнике данных. Разрешает запуск новой транзакции.
Execute	Выполняет определенный запрос: SQL-инструкцию, хранимую процедуру или текст, определяемый провайдером.
Open	Открывает соединение с источником данных.
OpenSchema	Возвращает информацию о схеме базы данных.
RollbackTrans	Отменяет любые изменения, выполненные во время транзакции, и завершает транзакцию (отменяет транзакцию). Разрешает запуск новой транзакции.

Одним из наиболее полезных методов (после **Open**, конечно) объекта **Connection** является метод **Execute**, имеющий следующий синтаксис:

Синтаксис

невозвращающий набор:
```
connection.Execute CommandText [, RecordsAffected]
```

возвращающий набор (объект **Recordset**):
```
Set recordset = connection.Execute (CommandText [, RecordsAffected])
```

Аргумент *CommandText* — строка, содержащая SQL-инструкцию, имя таблицы, хранимой процедуры, URL или текст, определяющий провайдера. Необязательный аргумент *RecordsAffected* — переменная типа **Long**, в которую провайдер возвращает число записей, принимающих «участие» в операции.

Пример использования метода **Execute** приведен далее при рассмотрении объекта **Recordset**, поскольку именно при помощи этого объекта можно легко увидеть результат выполнения метода **Execute**.

Поскольку технология ADO представляет собой объектно-ориентированный интерфейс, объект **Connection** имеет события, которые полезны для «перехвата» ошибок, при выполнении асинхронных операций с базами данных и в других случаях:

BeginTransComplete	Инициализируется после завершения операции **Connection.BeginTrans**.
CommitTransComplete	Инициализируется после завершения операции **Connection.CommitTrans**.
RollbackTransComplete	Инициализируется после завершения операции **Connection.RollbackTrans**.
ConnectComplete	Инициализируется после установки соединения (начало соединения).
Disconnect	Инициализируется после операции разъединения.
ExecuteComplete	Инициализируется после выполнения команды.
InfoMessage	Инициализируется при появлении предупреждения во время выполнения операции **ConnectionEvent**.
WillConnect	Инициализируется перед началом соединения.
WillExecute	Инициализируется непосредственно перед тем, как ожидающая (выполнения для соединения) команда выполняется.

Использование Microsoft OLE DB-провайдеров для подключения к источникам данных

Для подключения к источнику данных можно использовать метод **Open** объекта **Connection** библиотеки ADO. При этом необходимо посредством свойства **ConnectionString** предоставить информацию об источнике в виде строки подключения, подобной строке подключения ODBC. Провайдера данных можно указать также в свойстве **Provider**. Это должна быть строка с типом провайдера OLE DB для подключения. При использовании провайдера ODBC для OLE DB его можно вообще не указывать, так как он используется по умолчанию. Однако хорошим «тоном» считается указывать провайдера явно. Далее приведены особенности работы с некоторыми провайдерами. Часть информации следующих разделов вначале может быть не совсем понятной, и при первом чтении можно ее пропустить.

OLE DB Provider for Microsoft Jet

Для подключения к базам данных Microsoft Jet следует использовать OLE DB Provider for Microsoft Jet. Чтобы соединиться с этим провайдером, необходимо аргументу *Provider* свойства **ConnectionString** (связывающая строка) объекта **ADODB.Connection** присвоить значение (при чтении свойства **Provider** объекта **ADODB.Connection** возвращается эта же строка):
Microsoft.Jet.OLEDB.4.0

Обычно свойство **ConnectionString** содержит строку следующего вида:
“Provider=Microsoft.Jet.OLEDB.4.0;Data Source=*DatabaseName*;User ID=*UserName*;Password=*UserPassword*”

Значениями ключевых слов являются:

- **Provider** — OLE DB Provider for Microsoft Jet;
- **Data Source** — полное (имя драйвера, путь, имя файла) имя файла с базой данных (например, **c:\master\mdb\firms.mdb**);
- **User ID** — имя пользователя; при отсутствии этого аргумента используется строка **“admin”**[3];

[3] При установке Access всегда создается стандартная рабочая группа, содержащая один встроенный код пользователя и два встроенных кода групп. Код пользователя — “admin”, а его

- **Password** — пароль пользователя, при отсутствии этого аргумента используется пустая строка “ ”.

Строку **“Microsoft.Jet.OLEDB.4.0”** можно присваивать также свойству **Provider** объекта **ADODB.Connection**, тогда свойству следует присваивать полное имя файла с базой данных. Например:

```
Set cn = New ADODB.Connection
cn.Provider = "Microsoft.Jet.OLEDB.4.0"
cn.ConnectionString = "f:\фирма.mdb"
```

OLE DB Provider for Microsoft Jet поддерживает несколько специфических для провайдера динамических свойств в дополнение к свойствам, определяемым моделью ADO. Эти дополнительные свойства[4] относятся как к объекту **Connection** (могут устанавливаться посредством коллекции **Properties** объекта **Connection** или как часть связывающей строки), так и к объектам **Recordset** и **Command** (доступны и устанавливаются посредством коллекций **Properties** объектов **Recordset** и **Command**).

По умолчанию OLE DB Provider for Microsoft Jet открывает базы данных Microsoft Jet в режиме «чтения/записи». Используя свойство **Mode** объекта **Connection**, можно изменить режим открытия базы. Например, на рис. 18.3 показан фрагмент кода и всплывающий список VB-редактора с **Mode**-константами.

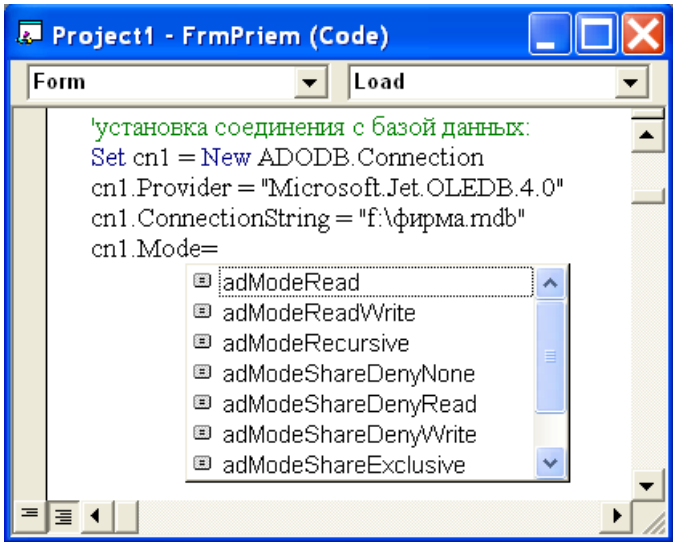

Рис. 18.3
Фрагмент кода и всплывающий список VB-редактора с **Mode**-константами

При подключении к базе данных посредством OLE DB Provider for Microsoft Jet текст в свойстве **CommandText** объекта **Command.CommandText** (рассматривается далее в книге) должен соответствовать диалекту Microsoft Jet SQL (можно определять наименования таблиц и запросы, возвращающие наборы, запросы-действия, но нельзя описывать хранимые процедуры).

О других особенностях этого провайдера можно узнать из справочной VB- или VBA-системы в разделе **OLE DB Provider for Microsoft Jet**.

Microsoft OLE DB Provider for ODBC

В настоящее время, практически, все базы данных доступны посредством ODBC. ODBC-драйверы имеются для всех основных СУБД, включая Microsoft SQL Server, Microsoft Access,

---

пароль — “”. Access автоматически предоставляет этому пользователю все права, и при создании базы данных он становится владельцем этой базы. Поскольку большинство пользователей Access «не включают» защиту и загружают Access как admin-пользователи, они могут «смело» обмениваться своими mdb-файлами, а из VB- или VBA-кода при помощи OLE DB Provider for Microsoft Jet к этим файлам можно обращаться, используя строку ‘Provider=Microsoft.Jet.OLEDB.4.0;Data Source=*DatabaseName*; User ID=””;Password=”” ’ в качестве свойства **ConnectionString**.

[4] Наименования и назначение свойств можно найти в документации по ADO.

FofPro и Oracle. Microsoft OLE DB Provider for ODBC позволяет соединяться с любым ODBC-источником данных. Для ADO этот провайдер является провайдером по умолчанию.

Чтобы соединиться с этим провайдером, необходимо аргументу *Provider* свойства **ConnectionString** (связывающая строка или строка подключения) объекта **ADODB.Connection** присвоить значение (при чтении свойства **Provider** объекта **ADODB.Connection** возвращается эта же строка):
MSDASQL

Обычно свойство **ConnectionString** содержит строку следующего вида:
"Provider=MSDASQL;DSN=*dsnName*;UID=*UserName*;PWD=*UserPassword*"

Значениями ключевых слов являются:

- **Provider** — Microsoft OLE DB Provider for ODBC;
- **Data Source** — имя источника данных;
- **User ID** — имя пользователя;
- **Password** — пароль пользователя.

Поскольку Microsoft OLE DB Provider for ODBC для ADO — провайдер по умолчанию, параметр **Provider** можно опускать.

При использовании Microsoft OLE DB Provider for ODBC можно соединяться посредством предопределенного источника данных (DSN или FileDSN) или без него. Для этого необходимо использовать следующий синтаксис строки подключения:

синтаксис с DSN (или FileDSN):
"[Provider=MSDASQL;] {DSN=*Name* | FileDSN=*FileName*} ; [DATABASE=*DatabaseName*;] UID=*UserName*; PWD=*UserPassword*"

синтаксис без DSN:
"[Provider=MSDASQL;] DRIVER=*DriverName*; SERVER=*ServerName* DATABASE=*DatabaseName*; UID= *UserName*; PWD= *UserPassword*"

Несмотря на то, что с конкретным DSN всегда связана определенная база данных, вы можете переопределить ее, используя аргумент **DATABASE** (см. синтаксис с DSN). Таким образом, подключение будет происходить именно с тем файлом, который указан в связывающей строке, а не с тем, который когда-то по уже забытой причине был выбран при работе с программой ODBC Driver Manager.

Синтаксис строки подключения без DSN позволяет посредством этого драйвера подключиться к базе данных, находящейся, например на SQL Server (параметр **DRIVER** для этого — "**SQL Server"**).

Далее в этой главе приводятся примеры использования строк подключения и с DSN, и без DSN.

О других особенностях этого провайдера можно узнать из справочной VB- или VBA-системы в разделе **Microsoft OLE DB Provider for ODBC**.

Microsoft OLE DB Provider for SQL Server

Microsoft OLE DB Provider for SQL Server позволяет посредством ADO обращаться к Microsoft SQL Server, работа с которым рассматривается в части III этой книги. Чтобы соединиться с этим провайдером, необходимо аргументу *Provider* свойства **ConnectionString** объекта **ADODB.Connection** присвоить значение:
SQLOLEDB

Строка подключения для этого провайдера обычно имеет следующий вид:
"Provider=SQLOLEDB; Data Source=*ServerName*; Initial Catalog=*DatabaseName*; User ID=*UserName*; PWD=*UserPassword*"

Значениями ключевых слов являются:

- **Provider** — Microsoft OLE DB Provider for SQL Server;
- **Data Source** (можно также **Server**) — имя сервера;
- **Initial Catalog** (можно также **Database**) — имя базы данных на сервере;
- **User ID** (можно также **uid**) — имя пользователя (для сервера);

- **Password** (можно также **pwd**) — пароль пользователя (для сервера).

Этот провайдер поддерживает несколько специфических параметров соединения в дополнение к параметрам, определяемым ADO. Эти свойства (как и ADO-свойства соединения) могут устанавливаться посредством коллекции **Properties** объекта **Connection** или как часть строки **ConnectionString** и приведены в следующей таблице.

Параметр	Описание
Trusted_Connection	Указывает режим аутентификации пользователя. Может устанавливаться в **Yes** или **No**. Значением по умолчанию является **No**. Если это свойство устанавливается в **Yes**, то Microsoft OLE DB Provider for SQL Server использует режим Microsoft Windows NT Authentication для санкционирования пользовательского доступа к базе данных SQL Server, определяемой значениями свойств **Location** и **Datasource**. Если это свойство установлено в **No**, Microsoft OLE DB Provider for SQL Server использует режим **Mixed** для санкционирования пользовательского доступа к базе данных SQL Server. Имя и пароль для SQL Server определяются в свойствах **User Id** и **Password**.
Current Language	Указывает язык, используемый для выбора и форматирования сообщений системы. Язык должен быть установлен на SQL Server, в противном случае открытие соединения будет неуспешным.
Network Address	Указывает сетевой адрес SQL Server, определяемый свойством **Location**.
Network Library	Указывает имя сетевой библиотеки (DLL), используемой для коммуникации с SQL Server. Имя не должно включать путь или расширение имени dll-файла. Имя по умолчанию предоставляется клиентской конфигурацией SQL Server.
Use Procedure for Prepare	Определяет, создает ли SQL Server временные хранимые процедуры, когда команды подготавливаются (свойством **Prepared**).
Auto Translate	Указывает, преобразуются ли OEM/ANSI-символы. Это свойство может быть установлено в **True** или **False**. Значением по умолчанию является **True**. Если это свойство установлено в **True**, Microsoft OLE DB Provider for SQL Server выполняет преобразование OEM/ANSI-символов, когда многобайтовые символьные строки (multi-byte character strings) извлекаются или пересылаются на SQL Server. Если это свойство устанавливается в **False**, Microsoft OLE DB Provider for SQL Server не выполняет преобразование OEM/ANSI-символов для многобайтовых символьных строковых данных.
Packet Size	Указывает размер сетевого пакета в байтах. Значение свойства размера пакета должно находится в диапазоне от 512 до 327617. По умолчанию размер Microsoft OLE DB Provider for SQL Server сетевого пакета павен 4096.
Application Name	Указывает имя клиентского приложения.
Workstation ID	Строка, идентифицирующая рабочую станцию.

