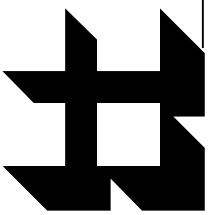

ГЛАВА 8

Работа с удаленными данными

Введение в технологию "клиент-сервер"

Базы данных Visual FoxPro могут быть расположены на сервере и использованы для работы в сети. В этом случае приложение, которое работает с этими базами данных, тоже располагается на сервере. Пользователь, запуская такое приложение, создает копию его на локальном компьютере. Возможна установка приложения и на локальном компьютере, в таком случае приложению должны быть известны пути к базам данных, находящихся на сервере. Такой сетевой вариант расположения баз данных называется технологией *"файл-сервер"*. Недостатки файл-серверных систем известны. Это и разрушение индексов в самый неподходящий момент, например, при выполнении срочных работ. При этом приходится прекращать работу и восстанавливать индексы. Это может быть и монопольный захват какого-либо сетевого файла одним из пользователей, делающий невозможной общую работу с этим файлом. При работе с данными используется навигационный способ доступа, при этом по сети передаются большие объемы данных, что является причиной перегруженности сети и снижения производительности. Кроме того, затрудняется соблюдение конфиденциальности. Из-за этих недостатков применение файл-серверных систем ограничено небольшими сетями. Для сетей с большим количеством пользователей более предпочтительна технология *"клиент-сервер"*.

Технологией "клиент-сервер" называется технология работы в вычислительной сети, при которой все вычисления выполняются на одном мощном центральном компьютере, называемом сервером, а запросы и результаты вычислений находятся на остальных компьютерах, называемых клиентами или рабочими станциями (хотя возможно и совмещение сервера и клиента на одном компьютере). В отличие от файл-серверной технологии, при которой на сервере только хранятся файлы, а их обработка ведется на локальных машинах, клиент-серверная технология позволяет обеспечить одновременную работу большего количества пользователей без потерь времени. Но понятие "клиент-сервер" более широкое, и относится не только к физическим устройствам, но и к программному обеспечению.

Клиент-серверная система может быть двух- или трехзвенной. Однако, независимо от варианта установки, базы данных размещаются на сервере. Доступ пользователей к серверу производится либо при помощи приложений с их компьютеров-клиентов (в двухзвенных системах), либо посредством приложений, выполняющихся на специальном компьютере, который называется сервером приложений (в трехзвенных системах).

При необходимости произвести обработку информации, хранящейся в базе данных, запущенное на компьютере пользователя клиентское приложение формирует запрос на языке SQL (название от начальных букв — Structured Query Language). Сервер базы данных принимает запрос и обрабатывает его. После обработки запроса на компьютер пользователя передается только результат. Сам же файл, в котором хранились данные, послужившие источником для обработки, остается незаблокированным для доступа самого сервера по запросам других пользователей.

В некоторых клиент-серверных СУБД существуют дополнительные механизмы, снижающие нагрузку на сеть. Это, например, хранимые процедуры — т. е. программы обработки данных, хранящихся в БД. В этом случае от пользователя к серверу передается лишь вызов функции с параметрами вызова. Таким образом, рабочее место пользователя упрощается, логика работы программы переносится на сервер. Пользовательское место становится всего лишь средством отображения информации. Все это означает дальнейшее снижение нагрузки на сеть и пользовательские рабочие станции.

При работе в клиент-серверной системе приложение должно:

- устанавливать соединение с сервером;
- формировать и отправлять запрос серверу;
- обрабатывать полученные данные;
- возвращать обработанные данные клиенту;
- завершать соединение с сервером.

Технология работы файл-серверного приложения довольно сильно отличается от клиент-серверного. Именно поэтому гораздо проще написать "с нуля" клиент-серверное приложение, чем переделывать файл-серверное.

Установка соединения с удаленными данными

Данные в информационной системе могут быть расположены не только в базах данных Visual FoxPro, но и на SQL-сервере, в таблицах Excel, базах данных Access и Oracle и т. д. Такие данные называются удаленными. При этом совершенно не имеет значения, где эти файлы расположены физически: они могут находиться как на сервере, так и на одном и том же компьютере с Visual FoxPro. Для доступа к удаленным данным используется стандартный протокол доступа, называемый ODBC (Open Database Connectivity interface) — открытый интерфейс взаимодействия с базами данных, разработанный Microsoft в 1991 г. Все базы данных и электронные таблицы имеют специальные драйверы ODBC, с помощью которых можно соединяться с ними и полу-

чать к ним доступ. Для организации такого "общения" нужно установить соответствующий драйвер. Драйверы ODBC для наиболее популярных баз данных и электронных таблиц входят в комплект VFP. Если вы выбрали опцию установки **Complete**, то драйверы установятся автоматически. Чтобы установить конкретные драйверы, выберите опцию установки **Custom** и отметьте требуемые драйверы ODBC (рис. 8.1).

Компания Microsoft бесплатно распространяет продукт под названием MDAC (Microsoft Data Access Components). По сути, это набор драйверов компании Microsoft для доступа к самым разным хранилищам баз данных. Таким образом, если у вас вдруг не оказалось необходимых драйверов, или они имеют не ту версию, следует скачать с сайта Microsoft последнюю версию MDAC и установить на своем компьютере.

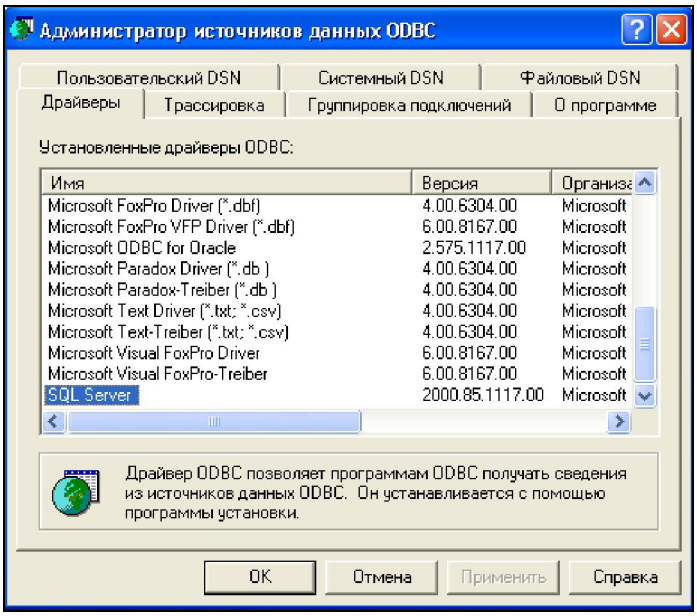

Рис. 8.1. Окно **Администратор источников данных ODBC**

Установка источника данных ODBC

Для установки соединения с сервером нужен только и исключительно драйвер.

Однако само соединение может обладать некоторыми настройками. Эти настройки — это указание драйвера, а также конкретного источника данных, к которому необходимо подключиться, плюс некоторые технические характеристики, определяющие работу только и исключительно этого самого соединения.

Существует возможность вручную указывать каждую из этих характеристик в момент или после установки соединения, но есть возможность сразу установить некоторый набор этих характеристик. Этим занимается DSN — Data Source Name.

Этот комплекс настроек получает имя, чтобы отличить его от другого комплекса настроек. То есть можно создать именованное соединение с уже установленными характеристиками, заданными в DSN, а можно прописать все необходимые характеристики в строке соединения, если установить переключатель в положение **Connection String**.

После установки драйвера необходимо установить источник данных ODBC. Источник данных можно установить следующим образом:

1. Выбрать на панели управления компьютера **Пуск | Панель управления | Администрирование | Источник данных (ODBC)** (рис. 8.2).

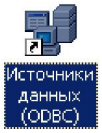

Рис. 8.2. Выбор пиктограммы "Источники данных (ODBC)" в папке Администрирование

2. Выберите на вкладке окна тип источников данных — пользовательский DSN, системный DSN или файловый DSN. Какой из них лучше выбрать? Пользовательский DSN увидит только тот пользователь, который его создал. Системный DSN увидят все пользователи данного компьютера. Файловый DSN используют для переноса данных с одного компьютера на другой.
3. Нажмите кнопку **Добавить**.
4. Перед вами появится окно создания нового соединения (рис. 8.3).

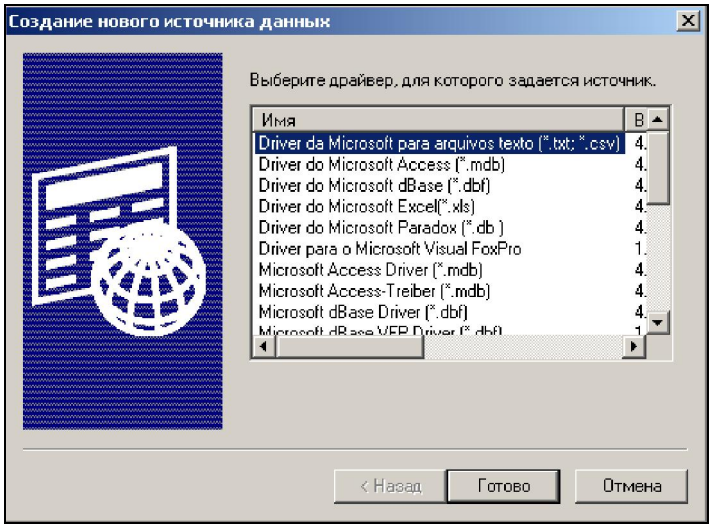

Рис. 8.3. Окно создания нового источника данных

Выберите нужный драйвер и нажмите кнопку **Готово**.

5. В следующем окне необходимо установить параметры соединения (рис. 8.4).

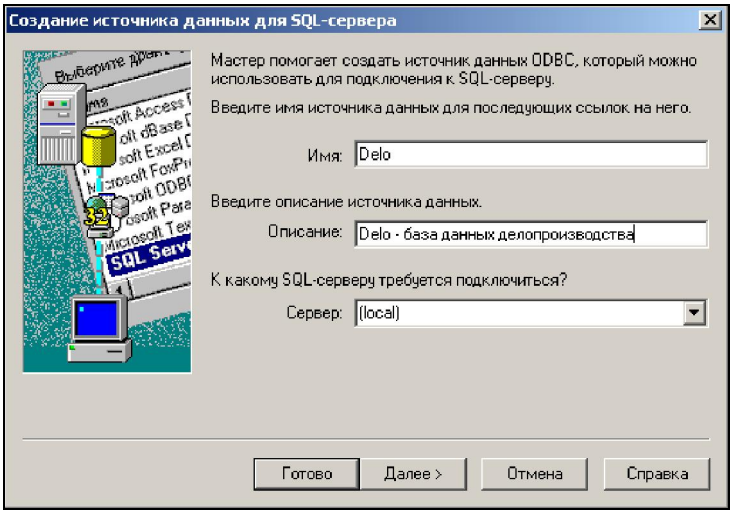

Рис. 8.4. Окно настройки параметров соединения

Вид окна зависит от типа создаваемого источника данных, однако для любого источника необходимо ввести имя, которое потом используется для ссылки на источник данных. В поле **Описание** можно ввести комментарий в виде краткого описания источника данных.

Кстати, если речь идет о создании DSN для MS SQL, нужно снять (не устанавливать) флажок в настройке **Использовать национальные настройки** (**Use regional settings when outputting currency, numbers, dates and times** в английской редакции) (рис. 8.5).

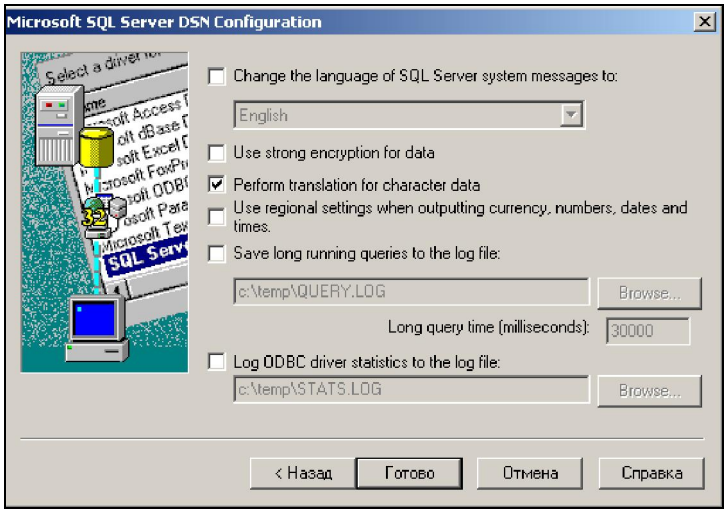

Рис. 8.5. Окно настройки DSN Configuration

В противном случае появляются проблемы с корректным отображением числовых данных, поскольку добавляется разделитель между тройками цифр. Например, число 123456 будет выглядеть как 123 456. Для Visual FoxPro это означает символ, отличный от числа, поэтому число будет прочитано как 123. По умолчанию флажок не установлен, и менять его не следует.

Именованное соединение

Именованное соединение хранится в базе данных и позволяет ссылаться на него по имени при создании удаленного соединения.