О других особенностях этого провайдера можно узнать из справочной VB- или VBA-системы в разделе **Microsoft OLE DB Provider for SQL Server**.

Примеры подключения к базам данных из VB- и VBA-кода с использованием Microsoft OLE DB-провайдеров

В коде листинга 18.1 используется провайдер ODBC для OLE DB. Строка подключения включает только имя источника данных, указанное в диалоговом окне **ODBC Microsoft Access Setup** в текстовом поле **Data Source Name**.

Листинг 18.1 Открытие ADO-соединения с использованием ODBC-провайдера

```
1:   Dim cn  As ADODB.Connection
2:
3:
4:  Function ADODB_ConnectedODBC () As Boolean
5:  ' устанавливает соединение для ODBC
6:
7:     On Error GoTo err_not_connection
8:
9:     Set cn = New ADODB.Connection
10:     cn.Provider = "MSDASQL"
11:     cn.ConnectionString = "DSN=Тестирование ФИРМА.MDB;"
12:     cn.Open
13:
```

```
14:     ADODB_ConnectedODBC = True
15:     Exit Function
16:
17:  err_not_connection:
18:     ADODB_ConnectedODBC = False
19:
20:  End Function
21:
22:
23:  Sub Main_ADODB()
24:  ' тестирует ADODB_ConnectedODBC
25:
26:     If ADODB_ConnectedODBC () Then
27:        MsgBox "Похоже, соединение установлено..."
28:
29:        ' код, использующий установленное соединение:
30:        '. . .
31:        cn.Close          ' закрыть соединение
32:     Else
33:        MsgBox "Что-то не сложилось с соединением!"
34:
35:        ' код, который не использует соединение:
36:        '. . .
37:
38:     End If
39:
40:  End Sub
```

Функция **ADODB_ConnectedODBC** (строки 4–20) только открывает соединение и возвращает значение константы **`True`**, если не возникает ошибка при подключении к источнику данных. Поскольку ошибки при подключении к источникам данных — дело обычное, следует всегда вставлять в код операторы, реагирующие на возможные отказы, для улучшения «отношений» между пользователем и разрабатываемым для него приложением.

Процедура **Main_ADODB** (строки 23–40) вызывает функцию **ADODB_ConnectedODBC** и выполняет тот (строки 27–31) или иной код (строки 33–36), основываясь на возвращаемом функцией значении. В этой же процедуре открытое соединение закрывается методом **`Close`** (строка 31) объекта **cn** типа **`ADODB.Connection`**.

Код следующего листинга содержит функцию, которая использует ODBC-провайдер (строка 8) для соединения с базой данных, находящейся на удаленном сервере. Здесь не используется DSN.

Листинг 18.2 Функция для открытия ADO-соединения с использованием ODBC-провайдера

```
1:   Function ADODB_ConnectedODBC2() As Boolean
2:   ' устанавливает соединение для ODBC с базой данных на SQL Server
3:      Set cn = New ADODB.Connection
4:
5:      On Error GoTo err_not_connection
6:
7:      Set cn = New ADODB.Connection
8:      cn.Provider = "MSDASQL"
9:      cn.ConnectionString = _
10:       "DRIVER=SQL Server;SERVER=VOVA;DATABASE=УчетТоваров;" & _
11:       "UID=Dmitry;PWD=Dmitry"
12:     cn.Open
13:
14:     ADODB_ConnectedODBC2 = True
15:     Exit Function
16:
17:  err_not_connection:
18:     ADODB_ConnectedODBC2 = False
19:
20:  End Function
```

Код листинга 18.3 отличается от кода листингов 18.1 и 18.2 только использованием другого провайдера — Microsoft.Jet.OLEDB.4.0 (строка 11). Имя функции **ADODB_ConnectedODBC**, предназначенной для установления соединения, изменено на **ADODB_ConnectedJet**.

Листинг 18.3 Открытие ADO-соединения с использованием Jet-провайдера

```
1:  Dim cn  As ADODB.Connection
2:
3
4:  Function ADODB_ConnectedJet() As Boolean
5:  ' устанавливает соединение для Jet-провайдера
6:
7:
8:      On Error GoTo err_not_connection
9:
10:     Set cn = New ADODB.Connection
11:     cn.Provider = "Microsoft.Jet.OLEDB.4.0"
12:     cn.ConnectionString = "f:\фирма.mdb"
13:     cn.Open
14:
15:     ADODB_ConnectedJet = True
16:     Exit Function
17:
18: err_not_connection:
19:     ADODB_ConnectedJet = False
20:
21: End Function
22:
23: Sub Test_ADODB_Connected()
24: ' тестирует ADODB_Connected: устанавливает соединение и
25: ' использует данные из таблицы Товары
26:
27:     If ADODB_ConnectedJet() Then
28:         MsgBox "Похоже, соединение установлено..."
29:         ' код, использующий установленное соединение:
30:
31:         Set rs = New ADODB.Recordset
32:         rs.CursorType = adOpenDynamic
33:         rs.Source = "SELECT * FROM Товары"
34:         Set rs.ActiveConnection = cn
35:         rs.Open
36:
37:         MsgBox rs.Fields(1) & "  " & rs.Fields(1).Name & "  " & rs.Fields(1).Type
38:
39:         cn.Close    ' закрыть соединение
40:     Else
41:         MsgBox " Что-то не сложилось! "
42:     End If
43:
44: End Sub
```

Код листингов 18.1–18.3 может быть выполнен как в VBA-, так и в VB-модуле. Причем для выполнения этого кода в VBA-модуле нет необходимости в форме, поскольку никакие элементы управления не используются. Информацию из базы данных можно помещать непосредственно, например, в рабочие листы Microsoft Excel или таблицы и формы Microsoft Word для дальнейшей обработки.

Эти же коды могут выполняться и в Access. Заметим, что таким образом при в Access происходит подключение не к текущей базе данных, а к указанной в коде функции, которая осуществляет подключение.

Коллекция Errors объекта Connection

Коллекция **Errors** объекта **Connection** содержит все **Error**-объекты. Каждый **Error**-объект представляет определенную ошибку провайдера, но не ADO-ошибку. Любая операция, выполняемая ADO-объектами, может вызвать одну или несколько ошибок. При возникновении ошибок в коллекцию **Errors** объекта **Connection** может быть помещен один или более **Error**-объектов. Если ошибку генерирует другая ADO-операция, **Errors**-коллекция освобождается и новый набор **Error**-объектов помещается в **Errors**-коллекцию.

ADO-ошибки объявляются в run-time-механизме обработки исключений. Например, в Microsoft Visual Basic возникновение ошибки, относящейся к ADO, инициирует событие **onError** и появление информации об ошибке в **Err**-объекте.

Набор **Error**-объектов в коллекции **Errors** описывает все ошибки, которые возникают при выполнении одного оператора. Некоторые свойства и методы возвращают предупреждения, которые добавляются как **Error**-объекты в **Errors**-коллекцию, но не прекращают выполнение программы. Перед тем как вы вызываете **Resync**, **UpdateBatch** или **CancelBatch** методы для **Recordset**-объекта, метод **Open** для **Connection**-объекта или устанавливаете **Filter**-свойство для **Recordset**-объекта, следует использовать медот **Clear** для коллекции **Errors**. Тогда вы можете, считывая значение свойства **Count** коллекции **Errors**, проверять возвращаемые предупреждения.

Коллекция **Errors** имеет два свойства — **Count** (количество элементов в коллекции) и **Item** (определяет член коллекции по имени или номеру; члены коллекции нумеруются, начиная с нуля) — и два метода — **Clear** (удаляет все **Error**-объекты из коллекции) и **Refresh** (обновляет объекты в коллекции для отражения ее текущего состояния).

Объект Recordset

Конечной целью большинства приложений, использующих базы данных, является создание просмотр и обновление наборов записей. ADO-объект **Recordset** (рис. 18.3) предназначен, как и в модели DAO, для доступа к наборам записей в источнике данных (можно говорить «к информации, предоставляемой провайдером»).

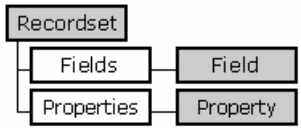

Рис. 18.3
ADO-объект **Recordset** предназначен для доступа к наборам записей в источнике данных

ADO-объект **Recordset** имеет довольно много свойств, методов и событий, которые вряд ли можно освоить одновременно и быстро. Скорее, на их изучение уйдет много времени. Те свойства и методы, которые дают понятные и самые необходимые результаты, приведены ниже.

Свойства **ADODB.Recordset**	
ActiveConnection	Указатель на открытое соединение, которому соответствует объект **Recortset**. Это очень важное свойство, которое связывает объект **ADODB.Recordset** с ранее созданным объектом **ADODB.Connection**.
BOF	При равенстве значению константы **True** означает, что указатель записи установлен перед первой записью (строкой) набора и нет текущей записи.
Bookmark	Значение (типа **Variant**), возвращающее ссылку на определенную запись.
CacheSize	Определяет число записей в локальной памяти (КЭШе) для уменьшения числа обращений к серверу (тип **Long**).
CursorType	Определяет тип курсора набора и может принимать значение одной из CursorTypeEnum-констант: **adOpenDynamic** [используется динамический курсор; отображаются добавления, изменения и удаления, выполняемые другими пользователями; все типы перемещения по записям набора доступны, за исключением перемещения на закладку, если его не поддерживает провайдер], **adOpenForwardOnly** [набор с однонаправленным перемещением курсора и записей «только для чтения»], **adOpenKeyset** [подобен динамическому курсору, но данные, добавляемые другими пользователями, скрываются; данные, удаляемые другими, недоступны, а изменяемые — видимы], **adOpenStatic** [используется статический курсор; статическая копия набора записей для нахождения данных и генерации отчетов; добавления, изменения или удаления данных другими пользователями невидимы]. По умолчанию используется значение константы **adOpenForwardOnly** (тип **Long**).
EditMode	Возвращает состояние редактирования набора и может принимать значение одной из EditModeEnum-костант: **adEditNone** [операции редактирования не выполняются] (значение по умолчанию), **adEditInProgress** [текущая

	запись редактировалась, но не сохранена], **adEditAdd** [использовался метод **AddNew**, в буфере копирования находится новая запись, которая не была сохранена в базе данных], **adEditDelete** [текущая запись была удалена] (тип **Long**).
EOF	При равенстве значению константы **True** означает то, что указатель записи установлен за последней записью набора и нет текущей записи.
LockType	Значение (типа **Long**), определяющее метод блокировки записи, используемый при открытии набора: **adLockReadOnly** [определяет доступ «только для чтения» (значение по умолчанию)], **adLockBatchOptimistic** [используется пакетный режим обновления записей вместо немедленного по умолчанию], **adLockOptimistic** [блокировка записи или страницы только на время обновления], **adLockPessimistic** [блокировка записи или страницы только на время редактирования и в процессе обновления].
RecordCount	Возвращает количество записей (тип **Long**) в объекте **Recordset**.
Sort	Строка, содержащая SQL-выражение предложения **ORDER BY** без самих зарезервированных слов и определяющая порядок сортировки записей набора.
Source	Строка, содержащая SQL-инструкцию, имя таблицы, хранимой процедуры или связанного объекта **Command**.
Методы **ADODB.Recordset**	
AddNew	Добавляет новую запись к изменяемому набору данных.
CancelBatch	Прекращает ожидание пакетного обновления.
Clone	Создает копию объекта **Recordset** с независимым указателем записи.
Close	Закрывает объект **Recordset**.
Delete	Удаляет текущую запись из набора.
Find	Ищет записи, удовлетворяющие некоторому критерию.
GetRows	Возвращает двумерный массив (строки и столбцы) записей (тип **Variant**).
GetString	Возвращает разделяемую табуляцией строку для указанного количества записей.
Move	Перемещает указатель записи с текущей записи.
MoveFirst	Перемещает указатель записи к первой записи.
MoveLast	Перемещает указатель записи к последней записи.
MoveNext	Перемещает указатель записи к следующей записи.
MovePrevious	Перемещает указатель записи к предыдущей записи
Open	Открывает набор данных активного объекта **Connection** (или **Command**).
Resync	Перезагружает (refreshes) данные объекта **Recordset** или коллекции **Fields** объекта **Record** из исходной базы.
Save	Создает файл, содержащий копию набора записей.
Seek	Ищет индекс в объекте **Recordset** для быстрой локализации строки, которая совпадает с определенным значением (эта строка становится текущей).
Supports	Определяет, поддерживает ли **Recordset**-объект указанную (в качестве аргумента) функциональность (метод).
Update	Применяет результат модификации набора записей.
UpdateBatch	Применяет результат всех модификаций набора записей пакетного типа.

Для создания набора совсем необязательно подключаться к некоторому источнику данных. Например, следующий код создает набор из двух полей и заносит в них две записи из ячеек Excel-листа **Report**, затем сохраняют этот набор в XML-файле:

```
Dim rsADO As ADODB.Recordset
Set rsADO = New ADODB.Recordset

rsADO.Fields.Append "Первое поле", adVarWChar, 15
rsADO.Fields.Append "Второе поле", adInteger
```

```
rsADO.Open

rsADO.AddNew
rsADO.Fields(0) = Trim(Worksheets("Report").Cells(2, 1))
rsADO.Fields(1) = Trim(Worksheets("Report").Cells(2, 2))
rsADO.AddNew
rsADO.Fields(0) = Trim(Worksheets("Report").Cells(3, 1))
rsADO.Fields(1) = Trim(Worksheets("Report").Cells(3, 2))

rsADO.Save “C:\Мои документы”, adPersistXML
rsADO.Close
```

Курсоры в ADO

Курсор — это способ доступа к отдельным записям набора данных. Применительно к результатам SQL-запроса понятия «следующей» и «предыдущей» записям отсутствуют. Ранее в этой книге мы не уделяли внимания этому вопросу, и у пользователя могло возникнуть впечатление, что записи сами «выстраиваются» в очередь для их использования. На самом деле, все дело — в курсорах. Более того, курсоры могут создавать различные способы доступа к записям, в том числе и обычный последовательный доступ. С помощью курсоров можно:

- получать доступ к конкретной записи набора данных;
- изменять данные, хранимые в «текущей» записи;
- задавать различные уровни «чувствительности» к изменению общих данных другими пользователями.

Технология ADO поддерживает работу с несколькими типами курсоров. Курсоры позволяют не только последовательно перемещаться в наборе записей, но и определяют способ управления набором. Некоторые типы курсоров определяют режим «только на чтение» и перемещение по набору только в одном направлении. Считается, что это — самые «быстрые» курсоры. (По крайней мере, они так задумывались их создателями.) Другие разновидности курсоров допускают перемещение по набору записей в обоих направлениях и динамическое обновление информации в наборе при изменении данных другими пользователями. Последнее свойство указывает на то, что курсоры — это нечто б*о*льшее, чем механизм, поддерживающий перемещение по набору.

Курсоры реализуются посредством специальных библиотек — части программного обеспечения базы данных, либо API, предназначенного для доступа к данным.

Как указано в предыдущей таблице, в ADO тип курсора определяется в свойстве **`CursorType`** объекта **`Recordset`** и может быть одним из следующих:

Тип курсора	CursorTypeEnum-константа	Описание
Последовательный	**`adOpenForwardOnly`**	Набор с однонаправленным перемещением курсора и записей «только для чтения». Используется по умолчанию.
Ключевой	**`adOpenKeyset`**	Подобен динамическому курсору, но данные, добавляемые другими пользователями, скрываются, хотя учитываются данные, изменяемые и удаляемые другими пользователями. Тип курсора наиболее эффективен при большом количестве записей в наборе.
Динамический	**`adOpenDynamic`**	Динамический курсор; отображаются добавления, изменения и удаления, выполняемые другими пользователями; все типы перемещения по записям набора доступны, за исключением перемещения на закладку, если его не поддерживает провайдер. Самый неэффективный.
Статический	**`adOpenStatic`**	Статический курсор; статическая копия набора записей. Особенно полезен для нахождения данных и генерации отчетов; добавления, изменения или удаления данных другими пользователями

		невидимы. Эффективен при малом количестве записей в наборе.

Выбор типа курсора определяется способом работы с набором. Например, если вам необходимо заполнить какой-либо список некоторого диалогового окна (список товаров, наименований используемых магазинов и т.д.), то для наибольшей скорости выполнения данной операции следует использовать последовательный курсор.

Тип курсора может не поддерживаться используемым провайдером. Для определения того, поддерживается ли провайдером и зависящий от выбранного курсора метод, можно использовать метод **Supports** объекта **Recordset**. Синтаксис метода **Supports** следующий:

Синтаксис

```
boolean = recordset.Supports( CursorOptions )
```

Метод возвращает **Boolean**-значение, указывающее, все ли возможности, заданные в аргументе *CursorOptions*, поддерживаются провайдером. Параметр *CursorOptions* — это выражение типа **Long**, определяющее одну или более CursorOptionEnum-констант:

Константа	Назначение
adAddNew	Поддерживается метод **AddNew** для добавления новых записей в набор.
adApproxPosition	Поддерживаются свойства **AbsolutePosition** и **AbsolutePage**.
adBookmark	В данном наборе можно использовать закладки (поддерживается свойство **Bookmark**).
adDelete	Можно удалять записи набора (поддерживается метод **Delete**).
adFind	Поддерживается метод **Find** для поиска строки в наборе.
adHoldRecords	Записи можно извлекать из базы данных без фиксации существующих изменений.
adIndex	Поддерживается свойство **Index**.
adMovePrevious	Поддерживаются методы **MoveFirst**, **MovePrevious**, **Move** и **GetRows**.
adNotify	Указывает, что исходный провайдер данных поддерживает уведомления.
adResync	Поддерживается метод **Resync** для обновления курсора с данными, которые видимы в исходной базе данных.
adSeek	Поддерживается метод **Seek** для поиска данных в объекте **Recordset**.
adUpdate	Поддерживается метод **Update** (для обновления данных).
adUpdateBatch	Поддерживается пакетное обновление (методы **UpdateBatch** и **CancelBatch**) для передачи группы изменений провайдеру.

Коллекция Field как свойство объекта Recordset

Прежде чем подробно рассмотреть некоторые свойства и методы объекта **Recordset**, следует отметить коллекцию **Fields**, которую можно рассматривать как свойство этого объекта. Именно посредством этого свойства можно из VB- или VBA-кода получить доступ к данным, хранящимся в полях записей. Коллекция **Fields** имеет одно свойство **Count** (количество полей набора **Recordset**) и два метода: **Item** (указывает определенный элемент коллекции) и **Refresh** (обновляет объекты в коллекции).

Свойства объекта **Fields** (см. справочную систему) в основном предназначены для чтения, например свойство **Name**, возвращающее имя поля в наборе, или свойство **Type**, возвращающее тип поля. Наиболее часто используется свойство **Value** (возвращающее значение поля), которое имеет статус «только для чтения» в наборах данных с последовательным доступом и наборах, открытых с блокировкой «только для чтения».

Далее рассматриваются некоторые свойства и методы объекта **Recordset**. В листинге 18.4 процедура **Test_ADODB_Connected** (из листинга 18.1) дополнена строками 10–16 для считывания информации из таблицы **Товары**. В строке 10 создается объект **rs** типа **Recordset**; в

строке 11 свойству **CursorType** объекта **rs** присваивается значение константы **adOpenDynamic**. В строке 12 определяется SQL-выражение для указания считываемых полей из источника данных (в этом случае — все поля таблицы **Товары**). В строке 13 посредством свойства **ActiveConnection** объекта **rs** назначается открытое соединение **cn**, а в строке 14 набор **rs** открывается методом **Open**.

В строке 16 используется свойство **Value** объекта **rs.Fields(1)** — значение второго по порядку поля набора **rs**. Ключевое слово **Value** здесь не указано, так как это свойство применяется по умолчанию. Свойство **rs.Fields(1).Name** в строке используется для получения наименования второго поля набора, а **rs.Fields(1).Type** — для типа поля. В строке 18 методом **Close** соединение **cn** закрывается.

Листинг 18.4 Открытие ADO-соединения и считывание информации из таблицы базы данных

```
1:   Sub Test_ADODB_Connected()
2:  ' тестирует ADODB_ConnectedODBC: устанавливает соединение и
3:  ' использует данные из таблицы Товары
4:
5:
6:      If ADODB_ConnectedODBC() Then
7:          MsgBox "Похоже, соединение установлено..."
8:          'код, использующий установленное соединение:
9:
10:         Set rs = New ADODB.Recordset
11:         rs.CursorType = adOpenDynamic
12:         rs.Source = "SELECT * FROM Товары"
13:         Set rs.ActiveConnection = cn
14:         rs.Open
15:
16:         MsgBox rs.Fields(1) & "  " & rs.Fields(1).Name & "  " & rs.Fields(1).Type
17:
18:         cn.Close    ' закрыть соединение
19:     Else
20:         MsgBox "Что-то не сложилось!"
21:     End If
22:
23:  End Sub
```

Для формы, приводимой в главе «Доступ к базам данных из VB-кода. Microsoft DAO» (рис. 18.4), можно написать код, представленный в листинге 18.5. Процедура **CmdLoad_Click** для установления соединения при помощи ADO и провайдера для ODBC[5] использует функцию **ADODB_ConnectedODBC**.

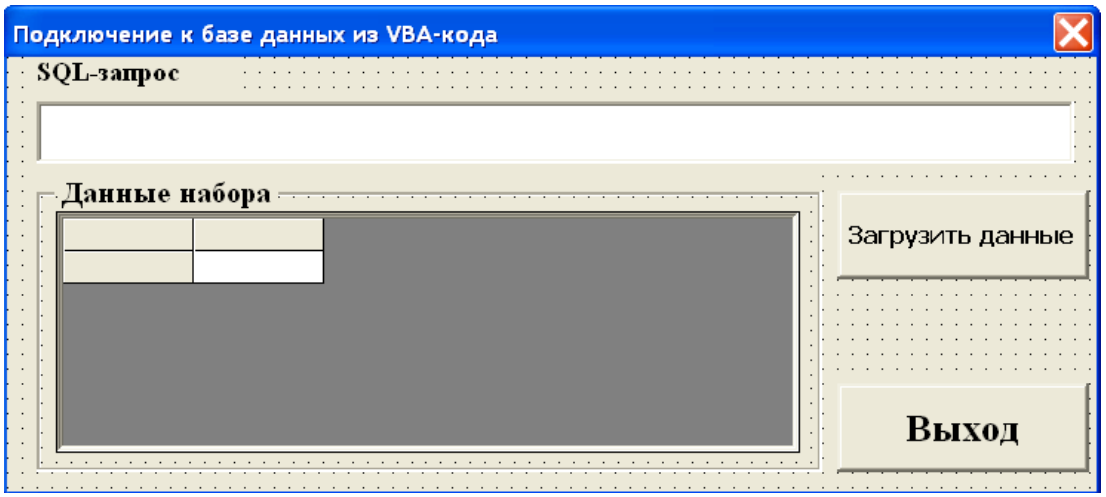

Рис. 18.4
Форма для тестирования ADO-соединения

[5] В следующей части книги будет использоваться провайдер для SQL Server.

Листинг 18.5 Открытие ADO-соединения и считывание информации в элемент управления MSFlexGrid

```
1:    Dim cn  As ADODB.Connection
2:
3:
4:   Function ADODB_ConnectedODBC() As Boolean
5:   ' устанавливает соединение для ODBC
6:
7:
8:       On Error GoTo err_not_connection
9:
10:      Set cn = New ADODB.Connection
11:      cn.Provider = "MSDASQL"
12:      cn.ConnectionString = "DSN=Тестирование ФИРМА.MDB;"
13:      cn.Open
14:
15:      ADODB_ConnectedODBC = True
16:      Exit Function
17:
18:  err_not_connection:
19:      ADODB_ConnectedODBC = False
20:
21:  End Function
22:
23:
24:  Private Sub CmdExit_Click()
25:      Unload Me
26:  End Sub
27:
28:
29:  Private Sub CmdLoad_Click()
30:
31:      Dim rstNwind As ADODB.Recordset
32:      Dim newElement As String, i As Integer
33:
34:      If Not ADODB_ConnectedODBC() Then
35:          MsgBox " Невозможно подключиться к базе данных! "
36:          Exit Sub
37:      End If
38:
39:
40:      Set rstNwind = New ADODB.Recordset
41:
42:      'открыть набор:
43:      rstNwind.Open TextBox1.Text, cn
44:
45:      rstNwind.MoveFirst      'перейти к началу набора
46:
47:      'задать количество строк:
48:      MSFlexGrid1.Cols = rstNwind.Fields.Count + 1
49:
50:      'установить ширину первых колонок
51:      MSFlexGrid1.ColWidth(0) = 1
52:      If MSFlexGrid1.Cols > 1 Then MSFlexGrid1.ColWidth(1) = 1500
53:      If MSFlexGrid1.Cols > 2 Then MSFlexGrid1.ColWidth(2) = 2000
54:
55:      'до конца набора добавлять записи в список
56:      Do While Not rstNwind.EOF
57:
58:          'сформировать элемент добавления:
59:          newElement = ""
60:          For i = 0 To rstNwind.Fields.Count - 1
61:              newElement = newElement & vbTab & rstNwind.Fields(i)
62:          Next
63:
64:          'добавить элемент к таблице:
65:          MSFlexGrid1.AddItem newElement
66:
67:          rstNwind.MoveNext
68:      Loop
69:
```

```
70:  End Sub
```

В отличие от предыдущего примера здесь имеется возможность в качестве свойства **Source** указать строку, вводимую во время работы приложения в текстовое окно. Как и ранее, в это текстовое окно можно ввести SQL-инструкцию для выбора данных из нескольких таблиц для определенных записей.