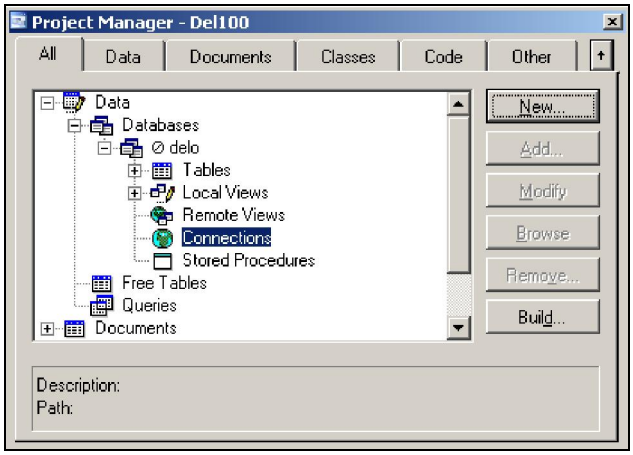

Рис. 8.6. Вкладка **Connections** в окне Менеджера проектов

Для создания именованного соединения выполните одно из следующих действий:

1. В диспетчере проектов выберите вкладку **Connections** (рис. 8.6).
2. Откройте базу данных и используйте команду `CREATE CONNECTION`, чтобы открыть конструктор соединений.
3. Можно использовать команду `CREATE CONNECTION` с именем соединения в качестве параметра.

Любой из этих способов вызовет окно конструктора соединений для задания параметров именованного соединения (рис. 8.7).

Область **Specify data source** (Определить источник данных) содержит переключатели **Data source**, **userid**, **password** (Источник данных, идентификатор пользователя, пароль) и **Connection string** (Строка соединения), определяющие способ соединения с источником данных. При выборе первой опции в области отображаются поля ввода (табл. 8.1).

При выборе переключателя **Connection string** (Строка соединения) вместо приведенных выше полей ввода отображается только одно поле ввода **Connect string** (Строка соединения), в котором можно ввести строку соединения с источником данных.

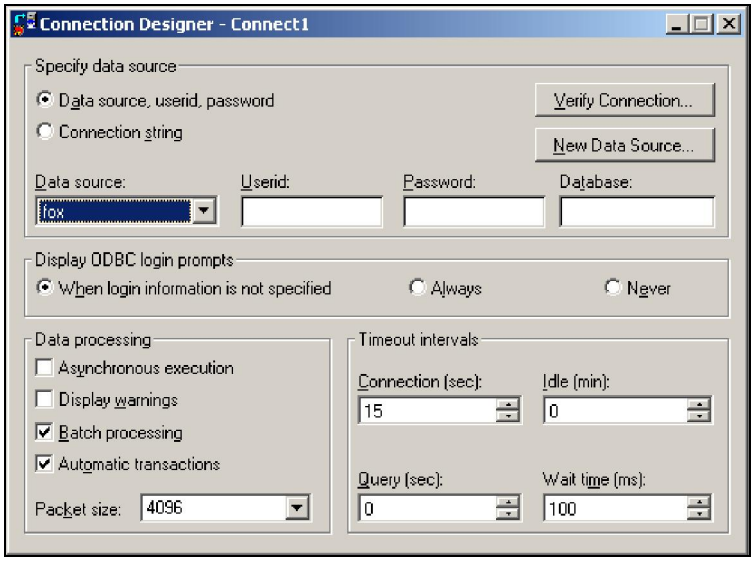

Рис. 8.7. Окно конструктора соединений

Таблица 8.1. *Область* ***Specify data source***

Наименование	**Описание**
Data source	Имя источника данных
Userid	Идентификатор пользователя
Password	Пароль пользователя
Database	Имя базы данных

Кнопка **Verify Connection…** (Проверить соединение) предназначена для проверки соединения с указанным источником данных. Кнопка **New Data Source…** (Новый источник данных) позволяет определить новый источник данных непосредственно из конструктора соединений.

Область **Display ODBC login prompts** содержит три опции, которые определяют порядок открытия диалогового окна **Data Source login** (доступ к источнику данных) (табл. 8.2).

Таблица 8.2. *Порядок открытия окна* ***Data Source Login***

Наименование	Описание
When login information is not specified	Окно будет открыто в случае, если логин и пароль пользователя не найдены в определении именованного соединения

Таблица 8.2 *(окончание)*

Наименование	Описание
Always	Окно открывается всегда
Never	Окно не открывается никогда

Область **Data processing** содержит ряд флажков, задающих параметры соединения (табл. 8.3).

Таблица 8.3. *Параметры соединения*

Наименование	Описание
Asynchronous execution	Задает асинхронное выполнение
Display warnings	Позволяет показывать предупреждения
Batch processing	Задает пакетный режим обработки данных
Automatic transactions	Задает автоматическую обработку транзакций
Packet size	Задает размер пакета данных, передаваемого от сервера баз данных

Функции `SQLCOLUMNS()`, `SQLEXEC()`, `SQLMORERESULTS()`, `SQLTABLES()` могут выполняться как в синхронном, так и асинхронном режиме. При асинхронном выполнении функция должна вызываться до тех пор, пока она не вернет результат, отличный от нуля (1 в случае удачного выполнения и –1 или –2 в случае ошибки).

Если результат равен 0, то функция продолжает выполняться. По умолчанию устанавливается синхронное выполнение.

При использовании механизма пакетных обновлений (Batch processing) любые изменения, вносимые пользователем, накапливаются в локальной памяти. Позже полный пакет этих изменений за одну операцию может быть внесен в базу данных. Очевидно, что подобный подход обеспечивает выигрыш в производительности, однако существуют и другие преимущества, делающие его удобным. В частности, при использовании механизма пакетных обновлений пользователь может выполнять изменения даже тогда, когда он отключен от базы данных.

Область **Timeout intervals** содержит опции, которые устанавливают временные параметры обработки данных (табл. 8.4).

Таблица 8.4. *Опции установки временных параметров обработки данных*

Наименование	Описание
Connection (sec)	Интервал тайм-аута соединения в сек
Query (sec)	Интервал тайм-аута запроса в сек
Idle (min)	Интервал деактивации соединения в мин
Wait time (ms)	Интервал ожидания выполнения оператора SQL в мс

Для определения значений параметров области **Data processing** можно использовать функцию `DBSETPROP()` или `SQLSETPROP()`.

По умолчанию соединению присваивается имя Connect1.

Созданные соединения отображаются в проекте на вкладке **Database**.

Как собственно DSN, так и объект `Connection` — это все некоторые предварительные настройки. Сами по себе они никакого соединения не устанавливают. Однако сделанные в них настройки используются при установке реального соединения. Впрочем, в Visual FoxPro существует возможность установки соединения и минуя сделанные в них настройки, через команду `SQLStringConnect()`.

Если новый объект `Remote View` создается с использованием дизайнера, то объект `Connection` должен существовать. Однако реально есть возможность работать с `Remote View` и без предварительно созданного объекта `Connection`. Хотя это сложнее в реализации.

Программным путем можно установить соединение с помощью команды

```
CREATE CONNECTION <имя соединения>
    DATASOURCE <имя источника данных>
    USERID <идентификатор пользователя>
    PASSWORD <пароль>
    CONNSTRING <строка соединения>
```

Удаленные представления (*Remote Views*)

Настройка параметров удаленных представлений

Опции принимаемых по умолчанию параметров удаленных представлений (табл. 8.5 и 8.6) можно настроить на вкладке **Tools | Options | Remote Data** (рис. 8.8).

Таблица 8.5. *Параметры удаленного представления*

Параметр	Описание
Share connection	В представлениях будет использовано текущее совместно используемое соединение
Fetch memo	Memo-поле не будет выбираться из источника данных, пока не будет активизировано в представлении данных
SQL updates: Criteria	Позволяет указать критерии обновления данных

SQL updates: Method	Задает метод обновления данных
Records to fetch at a time	Задает количество выбираемых одновременно записей из источника данных
Maximum records to fetch	Задает количество записей, возвращаемых в представлении
Use memo for fields	Позволяет использовать перевод символьных данных в memo-поля
Records to batch update	Задает количество обновляемых одновременно записей

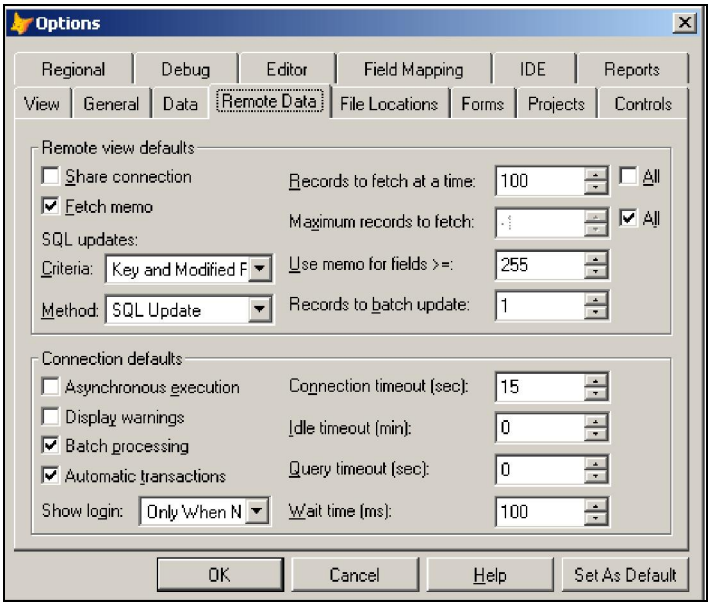

Рис. 8.8. Настройка параметров удаленных представлений и соединений

Таблица 8.6. *Параметры именованного соединения*

Параметр	Описание
Asynchronous execution	Задает признак асинхронной обработки
Display warnings	Позволяет выдавать предупреждающие сообщения
Batch processing	Включает режим пакетной обработки
Automatic transactions	Задает управление транзакциями в удаленной таблице
Show login	Показывает окно приглашения
Connection timeout (sec)	Задает интервал времени в секундах, в течение которого ожидается ответ сервера на установку соединения. При превышении интервала генерируется ошибка

Idle timeout (min)	Задает допустимое время простоя, после которого соединение будет прекращено. Но в случае поступления запроса по истечении интервала простоя, VFP автоматически пытается восстановить соединение
Query timeout (sec)	Задает интервал времени в секундах, в течение которого ожидается ответ на запрос от сервера. В случае превышения интервала генерируется ошибка
Wait time (ms)	Задает интервал времени, во время которого производятся проверки на завершение запроса

Создание удаленного представления

1. Открыть базу данных.
2. Вызвать мастер удаленных представлений **Wizards | Query | Remote view wizard**.
3. В диспетчере проектов выбираем **Databases | Remote View | New** либо `CREATE SQL VIEW (share)` с предложением `REMOTE` и/или `CONNECTION`.

Сквозной запрос (SQL pass-through)

Сквозной запрос SQL (SQL pass-through) обеспечивает прямой доступ к внешнему серверу с помощью функций сквозного запроса SQL. Технология сквозных запросов SQL предполагает, что оператор SQL посылается непосредственно на сервер. Поскольку операторы сквозного запроса SQL выполняются на сервере, они позволяют существенно повысить производительность работы приложения. В табл. 8.7 приводятся сравнительные характеристики технологии использования внешних представлений и сквозных запросов SQL.

Таблица 8.7. *Сравнение внешних представлений и сквозных запросов SQL*

Удаленное представление	Сквозные запросы SQL
Базируется на применении оператора SQL SELECT	Позволяет применять любые SQL-операторы, в том числе вложенные и содержащие в запросе функции или предложение UNION
Невозможность выполнять хранимые процедуры	Возможно выполнение хранимых процедур
Не обеспечивает управление транзакциями, позволяет только имплицитные транзакции	Обеспечивает явное управление транзакциями

Таким образом, технология сквозных запросов SQL предоставляет следующие преимущества по сравнению с удаленными представлениями:

- вы можете пользоваться функциональными возможностями сервера, например, хранимыми процедурами и встроенными функциями на базе сервера;

- вы можете использовать расширенные возможности SQL, поддерживаемые сервером, а также команды определения данных, администрирования сервера и защиты;
- вы получаете более полный контроль над операторами `Update`, `Delete` и `Insert` сквозных запросов SQL;
- вы расширяете возможности контроля над внешними транзакциями.

Сквозные запросы SQL имеют и ряд недостатков. По умолчанию сквозной запрос SQL всегда возвращает необновляемые данные в курсор представления. Курсор можно сделать обновляемым, установив соответствующие свойства с помощью функции `CURSORSETPROP()`. Напротив, обновляемое удаленное представление обычно не требует установки свойств для обновления данных.

Команды SQL можно вводить прямо в окне команд или в программе, не пользуясь графическим конструктором представлений.

Независимо от того, какая технология используется — внешние представления или сквозные запросы SQL, вы можете выполнять запросы и обновлять внешние данные. Практика показывает, что обычно используются как внешние представления, так и сквозные запросы SQL (табл. 8.8).

Таблица 8.8. *Функции для работы с удаленными данными*

Функция	Назначение	Синтаксис
`SQLCONNECT()`	Устанавливает соединение с источником данных для сквозных запросов	`SQLCONNECT`(имя_источника_данных, идентификатор пользователя, пароль, имя соединения)
`SQLSTRINGCONNECT()`	Устанавливает соединение с источником данных, используя синтаксис строки соединения ODBC	`SQLSTRINGCONNECT`([флаг создания многопользовательского приложения] | [строка_соединения [, многопользовательское использование]])
`SQLDISCONNECT()`	Разрывает соединение с источником данных ODBC	`SQLDISCONNECT`(дескриптор_соединения)
`SQLCANCEL()`	Отменяет выполняемый SQL-запрос на активном соединении	`SQLCANCEL`(номер_активного_соединения)
`SQLEXEC()`	Выполняет сквозной sql-запрос для активного соединения	`SQLEXEC`(дескриптор_соединения, SQL_оператор, [имя курсора])
`SQLPREPARE()`	Подготавливает сквозной SQL-запрос для активного соединения	`SQLPREPARE`(дескриптор_соединения, SQL_оператор, [имя курсора])

SQLMORERESULTS()	Выбирает в курсор еще одно результирующее множество. Возвращает 0, если оператор, создающий множество, еще выполняется	SQLMORERESULTS(дескриптор соединения)
SQLCOMMIT()	Запрашивает фиксацию транзакции	SQLCOMMIT(номер_соединения)
SQLROLLBACK()	Запрашивает откат транзакции	SQLROLLBACK(дескриптор_соединения)
SQLCOLUMNS()	Сохраняет в курсоре список столбцов и информацию о каждом из них	SQLCOLUMNS(номер_активного_соединения, имя_удаленной_таблицы ['FOXPRO'] ['NATIVE'] [имя_курсора])
SQLTABLES()	Сохраняет в курсоре имена таблиц в источнике данных	SQLTABLES(дескриптор_соединения [тип таблицы] [имя курсора])
SQLSETPROP()	Устанавливает свойства активного соединения	SQLSETPROP(дескриптор_соединения, установка, значение_для_установки)

Таблица 8.8 *(окончание)*

Функция	Назначение	Синтаксис
SQLGETPROP()	Получает свойства активного соединения	SQLGETPROP(дескриптор_соединения, установка)

Если вы настраиваете соединение программным способом через команду SQLConnect() или команду SQLStringConnect(), то в случае ошибки создания соединения возникает системное окно для настройки реквизитов соединения. А как же быть, если такое окно вам вовсем не нужно?

Если вы используете для создания соединения объект Connection базы данных, то откройте этот объект на редактирование и сделайте следующие настройки:

- в разделе **Display ODBC login prompts** установите переключатель в положение **Never**;
- убедитесь, что в разделе **Data Procesing** переключатель **Display Warnings** выключен.

Если вы настраиваете соединение исключительно программными средствами, то установить нужные значения этих настроек можно, используя функцию SQLSetProp(). Это настройки **DispLogin** и **DispWarnings**.

Однако, как видно из синтаксиса функции SQLSetProp(), она делает настройки для конкретного соединения. Первым параметром идет номер уже созданного соединения. Но ведь соединения еще нет. Его только собираются создать.

Чтобы преодолеть это противоречие, используют соединение с номером 0. Физически — это вообще не соединение. По сути, это будут настройки для любого нового соединения, поскольку номером 0 обозначают первое не занятое значение номера

соединения. На уже созданные соединения эти настройки не распространяются. Получается примерно такой код:

```
* Сохранение текущих значений настроек
 LOCAL llDispWarning, lnDispLogin
 llDispWarning=SQLGetProp(0,'DispWarning')
 lnDispLogin=SQLGetProp(0,'DispLogin')

* Установка этих настроек в значения, препятствующие выдаче системных сообщений
 SQLSetProp(0,'DispWarning',.F.)
 SQLSetProp(0,'DispLogin',3)

* Формируем строку соединения для MS SQL сервера
 LOCAL lcStringConnect
 lcStringConnect='DRIVER=SQL Server;'+;
                 'SERVER=Имя_Сервера;'+;
                 'UID=Логин_Пользователя;'+;
                 'PWD=Пароль_Пользователя;'+;
                 'DataBase=Имя_базы_данных'

* Собственно попытка создания соединения
 lnNewConnection = SQLStringConnect(m.lcStringConnect)
 IF m.lnNewConnection <= 0
       * Произошла ошибка в момент установки соединения. Выясняем причину
       LOCAL laError(1)
       =AERROR(laError)
       * Анализ массива laError для уточнения причины ошибки
 ELSE
       * Соединение успешно установлено
       * и его номер записан в переменную lnNewConnection
 ENDIF

* Восстановление исходных настроек
 SQLSetProp(0,'DispWarning',m.llDispWarning)
 SQLSetProp(0,'DispLogin',m.lnDispLogin)
```

MS SQL Server

Microsoft SQL Server в настоящее время является одним из самых популярных серверов реляционных баз данных. Он может быть установлен как в сети для общего использования большим количеством пользователей, так и в качестве "desktop system" — автономного сервера для одного или нескольких компьютеров. В своем составе SQL Server имеет средства создания баз данных, работы с информацией баз данных, перенесения данных из других систем и в другие системы, резервного копирования и восстановления данных, развитую систему транзакций, систему репликации данных, реляционную подсистему для анализа, оптимизации и выполнения запросов клиентов, систему безопасности для управления правами доступа к объектам

базы данных и пр. Система не содержит средств разработки клиентских приложений. Высокая популярность обусловлена целым рядом функциональных возможностей сервера.

Поддержка расширенной памяти

Для поддержки больших адресных пространств SQL Server может использовать AWE — Address Windowing Extensions. Поддержка осуществляется для серверов под управлением Windows 2000 Advanced Server (до 8 Гбайт) и для Windows 2000 Datacenter — до 64 Гбайт.

Установка многократных экземпляров SQL Server

На одном и том же сервере могут быть установлены несколько экземпляров SQL Server. Это позволяет группировать типичные операции на разных экземплярах сервера.

Операции по поддержке баз данных

SQL Server 2000 позволяет сохранять ссылочную целостность при изменении данных в таблице.

Анализатор производительности

В SQL Server имеется новое инструментальное средство — Performance Analyzer, которое служит для сбора данных о производительности системы, их просмотра и анализа.

SQL Server Profiler

Этот инструмент необходим для трассировки и выявления "узких мест" приложения.

SQL Query Analyzer

Это мощное средство — браузер для просмотра объектов (баз данных, таблиц и т. д.).

Средства для администрирования

SQL Server имеет в своем составе ряд средств, облегчающих работу администратора.

К таким средствам можно отнести и ведение журнала транзакций, и инструментарий для сохранения и восстановления баз данных. Вы можете сконфигурировать расписание для каждого шага.

Перенос данных из VFP на SQL Server

В последние годы необходимость переноса данных из VFP на удаленные источники данных стала возникать довольно часто. Одна из причин такого явления — растущий объем информации, который за короткое время "вырастает" из отпущенного VFP ограничения на размер таблицы в 2 Гбайт. Для переноса данных из таблиц VFP на SQL Server служит мастер наращивания (Upsizing). Он позволяет перемещать ло-

кальные данные на удаленный сервер. Мастер можно вызвать из главного меню: **Tools | Wizards | Upsizing**. Перед тем как использовать мастер наращивания, нужно установить драйвер ODBC для выбранной базы данных, определить источник данных для связи с базой данных и именованное удаленное соединение. Следует учесть, что многие функции VFP не поддерживаются SQL-сервером. Выражения, содержащиеся в правилах проверки полей и записей, мастер наращивания пытается преобразовать в выражения Transact-SQL (табл. 8.9).

Таблица 8.9. *Преобразование выражений VFP в SQL*

Выражение VFP	Выражение SQL Server
.T.	1
.F.	0
#	<>
.AND.	AND
.NOT.	NOT
.NULL.	NULL
.OR.	OR

Таблица 8.9 *(окончание)*

Выражение VFP	Выражение SQL Server
=<	<=
=>	>=
ASC()	ASCII()
AT()	CHARINDEX()
CDOW()	DATENAME(dw,...)
CHR()	CHAR()
CMONTH()	DATENAME()
CTOD()	CONVERT(datetime,)
CTOT()	CONVERT(datetime,)
DATE()	GETDATE()
DATETIME()	GETDATE()
DAY()	DATEPART(dd,)
DOW()	DATEPART(dw,)
DTOC()	CONVERT(varchar,)
DTOR()	RADIANS()

DTOT()	CONVERT(datetime,)
HOUR()	DATEPART(HH,)
LIKE()	PARTINDEX()
MINUTE()	DATEPART(mi,)
MONTH()	DATEPART(mm,)
MTON()	CONVERT(money,)
NTOM()	CONVERT(float,)
RTOD()	DEGRESS()
SUBSTR()	SUBSTRING()
TTOC()	CONVERT(datetime,)
TTOD()	CONVERT(datetime,)
YEAR()	DATEPART(yy,)

Некоторые из функций имеют тот же вид в VFP, что и в SQL. Это такие функции, как CEILING(), LOG(), LOWER(), LTRIM(), RIGHT(), RTRIM(), SOUNDEX(), SPACE(), STR(), STUFF(), UPPER().

На первом этапе работы мастера наращивания необходимо выбрать и открыть локальную базы данных, которую вы хотите перенести на SQL Server (рис. 8.9).

Затем необходимо указать источник данных (рис. 8.10).

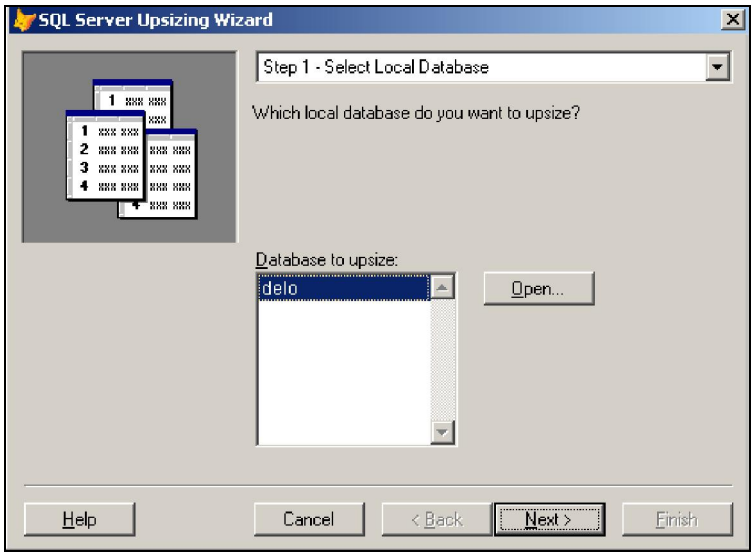

Рис. 8.9. Работа мастера наращивания. Шаг 1 — выбор локальной базы данных

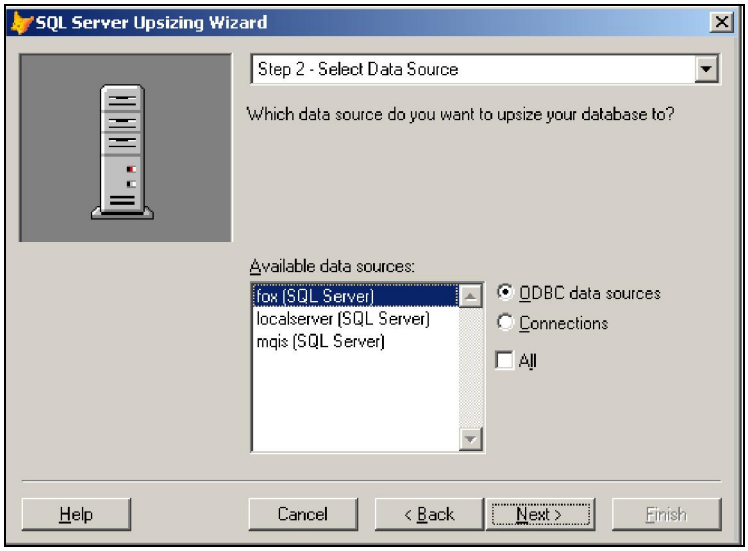

Рис. 8.10. Мастер наращивания. Шаг 2 — определение источника данных

Если используется именованное соединение, то мастер наращивания будет связывать именованное соединение со всеми удаленными представлениями. Если вместо имени соединения использовать имя источника данных, мастер наращивания создает именованное соединение с именем Upsize1, Upsize2 и т. д. По умолчанию открывается окно **ODBC Login**. Если же используется именованное соединение с сохраненным паролем, мастер регистрирует вас на выбранном SQL Server.

На третьем шаге производится выбор таблиц для переноса (рис. 8.11).