В строке 43 записан оператор для открытия набора **rstNwind**:

```
43:      rstNwind.Open TextBox1.Text, cn
```

Ранее вместо этого использовались операторы, подобные следующим:

```
rstNwind.Source = TextBox1.Text
Set rstNwind.ActiveConnection = cn
rstNwind.Open
```

Для примера использования метода **Execute** объекта **Connection** добавим к форме, которая приведена на рис. 18.3, кнопку с наименованием «Выполнить инструкцию» (рис. 18.4).

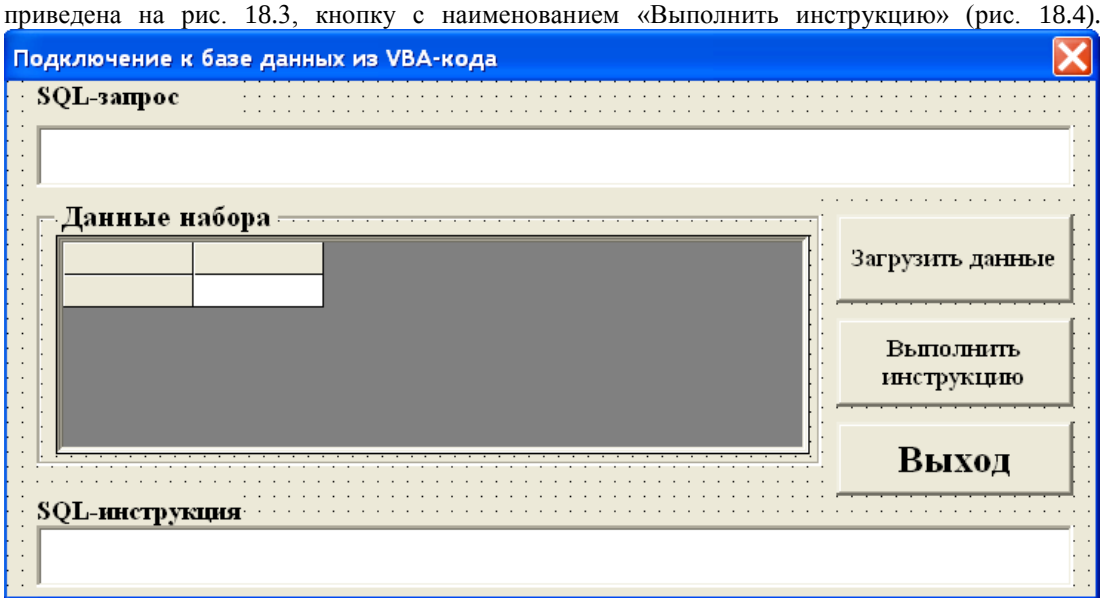

Рис. 18.4
Форма для тестирования приложения с использованием метода **Execute** объекта **Connection** в режиме разработки

Событийная процедура кнопки **Выполнить инструкцию** содержит всего один оператор (кроме заголовка и оператора окончания процедуры):

```
Private Sub CommandButton1_Click()
   cn.Execute TextBox2.Text
End Sub
```

Таким образом, эта процедура использует содержимое текстового окна **TextBox2** в качестве аргумента *CommandText* метода **Execute** и выполняет этот метод для соединения **cn**. Конечно, необходимо предусмотреть обработчик ошибок в этой процедуре, так как пользователь может ввести неверную инструкцию, но сейчас это не очень важно.

Для тестирования приложения следует выполнить:

- запустить приложение;
- в текстовом окне SQL-запрос ввести строку “**Товары”**;
- щелкнуть кнопку **Загрузить список**;
- ввести в текстовом окне SQL-инструкцию
  INSERT INTO Товары (КодТовара, НаимТовара) VALUES ('1','Совсем новый товар')
- щелкнуть кнопку **Выполнить инструкцию**;
- щелкнуть кнопку **Загрузить данные**;

Если все сделать так, как указано, то при следующей загрузке (щелчок на кнопке **Загрузить данные**) в окне списка **Список товаров** можно будет увидеть новый товар (рис. 18.5).

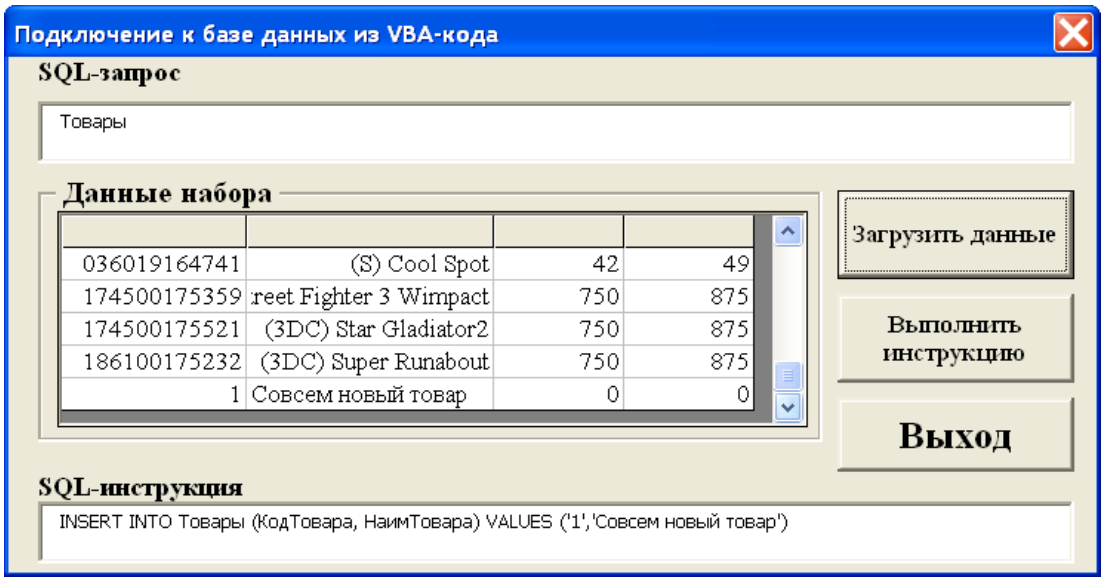

Рис. 18.5
Форма для тестирования приложения с использованием метода **Execute** объекта **Connection** в режиме выполнения

Далее можно вводить любые релевантные SQL-инструкции, например

```
DELETE FROM Товары WHERE КодТовара='1'
```

Эта инструкция удалит запись, которая содержит в поле **КодТовара** значение “1”.

Добавлять записи в набор можно не только SQL-инструкцией для метода **`Connection.Execute`**. Это позволяет сделать метод **`AddNew`** объкта **`Recordset`**, который имеет следующий синтаксис:

Синтаксис

```
recordset.AddNew [FieldList][, Values]
```

Необязательный аргумент *FieldList* — одно имя, массив имен или порядковых номеров полей в новой записи. Необязательный аргумент *Values* — одно значение или массив для полей в новой записи. Если *Fieldlist* — массив, *Values* должен быть тоже массивом с тем же самым количеством элементов. Обычно аргументы этого метода не используются. Чаще всего метод применяется для добавления пустой записи, а затем запись редактируется, как в следующем примере.

Изменим форму, приведенную на рис. 18.5: вместо текстового окна и кнопки в нижней части формы поместим на форму текстовые окна и кнопку для ввода данных, как показано на рис. 18.6.

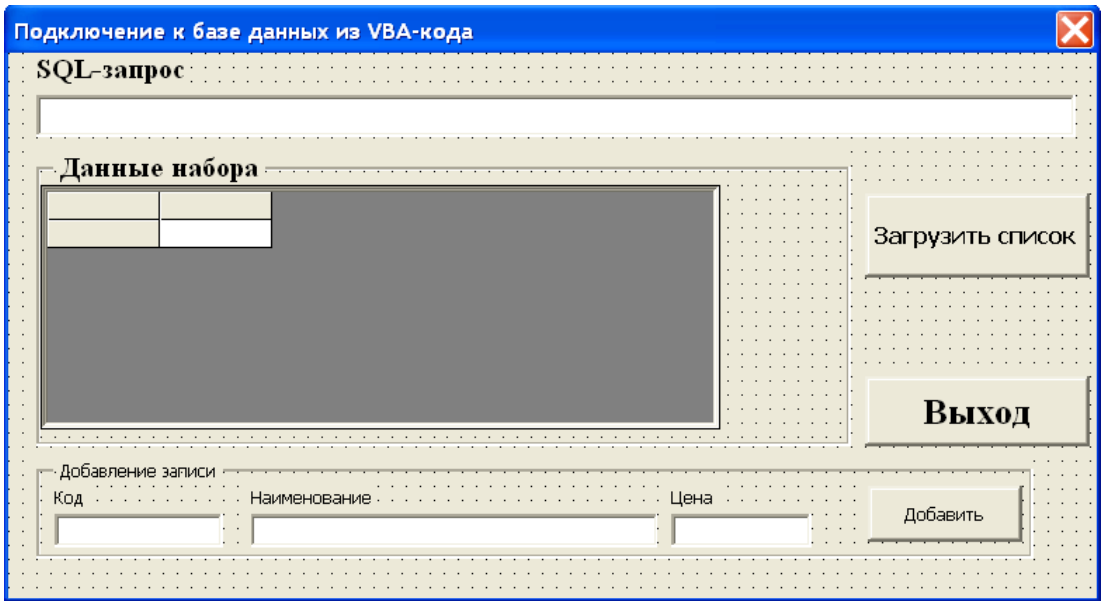

Рис. 18.6
Форма для тестирования приложения с использованием метода **AddNew** объекта **Recordset** в режиме разработки

При этом таблица со свойствами элементов управления формы будет иметь следующий вид:

Тип элемента	Свойство, которое изменено (используется в коде)	Значение	Примечание
UserForm	Name	UserForm1	Имя главной формы, на которое можно ссылаться в коде.
Frame	Name	Frame1	Имя, на которое можно ссылаться в коде.
	Caption	Список товаров	Текст – заголовок.
MSFlexGrid	Name	MSFlexGrid1	Имя, на которое можно ссылаться в коде.
CommandButton	Name	CmdLoad	
	Caption	Загрузить список	Заголовок для кнопки.
CommandButton	Name	CmdExit	
	Caption	Выход	Заголовок для кнопки.
	Cancel	True	Клавиша Esq также вызовет процедуру CmdExit_Click.
Label	Name	Label1	Метка для TextBox1.
TextBox	Name	TextBox1	Текстовое окно для ввода строки с SQL-запросом.
Frame	Name	Frame2	Имя, на которое можно ссылаться в коде.
	Caption	Добавление записи	Текст – заголовок.
Label	Name	Label2	Метка для TextBox2.
Label	Name	Label3	Метка для TextBox3.
Label	Name	Label4	Метка для TextBox4.
TextBox	Name	TextBox2	Текстовое окно для ввода кода товара.
TextBox	Name	TextBox3	Текстовое окно для ввода наименования товара.
TextBox	Name	TextBox4	Текстовое окно для ввода цены товара.
CommandButton	Name	CmdAdd	Кнопка для добавления записи.
	Caption	Добавить	Заголовок кнопки.

В листинге 18.6 приведен код модуля формы, отображенной на рис. 18.6. В строках 81–93 описана событийная процедура **CmdAdd_Click**, которая добавляет к набору новую (пустую) за-

пись (строка 84), а затем, используя содержимое текстовых окон **TextBox2**, **TextBox3** и **TextBox4**, изменяет содержимое полей этой записи (строки 87–89). Код строки 91 добавляет запись в базу данных.

Листинг 18.6 Добавление данных в таблицу, отображаемую в объект Recordset

```
1:  Dim cn  As ADODB.Connection
2:  Dim rstNwind As ADODB.Recordset
3:
4:  Function ADODB_ConnectedJet() As Boolean
5:  ' устанавливает соединение для Jet-провайдера
6:
7:      On Error GoTo err_not_connection
8:
9:
10:     Set cn = New ADODB.Connection
11:
12:     cn.Provider = "Microsoft.Jet.OLEDB.4.0"
13:     cn.ConnectionString = "f:\фирма.mdb"
14:     cn.Open
15:
16:     ADODB_ConnectedJet = True
17:     Exit Function
18:
19: err_not_connection:
20:     ADODB_ConnectedJet = False
21:
22: End Function
23:
24: Private Sub CmdExit_Click()
25:
26:     rstNwind.Close
27:     cn.Close
28:
29:     Unload Me
30: End Sub
31:
32: Private Sub CmdLoad_Click()
33: ' загрузка списка записями набора
34:
35:     Dim newElement As String, i As Integer
36:
37:     If Not ADODB_ConnectedJet() Then
38:         MsgBox " Невозможно подключиться к базе данных!"
39:         Exit Sub
40:     End If
41:
42:
43:     Set rstNwind = New ADODB.Recordset
44:
45:     rstNwind.Source = TextBox1.Text
46:     rstNwind.LockType = adLockOptimistic
47:
48:     Set rstNwind.ActiveConnection = cn
49:
50:     'открыть набор:
51:     rstNwind.Open
52:
53:
54:     rstNwind.MoveFirst      'перейти к началу набора
55:
56:     'задать количество колонок:
57:     MSFlexGrid1.Cols = rstNwind.Fields.Count + 1
58:
59:     'установить ширину первых колонок
60:     MSFlexGrid1.ColWidth(0) = 1
61:     If MSFlexGrid1.Cols > 1 Then MSFlexGrid1.ColWidth(1) = 2500
62:     If MSFlexGrid1.Cols > 2 Then MSFlexGrid1.ColWidth(2) = 2000
63:
64:     'до конца набора добавлять записи в список
65:     Do While Not rstNwind.EOF
66:
```

```
67:         'сформировать элемент добавления:
68:         newElement = ""
69:         For i = 0 To rstNwind.Fields.Count - 1
70:            newElement = newElement & vbTab & rstNwind.Fields(i)
71:         Next
72:
73:         'добавить элемент к таблице:
74:         MSFlexGrid1.AddItem newElement
75:
76:         rstNwind.MoveNext
77:      Loop
78:
79:  End Sub
80:
81:  Private Sub CmdAdd_Click()
82:  'добавление записи (без проверки уникальности кода)
83:
84:      rstNwind.AddNew     'добавить пустую запись в набор
85:
86:      'редактирование данных в добавленной записи:
87:      rstNwind.Fields(0) = TextBox2.Text
88:      rstNwind.Fields(1) = TextBox3.Text
89:      rstNwind.Fields(2) = TextBox4.Text
90:
91:      rstNwind.Update     'сохранить запись
92:
93:  End Sub
```

Одним из наиболее «интересных» методов объекта **`ADODB.Recordset`** является метод **`Save`**, который записывает данные набора в файл и имеет следующий синтаксис:

Синтаксис

```
recordset.Save FileName, PersistFormat
```

Здесь *FileName* — полное имя файла для записи, *PersistFormat* — значение константы, определяющей формат файла: **`adPersistADTG`** (является значением по умолчанию и указывает на формат, являющийся собственностью Advanced Data Tablegram) или **`adPersistXML`** (XML-формат).