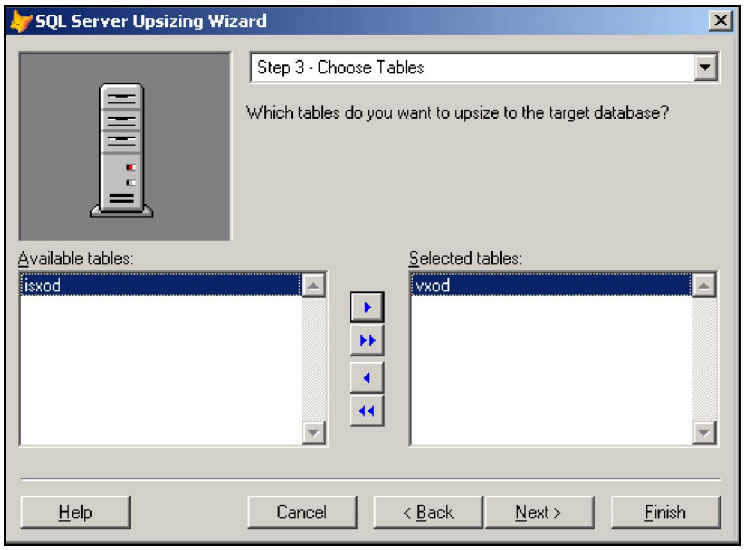

Рис. 8.11. Работа мастера наращивания. Шаг 3

Все открываемые таблицы должны быть открыты только в монопольном режиме, в противном случае информация не может быть перенесена. При переносе таблиц имена полей и типы данных преобразуются в поля SQL Server. Мастер показывает установленное по умолчанию преобразование типов данных и позволяет на четвертом шаге при необходимости его изменять с помощью комбинированного списка **Server Type** (рис. 8.12).

На пятом шаге определяется база данных — приемник. Данные могут быть перенесены как в новую базу данных, так и в существующую (рис. 8.13).

На шестом и седьмом шаге определяются устройство и размер базы данных и журнала транзакций. Минимальный размер новой базы данных — 2 Мбайт. Мастер наращивания создает все новые устройства на одном и том же физическом диске, на котором расположен SQL Server.

При использовании мастера наращивания желательно иметь права на создание таблиц на сервере.

На восьмом и девятом шаге определяются порядок переноса индексов, отношения, правила проверки и значения по умолчанию (рис. 8.14 и 8.15).

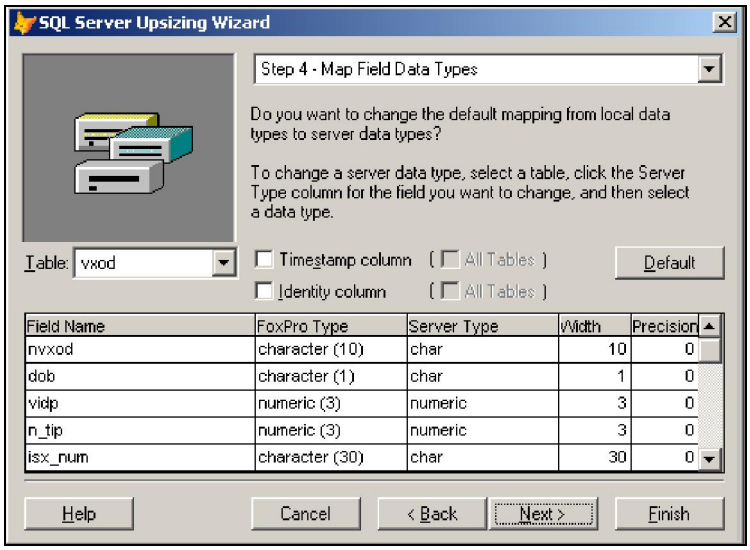

Рис. 8.12. Мастер наращивания. Шаг 4

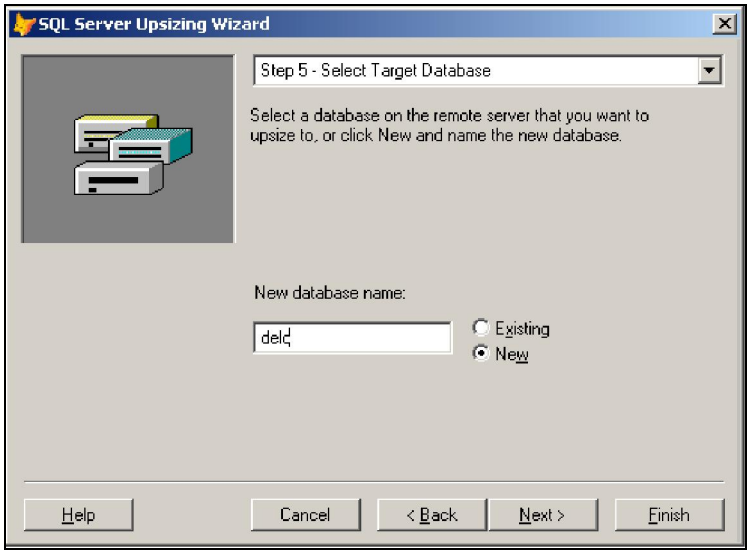

Рис. 8.13. Мастер наращивания. Шаг 5

После наращивания мастер создает отчет о переносе данных.

Однако возможности мастера ограничены. Он не только делает перенос данных из больших таблиц очень медленно (операция может занять несколько часов), но и зачастую неверно. Как правило, верно переносятся таблицы, содержащие числовые данные.

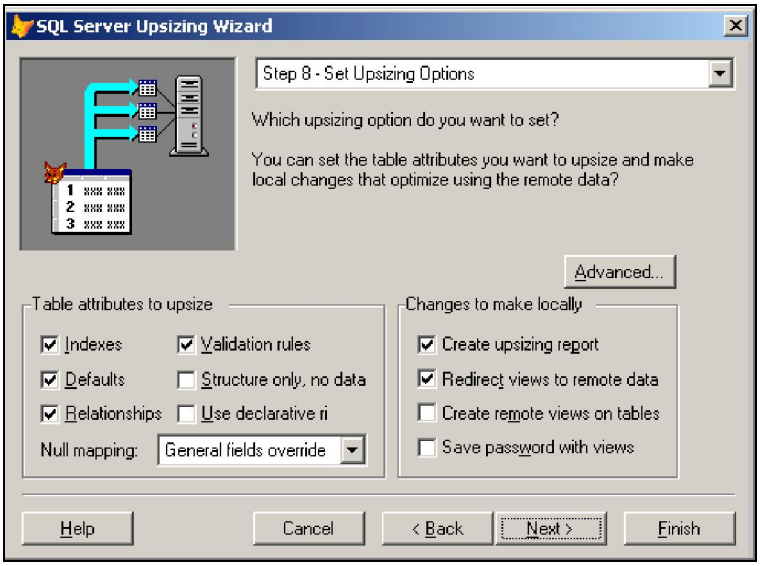

Рис. 8.14. Мастер наращивания. Шаг 8

Рис. 8.15. Мастер наращивания. Шаг 9

Поэтому чаще всего программисты пишут свои собственные программы для переноса данных. Один из вариантов программы любезно предоставлен Владимиром Журавлевым (листинг 8.1).

```
Set Exclusive On
Set Talk On
Set Deleted On
Set Date German
Set Century On
Store Sqlstringconnect("dsn=fox;uid=sa;pwd='',database='primer1'") To Con
=SQLExec(Con,"")
If Con < 0
        Wait Window "Нет соединения" Nowait
Else
        Wait Window "Соединение установлено" Nowait
Endif
Open Database primer Exclusive
If ! Used('prim1')
        Use prim1 Exclusive
Else
        Select prim1
Endif
Select prim1
Go Top
m.sql = 'insert into primer..prim1('
m.sql1 =') values ('
For ii =1 To Fcount()
        m.sql =m.sql+Iif(ii>1,',','')+Fields(ii)
        m.sql1 =m.sql1+Iif(ii>1,',','')+'?prim1.'+Fields(ii)
Endfor
m.res = m.sql+m.sql1+')'
Wait Window "m.res="+m.res
SELECT prim1
Go Top
SCAN
*       ? prim1.Id
*       ? prim1.naim
        a1=SQLExec(Con, m.res)
        If a1<0
                Wait Window "Ошибка Insert"
        Endif
ENDSCAN
SQLDisconnect(Con)
```

На компакт-диске программа находится в CHAR8\perenos.prg.

Надо учитывать, что типы данных SQL Server несколько отличаются от типов данных Visual FoxPro. Для того чтобы оценить эту разницу, можно создать в Visual FoxPro

таблицу с полями всех имеющихся типов данных VFP, а затем попытаться перенести ее на SQL Server (табл. 8.10).

Таблица 8.10. *Типы данных. Соответствие данных VFP и SQL*

Обозначение	Тип VFP	Тип SQL
C	Character	Char
Y	Currency	Money
D	Date	DateTime
T	DateTime	DateTime
B	Double	Float
F	Float	Float
G	General	Image
I	Integer	Int
L	Logical	Bit (0 или 1)
M	Memo	Text
M (binary)	Memo binary	Image
C (binary)	Char (binary)	Binary
N	Numeric	Float

Хочется подчеркнуть: в MS SQL нет такого типа данных, как `Date`, поэтому все преобразования `Date`, по сути, означают перевод в `DateTime`.

В каких случаях переход на SQL Server оправдан?

1. Если некоторые из ваших таблиц превышают (или могут в перспективе превысить) размер 2 Гбайт, это уже тревожный звонок, потому что ограничение на размер файлов в VFP составляет именно 2 Гбайт. При приближении к этим критическим размерам и достаточно большом количестве пользователей Visual FoxPro начинает несомненно уступать SQL Server по производительности в целом.
2. Существует показатель, по которому SQL Server предпочтительнее DBF — надежность. Перейдя к технологии "клиент-сервер", вы забудете о разрушенных индексных и memo-файлах.
3. Если вы создаете тиражируемые продукты, применение клиент-серверных технологий, скорее всего, привлечет к вам заказчиков.

Способы соединения с сервером

При разработке клиент-серверного приложения с помощью Visual FoxPro мы можем использовать одну из двух альтернативных технологий доступа к данным:

- удаленное представление (**Remote view**);
- сквозной запрос (**SQL pass-through**).

Первый способ предполагает создание объекта `Connection` в проекте в составе базы данных, а также удаленного представления (**Remote View**) с заданным параметром обновления данных (`Send SQL Updates`).

Второй способ можно выполнять по-разному: можно посылать на сервер прямые, так называемые "сквозные" запросы, а можно создать объект `CursorAdapter` для связи с базой SQL Server, настроить его параметры и с созданными временными таблицами работать как с обычными таблицами баз данных Visual FoxPro в режиме оптимистической буферизации данных.

Использование команд T-SQL

В MS SQL используется язык Transact-SQL (T-SQL), который является расширением языка SQL. В T-SQL имеются все возможности SQL, но, кроме того, имеются расширенные функции, системные хранимые процедуры и конструкции для программирования. Язык T-SQL обеспечивает выполнение ряда важных функций: создание объектов базы данных, управление объектами, пополнение таблиц базы данных новыми данными, обновление данных, уже имеющихся в таблицах, выполнение запросов, управление доступом пользователей к базе данных, а также осуществление общего администрирования базы данных (табл. 8.11).

Язык SQL делится на несколько частей:

- Операторы определения данных *(Data Definition Language, DDL)*.

  DDL включает следующие команды:

  ```
  ALTER TABLE
  CREATE TABLE
  DROP TABLE
  ```

  Язык определения данных (DDL) является частью SQL, дающей пользователю возможность создавать различные объекты базы данных и переопределять их структуру, например, создавать или удалять таблицы;

- Операторы манипуляции данными *(Data Manipulation Language, DML)*.

  DML включает следующие команды:

  ```
  INSERT
  DELETE
  UPDATE
  ```

  Язык манипуляции данными (DML) является частью SQL, дающей пользователю возможность манипулировать данными внутри объектов реляционной базы данных;

- Язык запросов к данным *(Data Query Language, DQL)*.

  DQL содержит только один оператор, но очень важный:

  ```
  SELECT
  ```

  Эта команда, имеющая множество опций и необязательных параметров, используется для построения запросов к реляционным базам данных. С ее помощью можно

конструировать запросы любой сложности — от самых общих до очень специальных и от самых простых до невероятно сложных.

- Операторы определения доступа к данным *(Data Control Language, DCL)*.

```
ALTER PASSWORD
GRANT
REVOKE
CREATE SYNONYM
```

Команды управления данными (DCL) позволяют управлять доступом к информации, находящейся внутри базы данных. Как правило, они используются для создания объектов, связанных с доступом к данным, а также служат для контроля над распределением привилегий между пользователями;

- Команды администрирования данных.

```
START AUDIT
STOP AUDIT
```

Команды администрирования данных дают пользователю возможность выполнять аудит и анализ операций внутри базы данных. Эти команды могут также помочь при анализе производительности системы данных в целом;

- Команды управления транзакциями.

В дополнение ко всем уже рассмотренным категориям команд есть еще команды, позволяющие пользователю управлять транзакциями базы данных.

```
COMMIT
ROLLBACK
SAVEPOINT
SET TRANSACTION
```

Краткий справочник по Transact-SQL

В табл. 8.11 представлены команды языка T-SQL.

Таблица 8.11. *Команды языка T-SQL*

Команда	Использование	Описание
ALTER DATABASE	ALTER DATABASE *database_name* [ON [DEFAULT | *database_device*][=*size*] [,*database_device*[=*size*]]...] [FOR LOAD]	Позволяет производить различные операции с базой данных после ее создания
ALTER TABLE	ALTER TABLE [*database*.[*owner*].]*table_name* [WITH NOCHECK] [ADD {*col_name column_properties column_constraints*] |[[,]*table_constraint*]} [,{*next_col_name*|*next_table_constraint*}]...] | DROP [CONSTRAINT] constraint_name[,constraint_name]	Программно изменяет структуру таблицы

CREATE DATABASE	CREATE DATABASE *database_name* [ON [DEFAULT|*database_device*][= *size*] [,*database_device*][= *size*]]...] [LOG ON *database_device*[= *size*] [,*database_device*[= *size*]...] [FOR LOAD]	Создает БД и журнал транзакций на указанных *devices*, *size* — размер в мегабайтах

Таблица 8.11 *(продолжение)*

Команда	Использование	Описание
CREATE TABLE	CREATE TABLE [*database*.[*owner*].]*table_name* ({*col_name column_properties* [*constraint* [*constraint* [...*constraint*]]] | [[,] *constraint*]} [[,] {*next_col_name* | *next_constraint*}...]) [ON segment_name]	Создает постоянную или временную таблицу
DROP TABLE	DROP TABLE [[*database*.]*owner*.]*table_name* [, [[database.]owner.]table_name...]	Уничтожает таблицу
DELETE	DELETE [FROM] {*table_name* | *view_name*} [FROM {*table_name* | *view_name*} [, {*table_name* | *view_name*}]...] [..., {*table_name16* | *view_name16*}]] [WHERE *clause*]	Оператор DELETE удаляет записи по одной строке, занося данные об этом в журнал транзакций
INSERT	INSERT [INTO] {*table_name* | *view_name*} [(*column_list*)] {DEFAULT VALUES | values_list | select_statement}	Добавляет запись в таблицу
UPDATE	UPDATE {*table_name* | *view_name*} SET [{*table_name* | *view_name*}] {*column_list* | *variable_list* | *variable_and_column_list*} [, {*column_list2* | *variable_list2* | *variable_and_column_list2*} ... [, {*column_listN* | *variable_listN* | *variable_and_column_listN*}]] [FROM {*table_name* | *view_name*} [, {*table_name* | *view_name*}]...] [..., {*table_name16* | *view_name16*}]] [WHERE clause]	Служит для обновления данных

SELECT	SELECT [ALL | DISTINCT] *select_list* [INTO [*new_table_name*]] [FROM {*table_name* | *view_name*}[(*optimizer_hints*)] [[, {*table_name2* | *view_name2*}[(*optimizer_hints*)] [..., {*table_name16* | *view_name16*}[(*optimizer_hints*)]]] [WHERE *clause*] [GROUP BY *clause*] [HAVING *clause*] [ORDER BY *clause*] [COMPUTE *clause*] [FOR BROWSE]	Производит выборку данных из БД или присваивает значения переменным, причем выборка данных и присваивание значений не могут производиться одновременно

Таблица 8.11 *(окончание)*

Команда	Использование	Описание
GRANT	GRANT {ALL | *permission_list*} ON {*table_name* [(*column_list*)] | *view_name* [(*column_list*)] | *stored_procedure_name* | *extended_stored_procedure_name*} TO {PUBLIC | *name_list*}	Назначает права пользователям
REVOKE	REVOKE {ALL | *permission_list*} ON {*table_name* [(*column_list*)] | *view_name* [(*column_list*)] | *stored_procedure_name* | *extended_stored_procedure_name*} FROM {PUBLIC | *name_list*	Изменяет права пользователей

Пример 1

```
lnRetval = SQLExec(THIS.nConn, lcSQL, "Person")
        IF lnRetval > 0
                IF RECCOUNT("Person") > 0
                        this.GenHTMLTable("Person")
                ELSE
                        this.GenHTMLNoRecords()
                ENDIF
                USE IN Person
        ELSE
                this.GenHTMLErrorState()
        ENDIF
```

Пример 2

```
lcCom = "insert into person("
lcCom = lcCom +"ID, FIO,DATAR, PASP_SER, PASP_NUM"
lcCom=lcCom+" values(0, ‘                    ’, ttoc(ctot('')),’    ’,
’      ’)"
con=sqlconnect('lcCom')
sqlexec(con,a)
```

Организация взаимодействия с удаленным источником данных

Как пользоваться функциями сквозных запросов SQL в Visual FoxPro?

1. Проверьте способность системы соединить компьютер с источником данных.
2. Воспользуйтесь утилитой Test ODBC в составе ODBC или ей подобной.
3. Установите соединение с источником данных с помощью функции

   `SQLCONNECT() или SQLSTRINGCONNECT().`

Например, если вы соединяете Visual FoxPro с источником данных master на сервере SQL Server, вам следует зарегистрироваться в качестве системного администратора (идентификатор пользователя **sa**) с паролем "", выдав следующую команду:

```
lhandle=SQLCONNECT('master','sa','')
```

4. Воспользуйтесь функциями сквозных запросов SQL в составе Visual FoxPro для извлечения данных в курсоры Visual FoxPro и обработки этих данных с помощью стандартных команд и функций Visual FoxPro. Например, можно выдать запрос в таблицу **Person** и просмотреть полученный курсор с помощью следующей команды:

```
? SQLEXEC (lhandle, "SELECT * FROM person" , "curs1")
```

5. Определите правильность выбранных данных в соответствии с условиями команды `SELECT`. Для просмотра можно использовать команду `BROWSE` .
6. Отсоединитесь от источника данных с помощью функции `SQLDISCONNECT()`.

```
=SQLDISCONNECT(lhandle)
```

Вызов хранимых процедур

Для вызова хранимой процедуры сервера можно использовать функцию `SQLExec()`.

Синтаксис, стандартный для любого сервера, выглядит следующим образом:

```
SQLExec(m.lnHandle,"{Call My_Proc (par1, par2)}")
```

Для передачи значения как параметра используется вопросительный знак:

```
Local lcPar1, lcPar2
lcPar1=100.01
lcPar2="символьное значение"
SQLExec(m.lnHandle,"{Call My_Proc (?m.lnPar1, ? m.lnPar2)}")
```

Конвертация в тип, понятный SQL-серверу, произойдет автоматически. Никаких дополнительных преобразований делать не надо.

В качестве параметра можно использовать не только переменные памяти, но и поля таблицы и объекты формы.

Однако следует иметь в виду, что переменные памяти с дробной частью для MS SQL-сервера будут сконвертированы в тип `REAL`, что может привести к потере точности. Для SQL-сервера существует специальный способ вызова:

```
SQLExec(m.lnHandle,"EXECUTE My_Proc par1, par2")
```

Здесь:

- `m.lnHandle` — номер установленного соединения с сервером;
- `My_Proc` — процедура, которую вы собираетесь вызвать;
- `Par1, par2` — параметры, передаваемые процедуре.

```
SQLExec(lnHandle,"{Call MyProc (?m.lnPar1, ?m.lnPar2)}").
```

Чтобы получить результат запроса из SQL в переменную, нужно создать на сервере процедуру:

```
CREATE PROCEDURE mysp_GetVersion @tcVersion Char(200) Output AS
  SELECT @tcVersion = @@VERSION
```

Для возвращения выходного параметра, т. е. параметра с признаком `OUTPUT`, используют следующую конструкцию:

```
lcVersion = SPACE(200)
lcCommand = "exec mysp_GetVersion ?@lcVersion"
=SQLEXEC(m.lnConn, m.lcCommand)
?lcVersion
```

Используются два символа — вопросительный знак и @. Не используйте в качестве параметра переменные памяти с именем, состоящим из одной буквы. В этом случае такая конструкция не сработает.

ActiveX Data Objects (ADO)

Разработка приложений существенно упрощается, если применить унифицированный механизм взаимодействия с самыми разными источниками данных. Такие технологии существуют, это, например, ODBC (Open Database Connectivity) и BDE (Borland Database Engine). На самом деле они являются частью более крупномасштабной технологии под названием Microsoft Data Access Components (MDAC). Термин MDAC является общим обозначением для всех разработанных компанией Microsoft технологий, связанных с БД. К этому набору относятся ADO, OLE DB, ODBC и RDS (Remote Data Services).

ADO — это технология программирования приложений для доступа к данным, разработанная компанией Microsoft и основанная на применении компонентов ActiveX. ADO позволяет представлять данные из различных источников, в том числе из реляционных баз данных (Access, Paradox и др.), текстовых файлов и т. д. в объектно-ориентированном виде. Объектная модель ADO состоит из следующих объектов высокого уровня и семейств объектов:

- `Connection` (представляет подключение к удаленному источнику данных);
- `RecordSet` (представляет набор строк, полученный от источника данных);
- `Command` (используется для выполнения команд и SQL-запросов с параметрами);
- `Record` (может представлять одну запись объекта `RecordSet` или же иерархическую структуру, состоящую из текстовых данных);
- `Stream` (используется для чтения и записи потоковых данных, например, документов XML или двоичных объектов);
- `Errors` (представляет ошибки);
- `Fields` (представляет столбцы таблицы базы данных);

- Parameters (представляет набор параметров SQL-инструкции);
- Properties (представляет набор свойств объекта).

Коллекция Properties и объект Property доступны через объекты Connection, RecordSet и Command. Коллекция Properties и объект Property содержат свойства, которые могут быть доступны только для чтения или для чтения-записи.

Объекты Connection, RecordSet и Command являются ключевыми объектами в объектной модели ADO. ADO-приложение может использовать объект Connection для установки соединения с сервером БД, объект Command — для выдачи команды к БД, таких как запросы, обновления и т. п., и объект RecordSet — для просмотра и манипулирования данными. Командный язык, используемый с объектом Command, зависит от провайдера для БД. В случае реляционных баз данных в качестве командного языка выступает SQL.

Соединение (Connection)

Объекты Connection, Command и RecordSet являются ключевыми элементами модели ADO. Объект Connection позволяет приложениям ADO установить связь с нужным источником данных. После этого объект Connection можно употреблять для непосредственного исполнения команд SQL, а также он может использоваться объектами Command и RecordSet. Для установления соединения с источником данных следует применить метод Open. Закрыть Connection поможет метод Close.

Execute — выполняет команду (типа T-SQL).

Команда (Command)

Объект Command используется для передачи команд в источник данных. Источники данных типа SQL Server состоят из динамических команд SQL, подготовленных команд SQL или вызовов хранимых процедур. Свойство CommandText содержит саму команду, а метод Execute ее запускает.

Параметр (Parameter)

Объекты Parameter применяются совместно с объектами Command. Объекты Parameter специфицируют индивидуальные атрибуты для каждого параметра, используемого объектом Command. Для создания объекта Parameter используется метод CreateParameter. Коллекция объектов Parameter, связанная с некоторым объектом Command, содержит все объекты такого типа, которые необходимы для заданного объекта Command.

Набор записей (RecordSet)

Разработчики значительно чаще пользуются объектами RecordSet, чем любыми другими объектами ADO. Первоначальной целью создания объекта RecordSet была отправка запросов SQL в SQL Server и последующий возврат результатов обработки запроса клиентскому приложению. Свойство Source (Источник) объекта RecordSet содержит сам запрос или команду SQL. Обычно для исполнения на сервере команды SQL применяют метод Open (Открыть) объекта RecordSet, а затем с помощью запроса наполняют набор данных в клиентском приложении.

Поле (Field)

Каждый объект `Field`, используемый вместе с объектом `RecordSet`, представляет определенный столбец, входящий в состав набора записей. Свойства объекта `Field` отражают тип данных столбца, его размер и значение. Коллекция объектов `Field`, автоматически формируемая при выполнении метода `Open ()`, содержит все объекты `Field` для каждого объекта `RecordSet`.

Свойство (Property)

Объект `Property` предоставляет информацию о различных характеристиках объектов `Connection`, `Command`, `RecordSet` и `Field`. Получить доступ к объектам `Property` можно через любой из этих объектов ADO.

Ошибка (Error)

Объект `Error` предоставляет пользователям информацию об ошибках, генерируемую средствами ADO в ходе исполнения программ. Несмотря на то, что методы объектов `Connection`, `Command` и `RecordSet` могут генерировать сообщения ADO о возникших ошибках, коллекция `Error` объекта `Connection` всегда содержит запись информации об ошибке. Свойство `Description` объекта `Error` содержит текстовое описание ошибки, а свойство `SQLState` возвращает код ошибки.

Каждый из перечисленных объектов включает в себя вложенные объекты и коллекции, в частности, значения всех полей текущей строки таблицы хранятся в коллекции `Fields` объекта `RecordSet`, а операциями над полями занимается объект `Field`.