Далее рассматривается пример использования метода **`Save`** объекта **`ADODB.Recordset`**. Для этого предлагается создать диалоговое окно на базе формы, представленной на рис. 18.21. В следующей ниже таблице описаны некоторые свойства элементов управления формы.

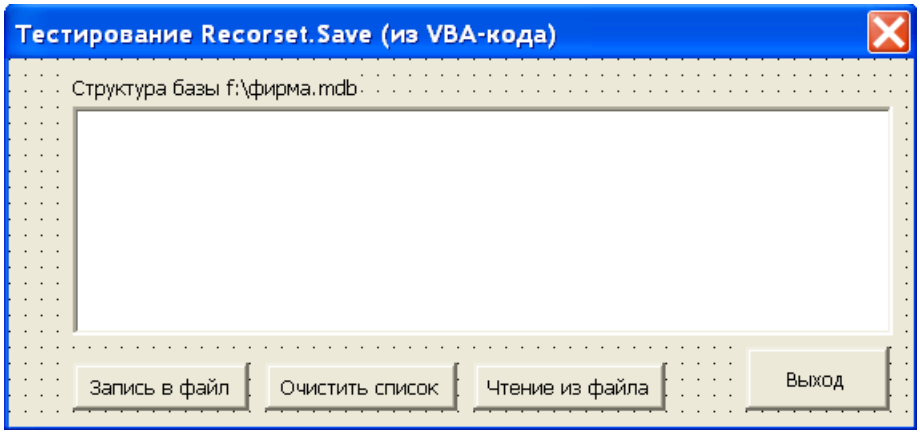

Рис. 18.7
Форма для тестирования приложения с использованием методов **ASave** и **Open** объекта **Recordset** в режиме разработки

Тип элемента	Свойство, которое изменено (используется в коде)	Значение	Примечание
UserForm	Name	UserForm2	Имя главной формы, на которое можно ссылаться в коде.
	Caption	Тестирование Recorset.Save (из VBA-кода)	

ListBox	Name	ListBox1	Окно для отображения списка.
CommandButton	Name	CmdLoad	Имя кнопки.
	Caption	Запись в файл	Заголовок для кнопки.
CommandButton	Name	CmdClear	Имя кнопки.
	Caption	Очистить список	Заголовок для кнопки.
CommandButton	Name	CmdRead	Имя кнопки.
	Caption	Чтение из файла	Заголовок для кнопки.
CommandButton	Name	CmdExit	Имя кнопки.
	Caption	Выход	Заголовок для кнопки.
	Cancel	True	Клавиша Esq также вызовет процедуру CmdExit_Click.
Label1	Name	Label1	
	Caption	Структура базы f:\фирма.mdb	

В листинге 18.7 представлен код модуля формы. Алгоритм работы с формой очень простой: при щелчке на кнопке **Запись в файл** посредством кода процедуры **CmdLoad_Click** (строки 27–54) создается набор **rstNwind** в результате вызова метода **`OpenSchema`** объекта **cn** типа **`Connection`** (строка 38). После создания набора его записи помещаются (строки 42–54) в список (рис. 18.8) и сохраняются в XML-файле **c:\rstNwindSave.xml** (строка 49) методом **`Save`**. Затем набор **rstNwind** и соединение **cn** закрываются (строки 58, 59).

Листинг 18.7 Использование методов Save и Open объекта Recordset

```
 1:  Dim cn  As ADODB.Connection
 2:
 3:  Function ADODB_ConnectedJet() As Boolean
 4:  ' устанавливает соединение для Jet-провайдера
 5:
 6:
 7:      On Error GoTo err_not_connection
 8:
 9:      Set cn = New ADODB.Connection
10:
11:      cn.Provider = "Microsoft.Jet.OLEDB.4.0"
12:      cn.ConnectionString = "f:\фирма.mdb"
13:      cn.Open
14:
15:      ADODB_ConnectedJet = True
16:      Exit Function
17:
18:  err_not_connection:
19:      ADODB_ConnectedJet = False
20:
21:  End Function
22:
23:  Private Sub CmdExit_Click()
24:      Unload Me
25:  End Sub
26:
27:  Private Sub CmdLoad_Click()
28:
29:      Dim rstNwind As ADODB.Recordset
30:      Dim newElement As String, i As Integer
31:
32:      If Not ADODB_ConnectedJet() Then
33:          MsgBox " Невозможно подключиться к базе данных!"
34:          Exit Sub
35:      End If
36:
37:      'открытие набора с данными о структуре базы данных:
38:      Set rstNwind = cn.OpenSchema(adSchemaTables)
39:
40:      'добавление элементов набора в список:
41:      Do Until rstNwind.EOF
```

```
42:         ListBox1.AddItem "Table name: " & _
43:             rstNwind!TABLE_NAME & vbCr & _
44:             "Table type: " & rstNwind!TABLE_TYPE & vbCr
45:             rstNwind.MoveNext
46:     Loop
47:
48:     'запись набора в файл:
49:     rstNwind.Save "c:\rstNwindSave.xml", adPersistXML
50:
51:     rstNwind.Close  'закрыть набор
52:     cn.Close        'закрыть соединение
53:
54: End Sub
55:
56:
57: Private Sub CmdRead_Click()
58:     'чтение набора из файла
59:
60:     Dim rstNwind As New ADODB.Recordset
61:     Dim newElement As String, i As Integer
62:
63:
64:     'открытие набора с данными о структуре базы данных:
65:     rstNwind.Open "c:\rstNwindSave.xml"
66:
67:
68:     'добавление элементов набора в список:
69:     Do Until rstNwind.EOF
70:         ListBox1.AddItem "Table name: " & _
71:             rstNwind!TABLE_NAME & vbCr & _
72:             "Table type: " & rstNwind!TABLE_TYPE & vbCr
73:         rstNwind.MoveNext
74:     Loop
75:
76:     rstNwind.Close  'закрыть набор
77:
78: End Sub
79:
80: Private Sub CmdClear_Click()
81:     ListBox1.Clear
82: End Sub
```

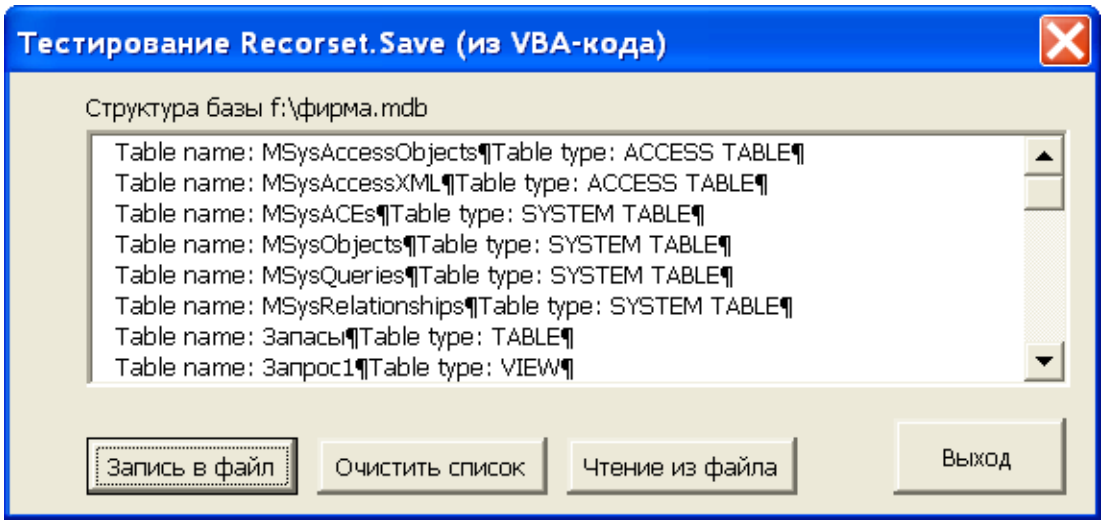

Рис. 18.8
Форма для тестирования приложения с использованием методов **ASave** и **Open** объекта **Recordset** в режиме выполнения

После загрузки набора в окно списка и файл можно щелкнуть на кнопке **Очистить список** для удаления элементов из списка процедурой **CmdClear_Click** (строки 80–82). В этот момент приложение никак не связано с базой данных, а в файле **c:\rstNwindSave.xml** хранятся записи

ранее используемого набора. На рис. 18.9 часть этого файла показана при помощи приложения Internet Explorer.

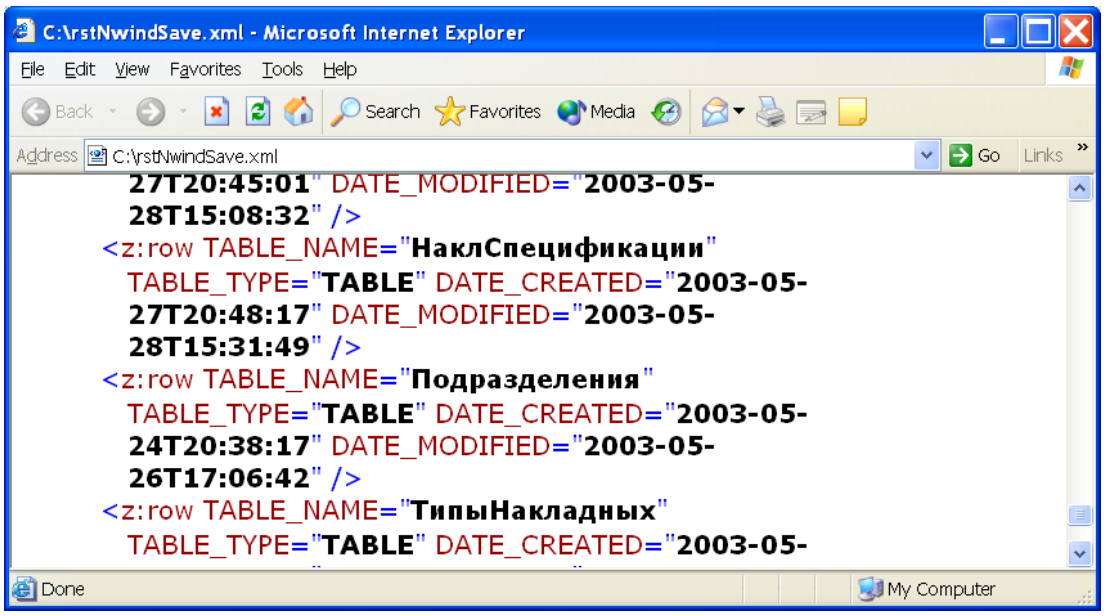

Рис. 18.9
Так выглядит файл C:\rstNwindSave.xml в приложении Internet Explorer после записи с использованием методов **ASave** и **Open** объекта **Recordset**

Теперь можно щелкнуть на кнопке **Чтение из файла**. Процедура **CmdRead_Click** (строки 64–85) откроет набор **rstNwind** (но уже с использованием в качестве источника файла **c:\rstNwindSave.xml**) и заполнит окно списка.

Использование провайдера Microsoft Jet 4.0 OLE DB для подключения к dbf-файлу

В статье Иванова Д.М. «Опыт использования ADO для доступа к базам данных форматов MS Access, xBase и Paradox», которую можно найти на Web-страничке с адресом http://vlad2000.h1.ru/Frames/Statyi/Access5.html, для подключения к dbf-файлу предлагается использовать строку подключения с параметром **`Extended Properties`**. В справочной системе нет сведений об этом параметре, хотя все действительно работает. Сам Иванов отмечает, что нашел эту информацию в Internet.