Объект `RecordSet` имеет большое количество свойств и методов. Некоторые интересные свойства объекта `RecordSet` приведены в табл. 8.12.

Таблица 8.12. *Свойства объекта `RecordSet`*

Свойство	Описание
`AbsolutePage`	Указывает страницу, на которой находится текущая запись
`AbsolutePosition`	С помощью этого свойства вы можете прочитать или установить номер позиции текущей записи. Например, вы можете сохранить номер текущей записи, затем перейти в другую позицию, проделать какие-либо манипуляции и вновь вернуться к первоначальному положению в наборе данных
`ActiveCommand`	Показывает команду, которой был создан объект `RecordSet`
`ActiveConnection`	Указывает соединение, которому принадлежат объекты `Command`, `RecordSet`, или `Record`
`BOF, EOF`	Эти свойства информируют о достижении начала и конца набора данных соответственно. Свойство `BOF` получает значение `True` (Истина), если достигнуто начало набора данных (точнее, указатель перемещается за первую запись), а свойство `EOF` воспринимает значение `True` (Истина), когда достигается конец набора данных (указатель текущей записи перемещается за последнюю запись). В пустом наборе данных оба свойства одновременно имеют значение `True` (Истина). Основное предназначение свойств `BOF` и `EOF` — организация перемещения по записям набора данных

BookMark	Свойство позволяет поставить закладку на текущую запись, чтобы можно было к ней быстро вернуться. Перед использованием этого свойства следует проверить значение свойства Bookmarkable. Если это свойство принимает значение False (Ложь), то набор данных не позволяет пользоваться закладками

Таблица 8.12 *(окончание)*

Свойство	Описание
Cachesize	Указывает количество записей объекта RecordSet, которые кэшируются локально в памяти
CursorLocation	Указывает положение курсора
CursorType	Показывает тип курсора, используемый в объекте RecordSet
DataSource	Указывает объект, который содержит данные, которые нужно представлять как объект RecordSet
EditMode	Свойство указывает на статус редактирования записей набора данных. Возможны следующие значения: dbEditNone (записи в данный момент не редактируются), DbEditlnProgress (был использован метод Edit, и текущая запись была скопирована в буфер для редактирования), dbEditAdd (был использован метод AddNew, и в буфере находится новая запись)
Filter	Показывает фильтр для данных RecordSet
Index	Показывает имя активного индекса объекта RecordSet
MaxRecords	Возвращает максимальный номер записи, возвращаемый запросом к RecordSet
PageCount	Показывает, сколько страниц содержит объект RecordSet
PageSize	Показывает, сколько записей содержит одна страница RecordSet
RecordCount	Свойство возвращает количество записей в наборе данных. Если вы создали объект RecordSet динамического типа (dbOpenDynaset), то свойство RecordCount может вернуть меньше записей, чем набор данных реально содержит. Чтобы гарантированно получить точное число записей, следует перейти сначала в конец набора данных, а потом вернуться к началу
Source	Показывает источник данных для RecordSet
State	Указывает состояние объекта для всех объектов — открыт или закрыт
Status	Показывает статус объекта Field

Методы объекта RecordSet позволяют манипулировать данными, а также осуществлять поиск, навигацию по набору данных и большое количество других полезных действий. Некоторые из этих методов описаны ниже (табл. 8.13).

Таблица 8.13. *Методы объекта RecordSet*

Метод	Описание
AddNew	Добавляет новую запись в набор данных. Каждому методу AddNew должен соответствовать метод Update
Cancel	Прекращает незаконченный вызов
CancelBatch	Прекращает выполнение пакетного изменения

Таблица 8.13 *(окончание)*

Метод	Описание
CancelUpdate	Отменяет любые изменения, которые были сделаны в текущей или новой колонке объекта RecordSet или коллекцию полей объекта Record, перед вызовом метода Update
Clone	Создает дубликат объекта RecordSet из существующего RecordSet
Close	Освобождает переменную типа RecordSet и удаляет соответствующий объект из коллекции. Не забывайте применять этот метод, когда набор данных вам уже не нужен
Delete	Удаляет текущую запись из набора данных
Move	Передает позицию текущей записи в объект RecordSet
MoveFirst, MoveLast, MoveNext, MovePrevious	Методы этой группы осуществляют переход соответственно к первой, последней, следующей и предыдущей записи набора записей (объекта RecordSet). Методы группы Move позволяют также организовать *тотальный* поиск по всему набору записей или по некоторой его части. Если вы применяете метод MovePrevious в то время как активной является первая запись набора, то свойство BOF приобретает значение True (Истина), а текущая запись становится неопределенной (т. е. в наборе отсутствует текущая запись). Повторное применение этого метода вызовет ошибку, а значением свойства BOF останется True (Истина). То же замечание можно сделать относительно метода MoveNext и свойства EOF для случая, когда текущей является последняя запись набора
NextRecordSet	Очищает текущий объект RecordSet и возвращает следующий RecordSet
Open	Открывает соединение с источником данных
Requery	Изменяет данные объекта RecordSet, выполняя запрос данных, на которых базируется объект
Save	Сохраняет RecordSet в файле или объекте Stream
Seek	Производит поиск по индексу RecordSet
Update	Сохраняет изменения, произведенные в текущем RecordSet
UpdateBatch	Записывает пакет изменений на диск

Но давайте перейдем от теории к практике.

Пример 1

Разместим на форме таблицу с данными из базы Access, применяя ADO. Для подключения базы данных в технологии ADO используется строка подключения (connection string). Вы можете набрать строку подключения вручную, однако в общем случае для создания строки подключения рекомендуется использовать построитель DataEnvironment, рабочее окно которого показано на рис. 8.16.

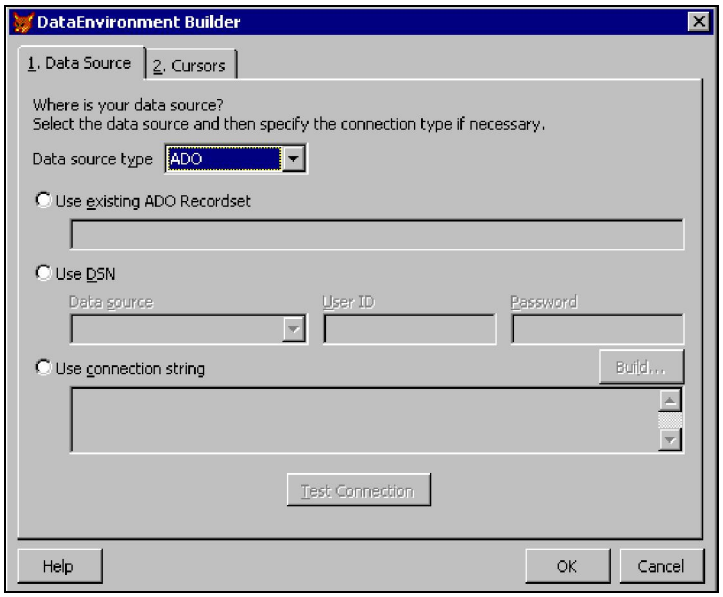

Рис. 8.16. Построитель DataEnvironment

Здесь:

Data source type — определяет тип источника данных. Выбираем — ADO.

Use existing ADO Recordset — определяет ссылку на действительный объект ADO `RecordSet`.

Use DSN — определяет имя источника данных (DSN) из списка **Data source** и значения в окнах **User ID** и **Password**.

Use connection string — определяет действительную строку соединения. Это представляет собой тот же самый тип строки соединения, который вы вводите для свойства ADO `ConnectionString`.

Вместо этого, вы можете щелкнуть по кнопке **Build…** для отображения диалогового окна **Data Link Properties**. Тогда вы сможете выбрать подходящий OLE DB Provider и прочие необходимые параметры строки соединения (рис. 8.17, табл. 8.14).

Таблица 8.14. *OLE DB Providers*

Провайдер	Описание
Microsoft Jet 4.0 OLE DB Provider	Базы данных MS Access и другие БД
Microsoft OLE DB Provider for DTS Packages	DTS Packages — это "пакеты" DTS, которые представляют собой некие программы, как правило, предназначенные для выполнения неких преобразований данных, так или иначе связанных с собственно MS SQL — сервером. Либо закачка данных из внешних источников, либо наоборот — перекачка данных во внешние источники. Либо просто запуск хранимой процедуры MS SQL-сервера

Таблица 8.14 *(окончание)*

Провайдер	Описание
Microsoft OLE DB Provider for Indexing Services	Внутренний механизм Windows, ускоряющий поиск информации в файлах при помощи построения каталога с файловой информацией. Служба индексирования Indexing Service интегрирована в IIS и часто используется для индексирования Web-узлов
Microsoft OLE DB Provider for Internet Publishing	Позволяет разработчикам манипулировать каталогами и файлами с использованием http
Microsoft OLE DB Provider for ODBC Drivers	Драйверы ODBC (по умолчанию)
Microsoft OLE DB Provider for Olap Services 8.0	Online Analytical Processing
Microsoft OLE DB Provider for ORACLE	Базы данных Oracle
Microsoft OLE DB Provider for OutLook Search	Базы данных Outlook
Microsoft OLE DB Provider for SQL Server	Базы данных MS SQL Server
Microsoft OLE DB Provider for Visual FoxPro	Базы данных Visual FoxPro
Microsoft OLE DB Simple Provider	Для создания ваших собственных провайдеров для простых текстовых данных
SQL Server Replication OLE DB Provider for DTS	Data Transformation Services — одна из служб MS SQL сервера
MSDataShape	Используется для создания структурированного объекта RecordSet, например: "Provider=MSDataShape;Data Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:\MyData\SomeDB.mdb"

Существует также огромное количество других провайдеров OLE DB для MDAC. Провайдеры OLE DB можно получить как от Microsoft, так и от независимых производителей. Список провайдеров OLE DB очень большой и постоянно меняется, поэтому его невозможно воспроизвести в данной книге.

Выберите провайдер Jet 4.0 OLE DB, для этого сделайте двойной щелчок на строке с надписью Jet 4.0 OLE DB Provider, на экране появится вкладка **Connection**. Внешний вид этой страницы для разных провайдеров может быть различным. Для провайдера Jet редактор предложит вам ввести имя базы данных и аутентификационные данные.

Протестируйте полученное соединение, используя кнопку **Test Connection** (рис. 8.18).

На вкладке **Advanced** (Дополнительно) вы можете контролировать режим доступа к базе данных. Здесь вы можете настроить монопольный доступ или доступ только для чтения. На вкладке **All** (Все) перечисляются все параметры строки подключения.

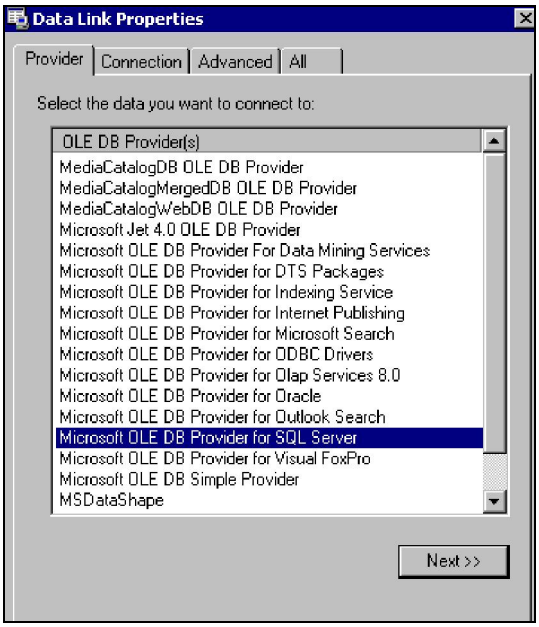

Рис. 8.17. Редактор соединений Data Link Properties

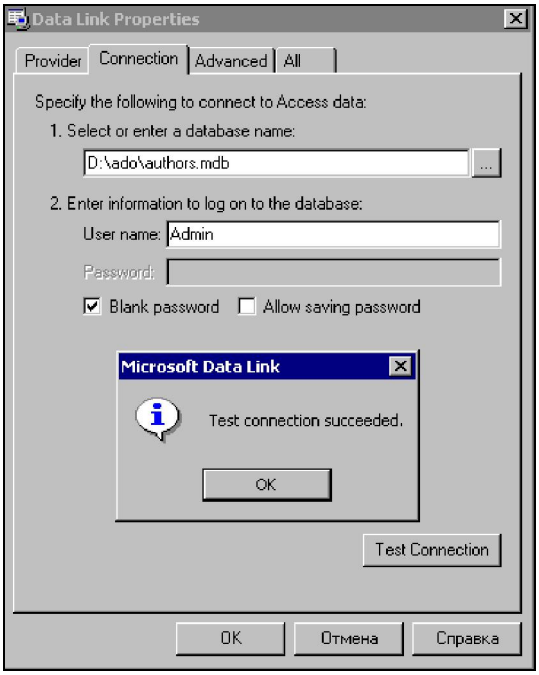

Рис. 8.18. Строка соединения

Следующий шаг — настройка `Cursor Adapter`. Об этом вы можете прочесть ниже в этой же главе.

Теперь нажмите левую кнопку мыши на заголовке созданного курсора и тащите его на форму, при этом на форме будет создан `Grid`. Запустив полученную форму, мы увидим данные из таблицы Authors, доступ к данным получен через ADO (рис. 8.19).

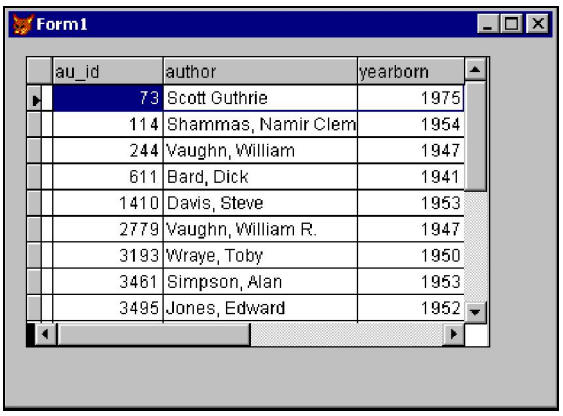

Рис. 8.19. Данные из таблицы Authors

Если в созданной форме открыть окно **Data Environment**, то в событии `CursorAdapter.Init` мы увидим не что иное, как программный код создания ADO-соединения (листинг 8.2).

```
local loConnDataSource
loConnDataSource = createobject('ADODB.Connection')
***<DataSource>
loConnDataSource.ConnectionString = [Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=D:\ado\authors.mdb;Persist Se] + ;
[curity Info=False;]
***</DataSource>
loConnDataSource.Open()
This.DataSource = createobject('ADODB.RecordSet')
This.DataSource.CursorLocation   = 3  && adUseClient
This.DataSource.LockType         = 3  && adLockOptimistic
This.DataSource.ActiveConnection = loConnDataSource
```

Однако у описанного подхода по-прежнему имеется один существенный недостаток: если вы идентифицируете базу данных при помощи некоторого имени файла, путь к этой базе будет жестко закодирован внутри исполняемого файла приложения. В результате возможности приложения будут существенно ограничены. Стоит только перенести базу данных в другое место — и после запуска формы мы увидим сообщение об ошибке (рис. 8.20).

Рис. 8.20. Ошибка отсутствия базы данных в указанном каталоге

Поэтому чаще всего применяется программный способ.

```
loConnDataSource.ConnectionString = [Provider=Microsoft.Jet.OLEDB.4.0;
Data Source=];
         +SYS(5)+CURDIR()+[authors.mdb;Persist Security Info=False;]
```

Кроме этого способа применяются еще файлы связи с данными, которые представляют собой INI-файлы.

```
[oledb]
Provider=Microsoft.Jet.OLEDB.4.0
Data Source=Data Source=D:\ado\authors.mdb
```

Информацию о том, как работать с INI-файлами, можно получить из *главы 28*.

Пример 2

```
Local oConn As ADODB.Connection
Local oSet As ADODB.RecordSet
oConn=CreateObject("ADODB.Connection")
oConn.ConnectionString = "DSN=fox; UID=sa; PWD="
oConn.Open
rs = CreateObject ("ADODB.Recordset")
rs.ActiveConnection = oConn
rs.Source = "SELECT * FROM Customers ORDER BY CompanyName"
rs.Open
Do While Not rs.EOF
   wait wind rs("Name")
   rs.MoveNext
enddo
```

В первой строке мы определяем объект `oConn` типа `ADODB.Connection`, или просто `Connection`, с помощью оператора `CreateObject`. (Стоит отметить, что все ADO-компоненты имеют идентификатор вида `ADODB.Object`.) Это самый важный ADO-компонент, отвечающий за связь между VFP-приложением и базой данных. У него очень много методов и свойств, и одно из них, `ConnectionString`, используется во второй строчке примера. Оно определяет собственно строку инициализации ODBC-соединения. После создания `Connection`-объекта и определения строки инициализации в третьей строке примера с помощью метода `Open` открывается доступ к базе данных.

Теперь выберем все записи из таблицы Users и упорядочим их по полю **Name**. Для этого нужно над открытой базой данных выполнить SQL-запрос `SELECT * FROM Users ORDER BY Name`. Для хранения результата запроса создадим объект `rs` типа `RecordSet`, привязываем его к `Connection`-объекту `oConn`, используя свойство `ActiveConnection`, задаем строку запроса в свойстве `Source`, после чего исполняем запрос с помощью метода `Open`.

Теперь в объекте `rs` содержится набор записей — результат запроса. Кроме того, для сканирования этих строк объект содержит указатель, первоначально установленный на самую первую запись. Теперь, используя метод `MoveNext`, можно последовательно перемещаться по всем записям, пока не дойдем до самой последней записи. С помощью свойства `EOF` определяется самая последняя запись объекта.

Пример 3

В примере открывается таблица Customers базы данных Northwind на SQL-сервере.

В файл записываются имена ее полей и данные первой записи.

```
oConn = CreateObject ("ADODB.Connection")
oConn.ConnectionString = "DSN=fox; UID=sa; PWD="
oConn.Open
IF FILE('cust_id.htm')
```

```
        ERASE "cust_id.htm"
endif
rs = CreateObject ("ADODB.Recordset")
rs.ActiveConnection = oConn
rs.Source = "SELECT * FROM Customers ORDER BY CustomerID"
rs.Open

oSet=oConn.Execute("SELECT * FROM Customers ORDER BY CustomerID")
For i=1 to oSet.Fields.Count
        STRTOFILE(oSet.Fields(i-1).name+CHR(13),"cust_id.htm", 1)
        STRTOFILE(TRANSFORM(oSet.fields(i-1).value+CHR(13)), "cust_id.htm",1)
NEXT
  rs.MoveNext
rs.Close
oConn.Close
```

Другие примеры — на компакт-диске.

Применение *CursorAdapter*

Базовый класс `CursorAdapter` (сокращенно CA) обеспечивает работу с широким диапазоном источников данных:

- Локальные;
- Open Database Connectivity (ODBC);
- ActiveX Data Object (ADO);
- Extensible Markup Language (XML).

`CursorAdapter` — это первый базовый класс в VFP, который обеспечивает преобразование между природными курсорами VFP и разными источниками данных и обладает способностью транслировать потоки данных ODBC, ADO recordsets и документов XML в курсоры Visual FoxPro.

Для создания адаптера курсора можно воспользоваться построителем адаптера курсора, выполнив следующие действия:

1. Откройте окно среды данных (Data Environment). Нажмите правую кнопку мыши и выберите в контекстном меню пункт **Builder**. На экране появится окно **DataEnvironment Builder**. Можно провести эксперимент с базой данных Northwind, которая существует "в двух лицах": как родная база данных VFP в C:\Program Files\Microsoft Visual FoxPro 9\Samples\Northwind, и как экспериментальная база данных SQL Server. Для родной базы данных в **DataSourceType** нужно выбрать из списка **Native**, для серверной — **ODBC** (рис. 8.21, 8.22).

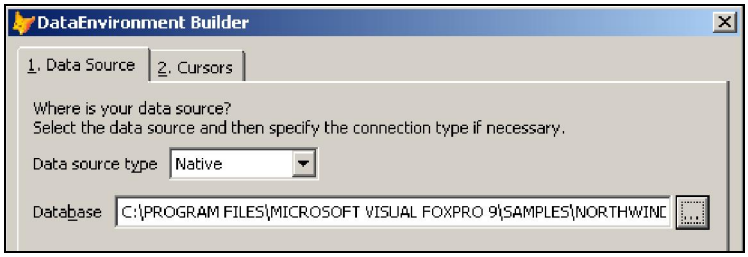

Рис. 8.21. Построитель DataEnvironment для родной базы данных Northwind

2. На второй вкладке мы должны определить `CursorAdapter`. Поскольку у нас пока его нет, нажимаем кнопку **New…** (рис. 8.23).

Появляется новое окно, имеющее три вкладки (рис. 8.24).

- **Properties** — позволяет задать имя и псевдоним адаптера курсора;
- **Data Access** — позволяет сформировать команду **Select** для выбора данных из источника (рис. 8.25);
- **Auto-Update** — позволяет задать параметры автоматического обновления содержимого адаптера курсора (рис. 8.26).

Присваиваем свое собственное имя создаваемому курсору и устанавливаем флажок **Use DataEnvironment data source**.

На второй вкладке необходимо определить перечень полей, включаемых в курсор. Для получения списка полей (фактически это обычный оператор `SELECT`) можно использовать кнопку **Build**.

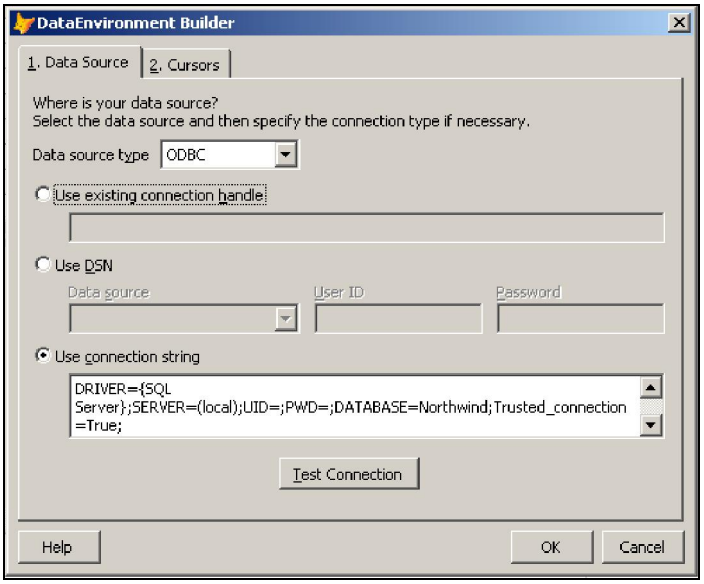

Рис. 8.22. Окно построителя DataEnvironment для базы данных Northwind на сервере

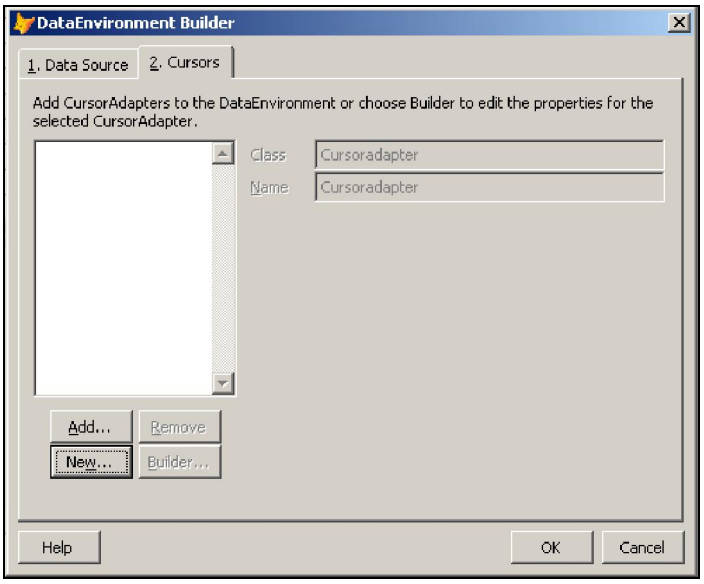

Рис. 8.23. Вкладка **Cursors** построителя DataEnvironment

Далее переходим к вкладке **Auto-Update**. Установим флажки в полях **Auto-update**, **Update all fields** и установим флажок в поле **Primary key** для поля **CUSTOMERID**. Остальное оставим по умолчанию. Примерно это выглядит так, как показано на рис. 8.26.