В следующем фрагменте кода приведен пример подключения к файлу **c:\dbf\t2.dbf** формата dBase III.

```
Dim cn As ADODB.Connection
Dim adoRS As ADODB.Recordset

pathDBF="c:\dbf\"                          'путь к dbf-файлу
strSQL = "SELECT * FROM T2.dbf"            ' SQL-запрос:

Set cn = New ADODB.Connection
cn.Provider = "Microsoft.Jet.OLEDB.4.0"
cn.ConnectionString = "Data Source=" & pathDBF & _
                      "; Extended Properties=dBase III"
cn.Open

Set adoRS = New ADODB.Recordset
adoRS.CursorType = adOpenKeyset
adoRS.LockType = adLockOptimistic
adoRS.Open strSQL, cn
```

Подобным образом можно подключаться к dbf-файлам формата dBase 4 и 5. В реестре Windows в секции HKEY_LOCAL_MACHINE\Software\Microsoft\Jet\4.0\ISAM Formats можно увидеть

точную запись для **`Extended Properties`**, если нужен отличный от dBase III формат (эта информация — также из статьи Иванова).

Пример подключения к mdb-файлу из Access

Как уже было отмечено, код листингов 18.1–18.3 можно использовать для подключения к внешней базе данных из Access. Хотя Access позволяет хранить данные (таблицы с информацией) и средства их обработки (формы, отчеты и т.п.) в одном файле, чаще бывает необходимо базу данных с таблицами хранить в одном mdb-файле, а все, что связано с управлением данными, — в другой. Если, например, ваш проект используется несколькими специалистами, а вы непрерывно модифицируете диалоговые формы и отчеты для работы с данными, то, конечно, передавать пользователям ваши доработки, не затрагивая пользовательских данных, удобно именно в отдельном файле.

На рис. 18.10 приведена форма в Design-режиме для файла control.mdb, который не содержит ни одной таблицы. В следующей ниже таблице описаны некоторые свойства элементов управления формы.

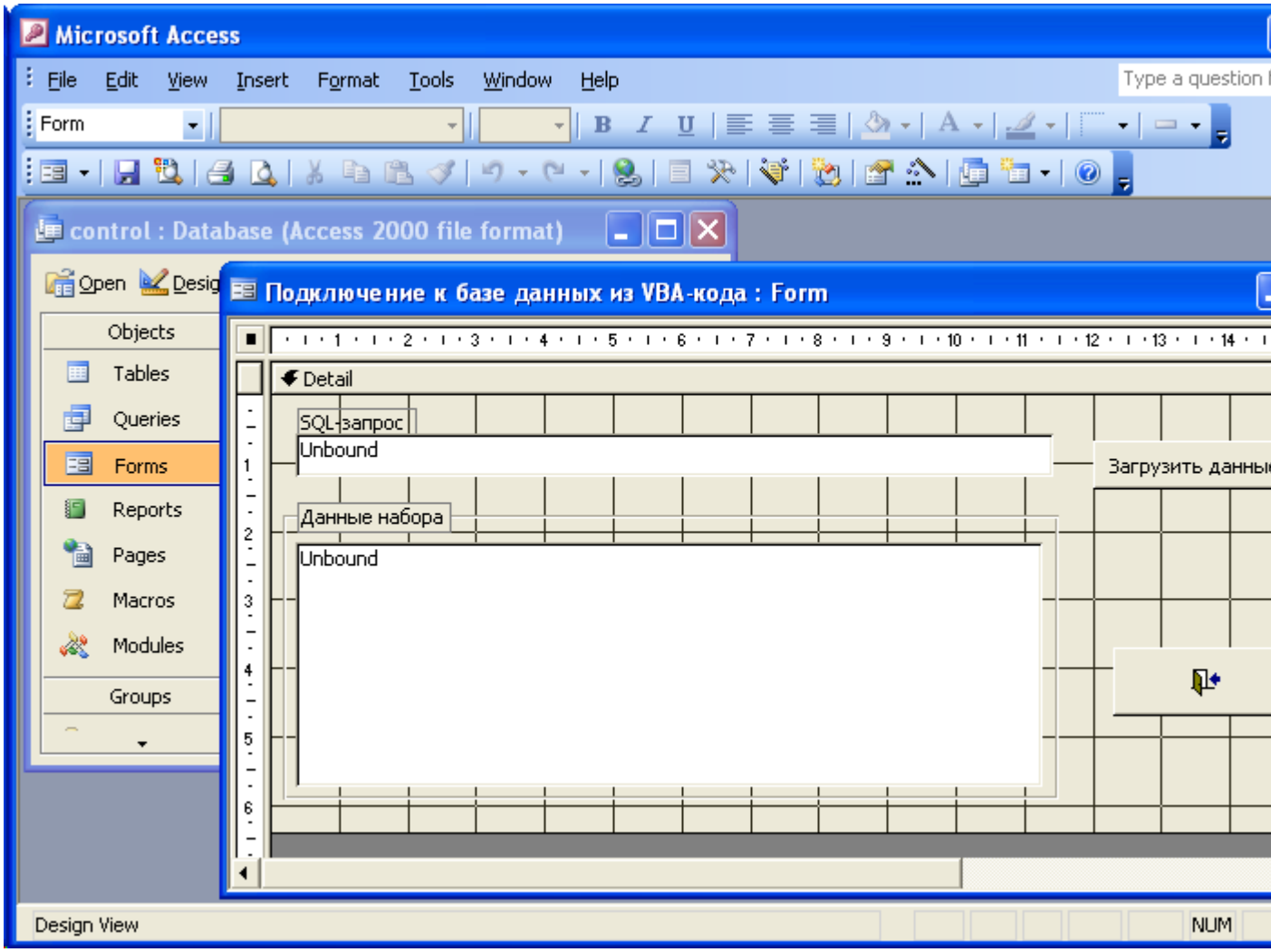

Рис. 18.10
Форма в Design-режиме для файла control.mdb, который не содержит ни одной таблицы

Тип элемента	Свойство, которое изменено (используется в коде)	Значение	Примечание
Form	Name	Form	
	Caption	Подключение к базе данных из VBA-кода	Заголовок формы.
List Box	Name	ListBox1	Окно для отображения одного поля набора.

	Row Source Type	Value List	
Command Button	Name	Кнопка6	Имя кнопки для выхода из формы.
Command Button	Name	CmdLoad	Имя кнопки для выполнения кода загрузки данных.
	Caption	Загрузить данные	Заголовок кнопки.
Text Box	Name	SQLText	Текстовое окно для ввода SQL-запроса.

Для подключения к внешней базе данных **фирма.mdb** в рассматриваемом примере используется функция **ADODB_ConnectedJet**, приведенная в листинге 18.3. Как показано на рис. 18.11, эта функция находится в модуле **Module1**, чтобы ее можно было вызвать из любой формы проекта.

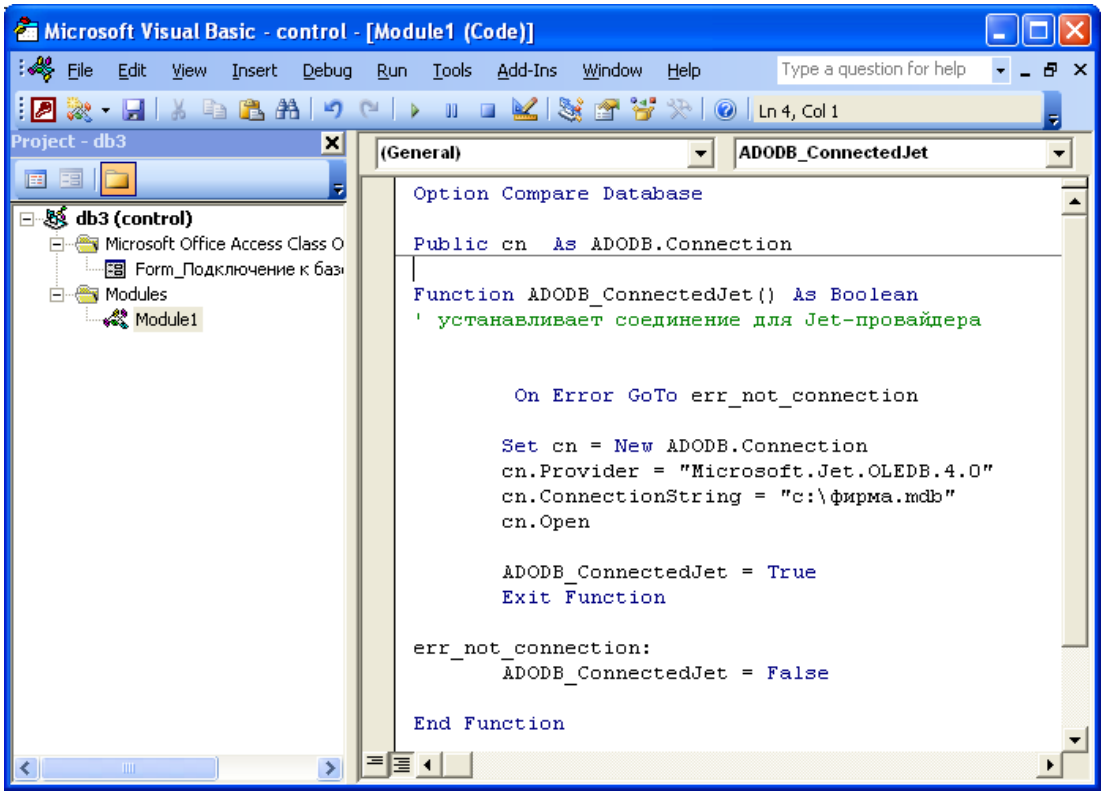

Рис. 18.11
Функция **ADODB_ConnectedJet** находится в модуле **Module1**, чтобы ее можно было вызвать из любой формы проекта

Код модуля формы **Form** приведен в листинге 18.8. Здесь в строках 3–37 описана процедура **CmdLoad_Click** — событийный обработчик кнопки **Загрузить данные**. В строке 5 возвращаемое значение функции **ADODB_ConnectedJet** используется в качестве условного выражения оператора **`If`**. Если соединение с mdb-файлом установлено (функция **ADODB_ConnectedJet** возвращает значение **`True`**), создается набор **rs** (тип **`ADODB.Recordset`**), использующий в качестве свойства **`Source`** SQL-запрос в текстовом окне **SQLText**.

Листинг 18.8 Код модуля формы

```
1:      Option Compare Database
2:
3:      Private Sub CmdLoad_Click()
4:
5:          If ADODB_ConnectedJet() Then
6:              'Похоже, соединение установлено
7:              Dim rs As ADODB.Recordset
8:              Set rs = New ADODB.Recordset
9:              Set rs.ActiveConnection = cn
10:             rs.CursorType = adOpenKeyset
11:
12:             'использовать SQL-запрос из текстового окна:
```

```
13:             SQLText.SetFocus
14:             rs.Source = SQLText.Text
15:
16:             'открыть набор:
17:             rs.Open
18:
19:             'удалить все элементы списка:
20:             For i = ListBox1.ListCount - 1 To 0 Step -1
21:                ListBox1.RemoveItem Index:=i
22:             Next
23:
24:             'заполнить список первыми "столцами" набора:
25:             rs.MoveFirst
26:             Do While Not rs.EOF
27:                ListBox1.AddItem rs.Fields(0)
28:                rs.MoveNext
29:             Loop
30:
31:             rs.Close      ' закрыть соединение
32:             cn.Close      ' закрыть соединение
33:          Else
34:                MsgBox "Невозможно подключиться к mdb-файлу!"
35:          End If
36:
37:       End Sub
38:
39:
40:       Private Sub Кнопка6_Click()
41:       On Error GoTo Err_Кнопка6_Click
42:
43:          DoCmd.Close
44:
45:       Exit_Кнопка6_Click:
46:          Exit Sub
47:
48:       Err_Кнопка6_Click:
49:          MsgBox Err.Description
50:          Resume Exit_Кнопка6_Click
51:
52:       End Sub
```

В строках 20–22 из **ListBox1** удаляются все отображаемые элементы, а в строках 26–29 **ListBox1** снова заполняется данными из открытого набора. В строках 40–52 описана (средствами Access) процедура выхода из диалогового окна.

На рис. 18.12 рассматриваемое диалоговое окно показано в режиме выполнения. Здесь в текстовом окне **SQLText** был введен SQL-запрос **SELECT НаимТовара FROM Товары**, а в списке **ListBox1** отображены значения первого (единственного) поля набора.

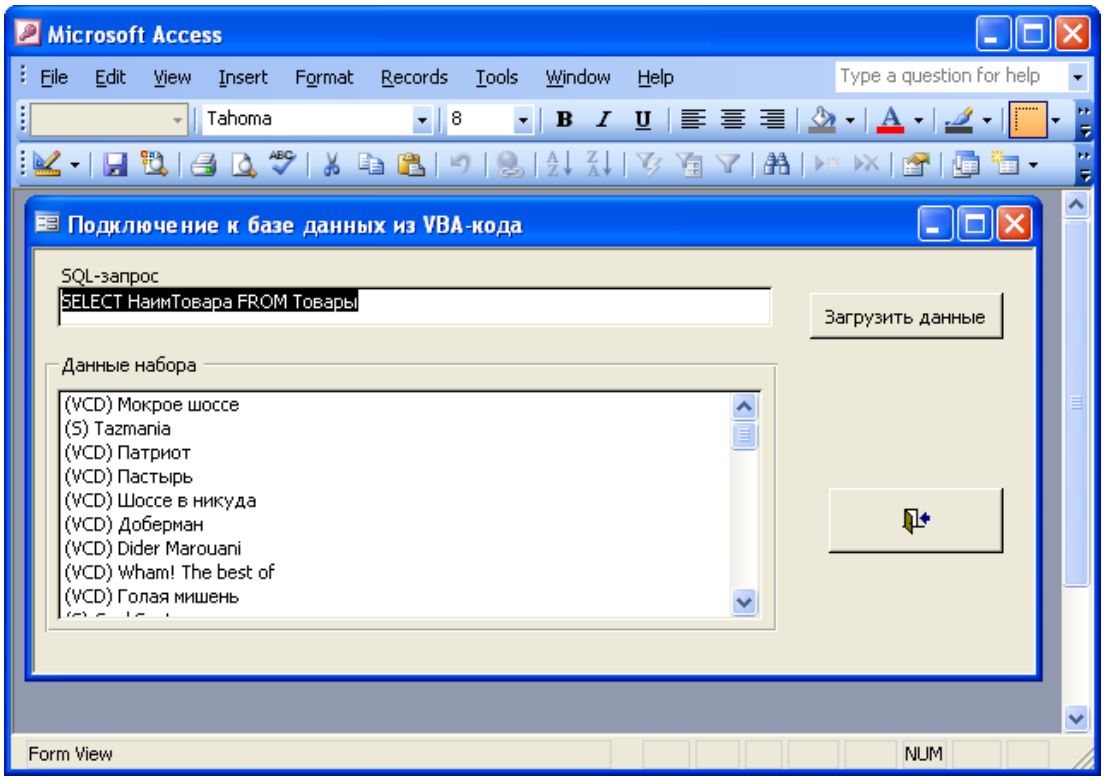

Рис. 18.12
В списке **ListBox1** отображены значения первого (единственного) поля набора

Обработка транзакций

При обновлении одновременно нескольких таблиц базы данных часто бывает необходимо, чтобы были внесены изменения либо во все таблицы, либо ни в одну из них. Это называется принципом «Все или ничего», а процесс одновременного (или почти одновременного) обновления нескольких таблиц базы (одной или нескольких) данных называется транзакцией.

Транзакция — это серия обновлений таблиц базы данных, которая может быть отменена, если в процессе ее выполнения произойдет сбой, который не позволит выполнить какое-либо обновление, что может привести к противоречивой информации в таблицах базы данных. Например, при регистрации товара на некотором складе необходимо занести данные о товаре в инвентаризационную ведомость (ранее в книге использовалась таблица **Запасы**) и в таблицу с накладными. То есть, по меньшей мере, нужно обновить информацию в двух таблицах (при регистрации нового товара может понадобиться изменить и справочник товаров с кодами и наименованиями товаров). Если обновление инвентаризационной ведомости пройдет успешно, а при записи данных в накладные произойдет сбой, то такая база данных будет содержать противоречивую информацию, что, между прочим, не так легко обнаружить. Если использовать механизм транзакций, то мы можем отменить обновление инвентаризационной ведомости, если при обновлении таблицы с накладными произошел сбой.

Для работы с транзакциями объект **`ADODB.Connection`** имеет следующие методы:

`BeginTrans`	Запускает транзакцию.
`Close`	Закрывает открытый объект и любые зависимые объекты.
`CommitTrans`	Завершает транзакцию, закрепляя изменения в источнике данных. Разрешает запуск новой транзакции.
`Execute`	Выполняет определенный запрос: SQL-инструкцию, хранимую процедуру или текст, определяемый провайдером.
`Open`	Открывает соединение с источником данных.
`OpenSchema`	Возвращает информацию о схеме базы данных.

RollbackTrans	Отменяет любые изменения, выполненные во время транзакции, и завершает транзакцию (отменяет транзакцию). Разрешает запуск новой транзакции.

Некоторые инструкции языка определения данных (DDL)

Ознакомившись с некоторыми понятиями технологий доступа к базам данных из кода Visual Basic, вы можете создавать таблицы базы данных при помощи языка SQL. Основу языка определения данных составляют инструкции **CREATE**, **DROP** и **ALTER**. Эти инстукции используются для определения таблиц, индексов, представлений и процедур.

Инструкция CREATE TABLE

Инструкция CREATE TABLE используется для описания новой таблицы текущей базы данных. Синтаксис запроса следующий:

СИНТАКСИС

CREATE [TEMPORARY] TABLE *tableName* (*fieldName1* [(*size*)] [NOT NULL] [WITH COMPRESSION | WITH COMP] [*index1*] [,*fieldName2*]

Здесь *field1*, *field2* — наименования полей, которые следует копировать в новую таблицу; *newTable* — наименование создаваемой таблицы; *externaldatabase* — полное имя внешней базы данных; ***source*** — источник данных: таблица, несколько таблиц или запрос .

Например, код листинга 18.9 создает в файле **c:\master\mdb\AllBase.mdb** новую таблицу с именем **TestTable**. Этот код можно записать в Word- или Excel-модуль и выполнить из режима разработки.

Листинг 18.9 Создание таблицы из VBA-кода при помощи SQL-инструкции

```
Dim cn  As ADODB.Connection

Function ADODB_ConnectedJet() As Boolean
' устанавливает соединение для Jet-провайдера

   'установить обработчик ошибок:
   On Error GoTo err_not_connection

   'инициализировать cn
    Set cn = New ADODB.Connection
    cn.Provider = "Microsoft.Jet.OLEDB.4.0"
    cn.ConnectionString = " c:\фирма.mdb.mdb"
    cn.Open     'отрыть соединение

    ADODB_ConnectedJet = True   'возвращаемое значение
    Exit Function   'обойти обработчик ошибок

err_not_connection:
    ADODB_ConnectedJet = False

End Function

Sub CreateTable()
'тестирует ADODB_Connected: устанавливает соединение и
'создает новую таблицу в базе данных

    If ADODB_ConnectedJet() Then
        MsgBox "Похоже, соединение установлено!"
        'код, использующий установленное соединение:

        strTable = "CREATE TABLE TestTable " & _
            "(field_1 TEXT(5), " & _
            " field_2 TEXT(30))"

        cn.Execute strTable
```

```
36
37        cn.Close    ' закрыть соединение
38     Else
39        MsgBox " Что-то не сложилось ..."
40     End If
41
42  End Sub
```

В строках 3–21 описана функция подключения к базе данных. Эта функция скопирована из листинга 18.3. В строках 23–42 содержится процедура **CreateTable**, которая, используя функцию **ADODB_ConnectedJet** для подключения к базе данных, создает (в строках 31–35) новую таблицу. Этот код можно без всяких изменений выполнить в любом приложении Microsoft Office и в Visual Basic, поскольку здесь не используются специфические элементы управления.

Инструкция CREATE INDEX

Инструкция **`CREATE INDEX`** создает новый индекс для существующей таблицы и имеет следующий синтаксис:

СИНТАКСИС

```
CREATE [ UNIQUE ] INDEX index
   ON table (field1 [ASC|DESC][, field2 [ASC|DESC], ...])
   [WITH { PRIMARY | DISALLOW NULL | IGNORE NULL }]
```

Здесь *index* — имя создаваемого индекса; *table* — имя таблицы, для которой создается индекс; *fieldi* — имя поля, включаемого в индекс. Для расположения элементов индекса в убывающем порядке следует использовать зарезервированное слово DESC; в противном случае будет принят порядок по возрастанию.

Зарезервированное слово **`UNIQUE`** следует использовать для запрета совпадения значений индексированных полей в разных записях.

Необязательное предложение **`WITH`** позволяет задать условия на значения. Например:

- С помощью параметра **`DISALLOW NULL`** можно запретить значения **`Null`** в индексированных полях новых записей.
- Параметр **`IGNORE NULL`** запрещает включение в индекс записей, имеющих значения **`Null`** в индексированных полях.
- С помощью зарезервированного слова **`PRIMARY`** можно указать, что индексированное поле должно быть ключевым (поскольку такой индекс по умолчанию является уникальным, зарезервированное слово **`UNIQUE`** можно опустить).

Процедура **CreateTable2** листинга 18.10, использующая функцию **ADODB_ConnectedJet**, отличается от процедуры **CreateTable** тем, что она не только создает новую таблицу (**TestTable2**), но и определяет ключевое поле (**field_1**).

Листинг 18.10 Создание таблицы и ключевого поля из VBA-кода при помощи SQL-инструкции

```
Sub CreateTable2()
'тестирует ADODB_Connected: устанавливает соединение и
'создает новую таблицу в базе данных, определяет первое
'поле таблицы как ключевое

   If ADODB_ConnectedJet() Then
      MsgBox "Фантастика! Соединение установлено!"
      'код, использующий установленное соединение:

      'создать таблицу:
      strTable = "CREATE TABLE TestTable2 " & _
         "(field_1 TEXT(5), " & _
         " field_2 TEXT(30))"

      cn.Execute strTable

      'определить ключевое поле:
```

```
    strTable = "CREATE INDEX index1 " & _
       " ON TestTable2 (field_1) WITH  PRIMARY "

    cn.Execute strTable

    cn.Close    ' закрыть соединение
  Else
    MsgBox " Что-то не сложилось ..."
  End If

End Sub
```

Если после выполнения кода листинга 18.10 открыть созданную таблицу при помощи **Конструктора таблиц**, то можно увидеть описание полей, приведенное на рис. 18.13.

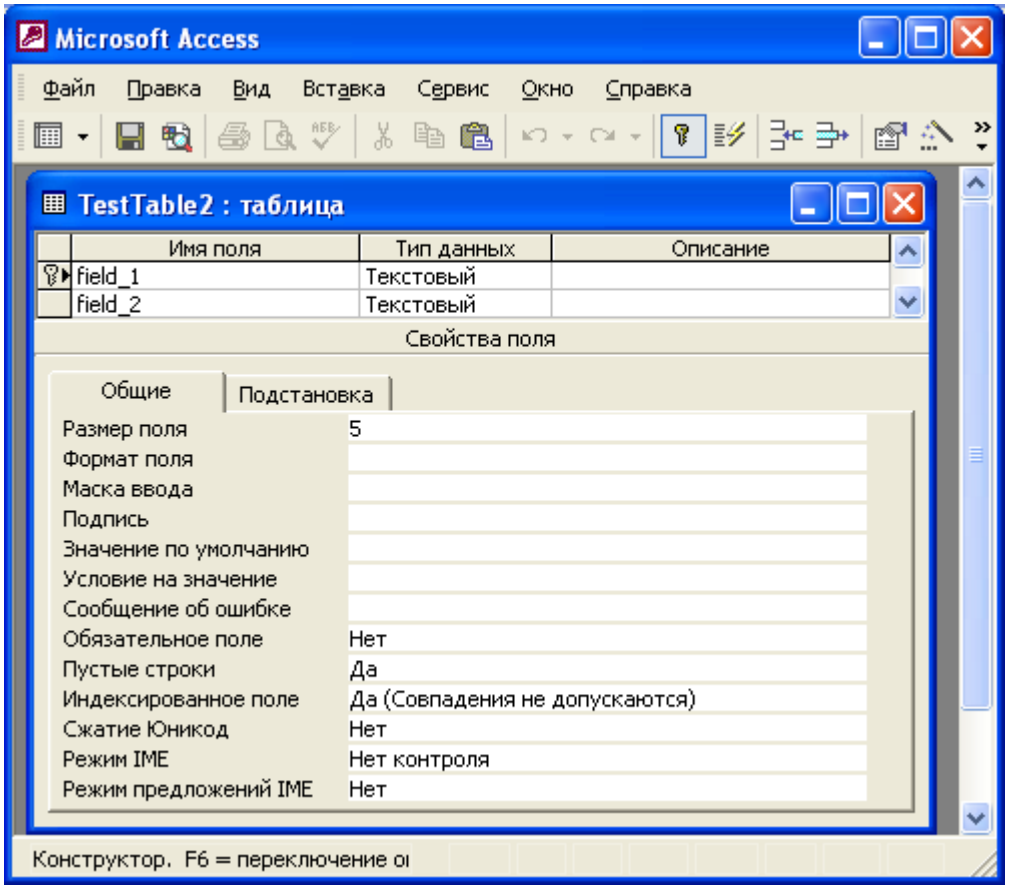

Рис. 18.13
Если после выполнения кода листинга 18.2 открыть созданную таблицу при помощи** Конструктора таблиц**, то можно увидеть именно такое описание полей

Инструкция ALTER TABLE

Инструкция **`ALTER TABLE`** изменяет структуру таблицы, созданной, быть может, с помощью инструкции **`CREATE TABLE`**, и имеет следующий синтаксис:

СИНТАКСИС

```
ALTER TABLE table {    ADD {    COLUMN  field_name  mun_поля [(size)] [NOT NULL]    [CONSTRAINT index_name] }
                     | ALTER COLUMN field_name  field _type[(size)]
                     | ADD CONSTRAINT <constraint_index>
                     | DROP {COLUMN field_name  I CONSTRAINT constraint_index_name }
                   }
```

Здесь *table* — наименование изменяемой таблицы; *field* — имя поля, добавляемого в *таблицу* или удаляемого из нее, или имя поля, заменяемого в *таблице*; *field _type* — тип данных *поля*; *size* — размер поля в знаках (только для текстовых и двоичных полей); *index* — индекс для *по-*

ля; *<constraint_index>* — описание составного индекса, добавляемого к *таблице*; *constraint_index_name* — имя составного индекса, который следует удалить.

Как следует из синтаксиса инструкции **ALTER TABLE**, с существующей таблицей можно производить следующие операции:

- Добавить новое поле (с помощью предложения **ADD COLUMN**). При этом следует указать имя поля, его тип и (для текстовых и двоичных полей) размер (необязательно). Кроме того, можно создать индекс по этому полю. Если для поля добавлено ограничение **NOT NULL**, то при добавлении новых записей это поле должно содержать допустимые данные. Например, следующая инструкция добавляет в таблицу **TestTable2** текстовое поле **field_3** длиной 5 знаков:

```
ALTER TABLE TestTable2  ADD COLUMN field_3 TEXT(5)
```

- Изменить тип существующего поля (с помощью предложения **ALTER COLUMN**). В этом случае следует указать имя поля, его тип и (для текстовых и двоичных полей) размер (необязательно). Например, следующая инструкция изменяет в таблице **TestTable2** размер поля **field_2**, переопределяя это поле как текстовое длиной 40 знаков:

```
ALTER TABLE TestTable2  ALTER COLUMN field_2 TEXT(40)
```

- Добавить составной индекс (с помощью зарезервированных слов **ADD CONSTRAINT**).

- Удалить поле (с помощью зарезервированных слов **DROP COLUMN**) или составной индекс (с помощью зарезервированных слов **DROP CONSTRAINT**).

Замечание

```
      Нельзя добавить или удалить одновременно несколько полей или
индексов. Инструкцию CREATE INDEX можно использовать для добавления
к таблице простого или составного индекса.
```

Замечание

```
      Ограничение NOT NULL можно наложить на поле только один раз.
При попытке применить это ограничение несколько раз возникает ошибка
выполнения.
```

Процедура **ALTER_TABLEtest** листинга 18.11, использующая функцию **ADODB_ConnectedJet**, добавляет к таблице **TestTable2** новое поле — **field_3**. Результат работы процедуры можно посмотреть при помощи коструктора таблиц в Microsoft Access — рис. 18.14. На этом рисунке видно, что таблица **TestTable2** включает три поля и длина поля **field_2** увеличена до 40 символов.

Листинг 18.11 Модификация таблицы из VBA-кода при помощи SQL-инструкции

```
1   Sub ALTER_TABLEtest()
2  'модифицирует таблицу базы данных
3  'при помощи SQL-инструкции ALTER TABLE
4
5     If ADODB_ConnectedJet() Then
6        MsgBox "Похоже, соединение установлено!"
7        'код, использующий установленное соединение:
8
9        'добавить новое поле:
10       strTable = "ALTER TABLE TestTable2 " & _
11               "    ADD COLUMN field_3 TEXT(5)"
12
13       cn.Execute strTable
14
15       'изменить длину поля:
16       strTable = "ALTER TABLE TestTable2" & _
17               "    ALTER COLUMN field_2 TEXT(40)"
```

```
18
19        cn.Execute strTable
20
21        cn.Close    ' закрыть соединение
22    Else
23        MsgBox " Что-то не сложилось ..."
24    End If
25
26 End Sub
```

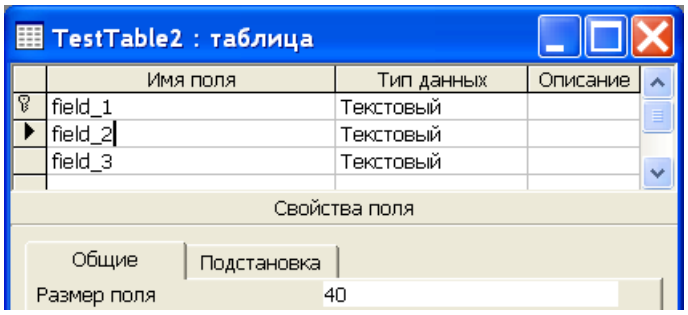

Рис. 18.14
Результат работы процедуры *ALTER_TABLEtest* ***можно посмотреть при помощи коструктора таблиц в Microsoft Access***

Если процедуру **ALTER_TABLEtest** попытаться выполнить еще раз, то возникнет ошибка (поле ‘field_3’ уже существует в таблице ‘TestTable2’), о которой будет сообщено посредством окна, приведенного на рис. 18.15.

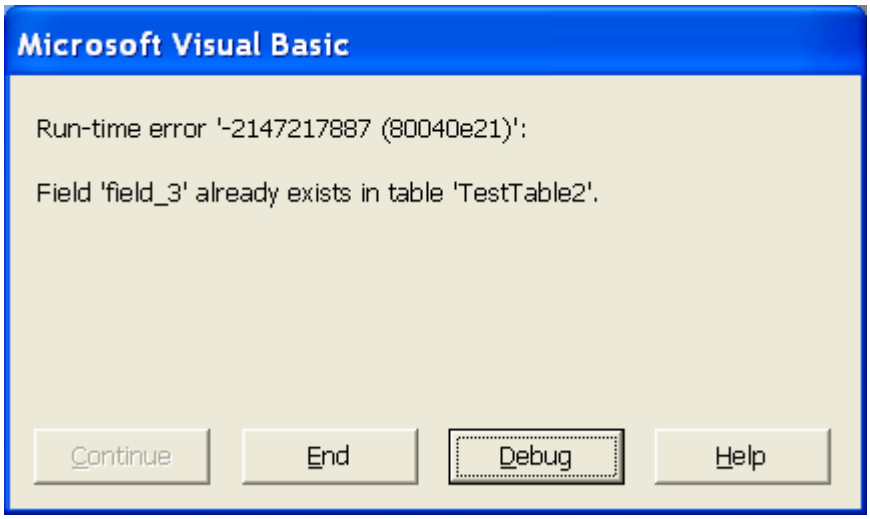

Рис. 18.15
Если процедуру *ALTER_TABLEtest1* ***попытаться выполнить еще раз, то возникнет ошибка***

Предложение CONSTRAINT

Создание *ограничения* с помощью предложения **CONSTRAINT** подобно использованию индекса, хотя оно также применяется для установления отношений между таблицами.

Предложение **CONSTRAINT** используется в инструкциях **ALTER TABLE** и **CREATE TABLE** для создания или удаления индексов. Существуют два типа предложений **CONSTRAINT**: одно для создания простого индекса (по одному полю), а второе для создания составного индекса (по нескольким полям).

СИНТАКСИС

простой индекс:

```
CONSTRAINT index_name {PRIMARY KEY | UNIQUE | NOT NULL |
  REFERENCES external_table [(external_field_1, external_field_2)]
  [ON UPDATE CASCADE | SET NULL]
  [ON DELETE CASCADE | SET NULL]}
```

составной индекс:

```
CONSTRAINT index_name
  {PRIMARY KEY (key_field_1[, key_field _2 [, ...]]) |
  UNIQUE (unique_1[,unique_2 [, ...]]) |
  NOT NULL (not_null_1[,not_null_2 [, ...]]) |
  FOREIGN KEY [NO INDEX] (ref_1[, ref_2 [, ...]])
    REFERENCES external_table [(external_field_1 [,external_field_2 [, ...]])]
  [ON UPDATE CASCADE | SET NULL]
  [ON DELETE CASCADE | SET NULL]}
```

Здесь *table* — наименование изменяемой таблицы; *index_name* — имя создаваемого индекса; *key_field_1*, *key_field_2* — имена одного или нескольких полей, которые следует обозначить как ключевые; *unique_1*, *unique_2* — имена одного или нескольких полей, которые следует включить в уникальный индекс; *not_null_1*, *not_null_2* — имена одного или нескольких полей, в которых запрещаются значения **Null**; *ref_1*, *ref_2* — имена одного или нескольких полей, включенных во внешний ключ, которые содержат ссылки на поля в другой таблице; *external_table* — имя внешней таблицы, которая содержит поля, указанные с помощью аргумента *external_field_i*; *external_field_1*, *external_field_2* — имя поля или имена полей таблицы *external_table*, указанные с помощью *ref_1* и *ref_2*; если адресуемое поле является ключом таблицы *external_table*, данное предложение можно опустить.

Инструкция CREATE PROCEDURE

Инструкция **CREATE PROCEDURE** создает хранимую процедуру и имеет следующий синтаксис:

СИНТАКСИС

```
CREATE PROCEDURE ProcedureName
  [parameter1 DataType1[,parameter2  DataType2 [, ...]]
  AS SQL-command
```

Здесь *ProcedureName* — имя процедуры (должно удовлетворять требованиям к именованию переменных), *parameter1*, *parameter1, ...* — имена параметров процедуры (до 255 параметров); *DataType1*, *DataType2*, … — типы параметров процедуры; *SQL-command* — инструкция SQL, такая как **SELECT**, **UPDATE**, **DELETE**, **INSERT**, **CREATE TABLE**, **DROP TABLE** и т. д.

Далее в книге приводятся примеры написания хранимых процедур, создаваемых с использованием SQL-инструкции **CREATE PROCEDURE**. Кодом листинга 18.12 при помощи SQL-инструкции **PROCEDURE** содается новый запрос в базе данных **фирма.mdb**. На рис. 18.16 показан созданный в результате выполнения этого кода запрос.

Листинг 18.12 Создание именованного объекта QueryDef из VBA-кода

```
1:  Sub ProcedureTest()
2:  'Тестирование SQL-инструкции PROCEDURE
3:
4:      Dim dbs As Database, rst As Recordset
5:      Dim qdf As QueryDef, strSql As String
6:
7:      Set dbs = OpenDatabase("c:\фирма.mdb")
8:
9:      strSql = "PROCEDURE Test1; " _
10:         & "SELECT field_1, " _
11:         & "field_2 FROM TestTable2 " _
12:         & "ORDER BY field_2;"
13:
14:     ' Создаииие именованного объекта QueryDef
15:     ' на основе SQL-инструкции
16:
17:     Set qdf = dbs.CreateQueryDef("НовыйЗапрос", strSql)
18:
19:     dbs.Close
20:
21: End Sub
```

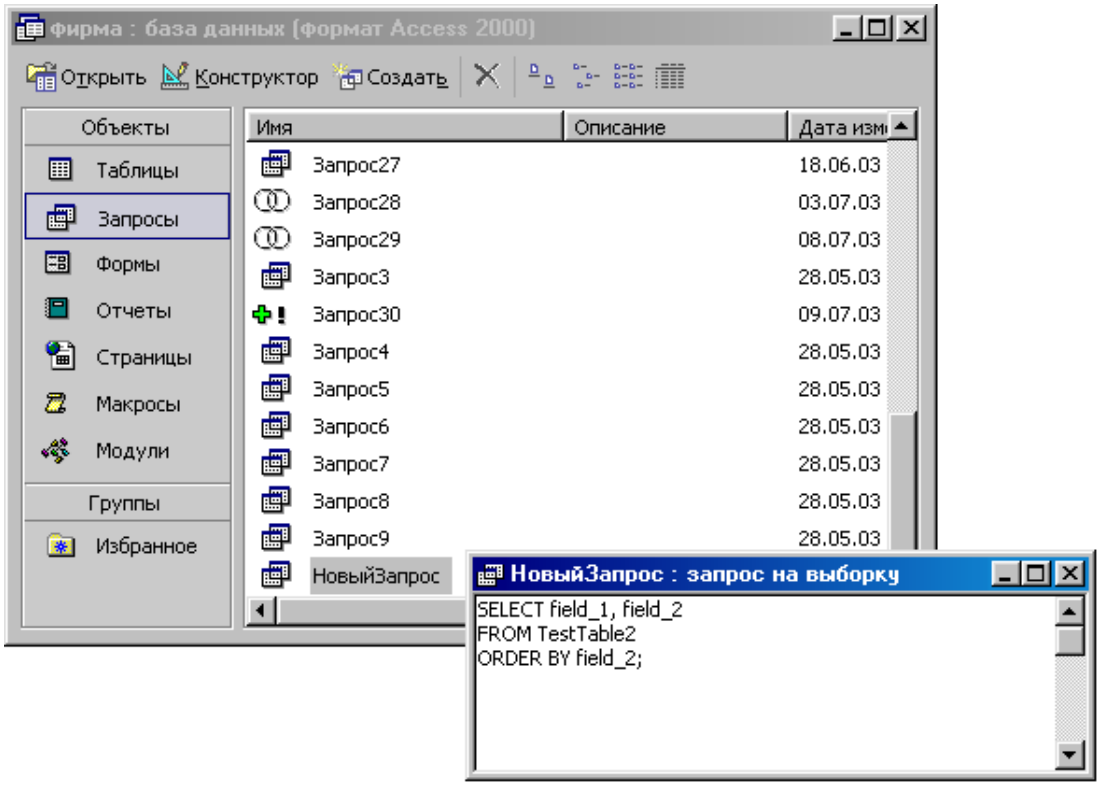

Рис. 18.16
Запрос, созданный в результате выполнения кода листинга 18.12