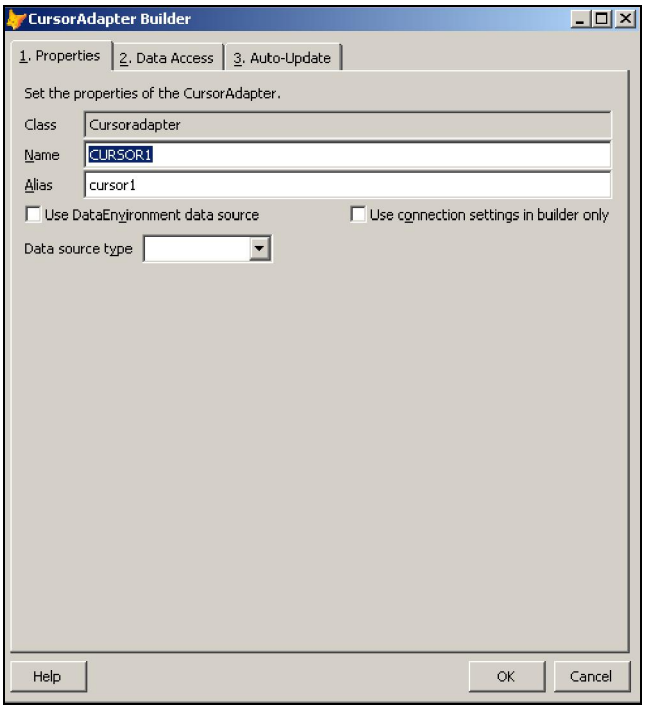

Рис. 8.24. Окно DataEnvironment. Построитель CursorAdapter

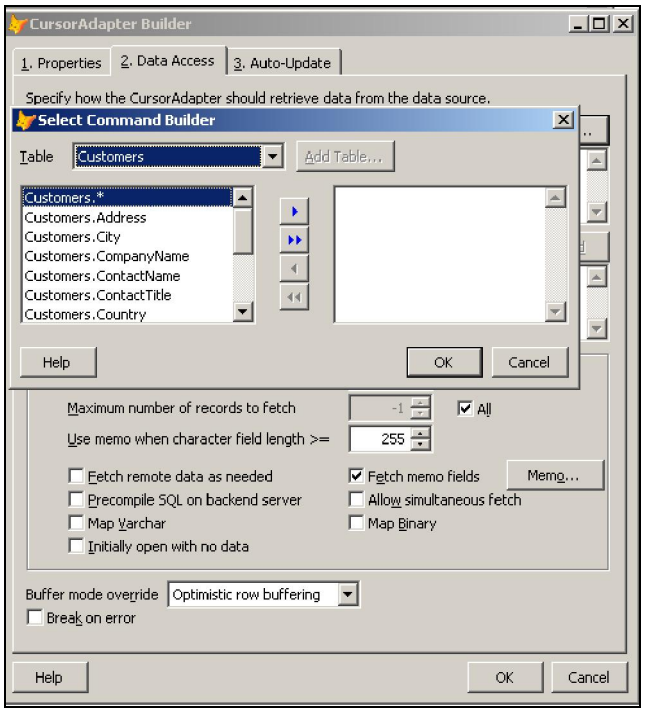

Рис. 8.25. Вкладка **Data Access** построителя CursorAdapter

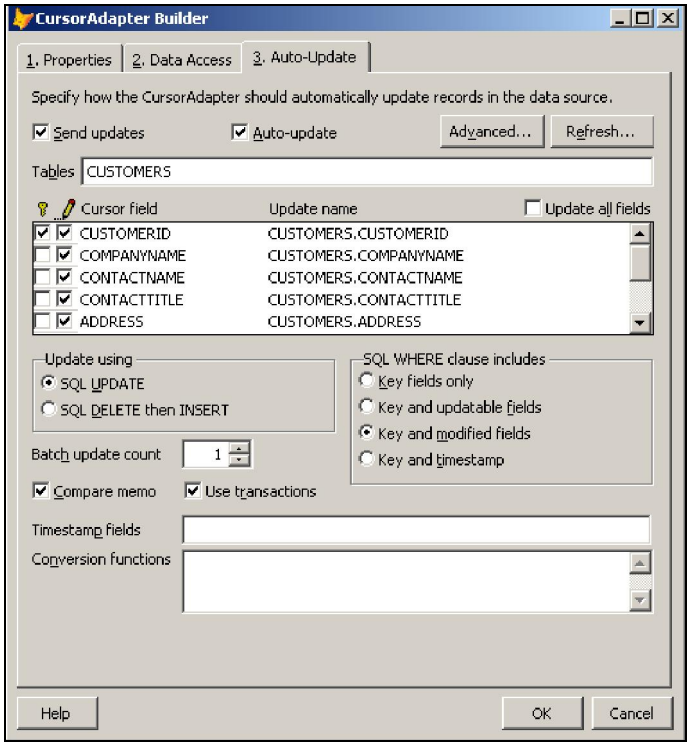

Рис. 8.26. Вкладка **Auto-Update** построителя CursorAdapter

И в первом, и во втором случае мы получим буферизованный курсор. Независимо от местоположения источника данных мы получим один и тот же результат (рис. 8.27).

Создав адаптер курсора, вы сможете работать с данными независимо от того, какой именно источник данных вами используется, т. е. взаимодействовать с внешними

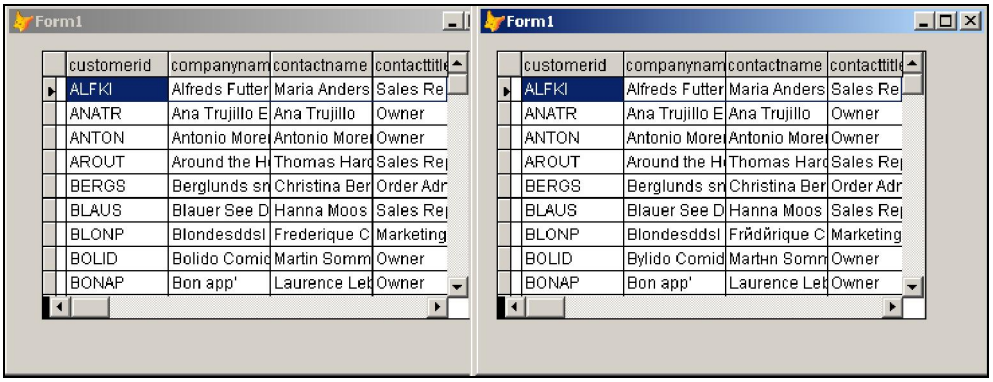

Рис. 8.27. Слева — результат для родной таблицы VFP,
справа — для таблицы, расположенной на SQL-сервере

данными исключительно только через методы и свойства CA-класса. Значения свойств будут зависеть от применяемого источника данных. Адаптер курсора позволяет преобразовывать данные внешнего источника во временные VFP-курсоры или измененные данные из VFP-курсоров передавать в соответствующие внешние источники.

Данные в VFP-курсорах, получаемых средствами CA, представляют из себя "локальную копию" внешних данных, и всегда являются буферизованными. А VFP-функция `TABLEUPDATE()`, сохраняющая изменения клиента из VFP-курсоров на свои внешние источники, поддерживает работу с CA.

Некоторые различия все-таки существуют, особенно если использовать метод `CursorFill` при установленном в `.F.` свойстве `DE AutoOpenTables` в случае, когда `CursorAdapter` использует источник данных, установленный для **DataEnvironment**. Но это уже тонкости.

Свойства и методы *CursorAdapter*

- Свойство `DataSourceType`. `CA`-класс поддерживает работу с внешними источниками данных: ODBC, ADO, XML, VFP-Native. Какой именно источник данных обслуживается экземпляром класса, определяется значением свойства `DataSourceType`.
- Метод `CursorFill()` — выполняет команду (из свойства `SelectCmd`), чтобы заполнить данными курсор (свойство `Alias`). Заполнение курсора может выполняться с использованием метода `AutoOpen()`, если класс `CA` помещен в экземпляр класса `DataEnvironment (DE)`. При выполнении метода `CursorFill()` также вызываются соответствующие события: `BeforeCursorFill()`, `AfterCursorFill()`.
- Метод `AutoOpen()` — вызывает метод `CursorFill()`, по умолчанию без параметров. Если `CA`-класс принадлежит экземпляру класса `DE`, то метод `DE.OpenTables()` вызывает метод `AutoOpen()` для каждого экземпляра `CA`. Такой механизм позволяет (при `DE.AutoOpenTables=.T.`) как открыть данные формы до наступления события `Load()` формы, так и более точно настроить параметры вызова метода `CursorFill()` в случае необходимости, например при параметризованных запросах для ADO-источника данных.
- Метод `CursorRefresh()` — обновляет данные в курсоре (свойство `Alias`) данными с сервера, повторно выполняя команду (из свойства `SelectCmd`) в соответствии типом источника данных (свойства `DataSourceType`). При выполнении этого метода также вызываются соответствующие события: `BeforeCursorRefresh()`, `AfterCursorRefresh()`.
- Метод `RecordRefresh()` — обновляет данные указанной(ых) записи(ей) в курсоре (свойство `Alias`) данными с сервера. Этот метод интегрирован с функцией `REFRESH()`. Команда обновления записи может быть уточнена с помощью свойства `RefreshCmd`. При выполнении метода также вызываются соответствующие события: `BeforeRecordRefresh()`, `AfterRecordRefresh()`.

- Методы CursorAttach(),CursorDetach() — позволяют подсоединиться/отсоединиться к/от заданному курсору, чтобы автоматически заполнить свойства экземпляра CA-класса, взяв их из указанного курсора.
- Свойство CursorSchema — позволяет (пере)определить структуры данных источника. В VFP 8.0 используется для XML-данных, в VFP 9.0 распространяется на все типы данных. В VFP 9.0 значение свойства CA.UseCursorSchema определяет значение по умолчанию для передачи как параметра lUseSchema в метод CursorFill(), если этот параметр не использован при обращении к этому методу.
- Свойство NoData — в VFP 9.0 определяет значение по умолчанию, передаваемое в качестве NoData параметра методу CursorFill() в том случае, если этот параметр не использован при обращении к этому методу.

CA-класс тесно интегрирован с функциями: CURSORSETPROP(), CURSORGETPROP(), TABLEUPDATE().

А теперь следует разобраться, как это работает.

Для начала пропишем строку соединения:

```
"DRIVER=(SQL Server);"+;
   "SERVER=(local);"+;
   "PASSWORD=;"+;
   "DATABASE=Northwind"

lnConnHandle=SQLSTRINGCONNECT(lcDSN)
```

и получим заголовок соединения:

```
lnConnHandle=SQLSTRINGCONNECT(lcDSN)
```

Создадим объект — экземпляр класса CursorAdapter, используя стандартную функцию CREATEOBJECT().

```
lo_CA=CREATEOBJECT("CursorAdapter")
```

Прежде всего, установим псевдоним для курсора, который создаст Visual FoxPro в объекте CursorAdapter.

```
lo_CA.Alias="cTestMySQL"
```

Поскольку мы используем тип источника данных ODBC, то и присвоим соответствующему свойству DataSourceType объекта указанный тип:

```
lo_CA.DataSourceType="ODBC"
```

Далее, необходимо передать полученное соединение объекту CursorAdapter. Это делается с помощью свойства DataSource, которому в данном случае мы присваиваем имя переменной памяти, в котором хранится заголовок соединения.

```
lo_CA.DataSource=lnConnHandle
```

Теперь самое важное — что мы желаем получить из нашей базы. Пусть это будут все поля таблицы Customers. Для выборки используется простая команда:

```
"Select * FROM Customers"
```

Эту команду мы должны поместить в свойство `SelectCmd` нашего объекта на базе класса `CursorAdapter`.

```
lo_CA.SelectCmd="Select * FROM Customers"
```

Все подготовлено, осталось только как-то передать данные из таблицы в курсор, который нам создаст объект `CursorAdapter`. Для этого используется метод `CursorFill` этого объекта.

```
lo_CA.CursorFill
```

Теперь соберем все вместе, откроем новый программный файл, скопируем полученный листинг и запустим (листинг 8.3).

```
lnConnHandle = sqlstringconnect('driver=SQL Server;server=(local);' + ;
        'database=Northwind;uid=sa;pwd=;trusted_connection=yes')

IF lnConnHandle < 0
   = MESSAGEBOX('Cannot make connection', 16, 'SQL Connect Error')
ELSE
   = MESSAGEBOX('Connection made', 48, 'SQL Connect Message')
ENDIF
lo_CA=CREATEOBJECT("CursorAdapter")
lo_CA.Alias="cTestSQL"
lo_CA.DataSourceType="ODBC"
lo_CA.DataSource=lnConnHandle
lo_CA.SelectCmd="Select * FROM Customers"
lo_CA.CursorFill
BROWSE
RELEASE lo_CA
SQLDISCONNECT(lnConnHandle)
```

И что же мы имеем в итоге? Такой вот результат (рис. 8.28).

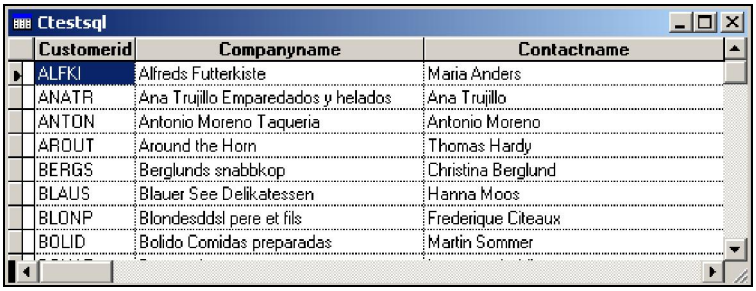

Рис. 8.28. Результат работы созданной программы

То есть для просмотра таблиц хватает. Однако необходимо и добавлять, и удалять, и обновлять. Для того чтобы мы могли обновить данные в исходной таблице, нам нужно указать значения для ряда свойств объекта `CursorAdapter`, а именно:

- `Tables`;
- `KeyFieldList`;
- `UpDatableFieldList`;
- `UpdateNameList`.

Вот здесь есть один нюанс. Если посмотреть на описание `UpdateCmdDataSource`, то там недвусмысленно сказано, что это может быть символьный тип даных, и в принципе, если используется машинный источник данных, то мы можем присвоить этому свойству строку с командой SQL — `sqlstringconnect`([dsn=MySQL2CA;]), где и укажем этот DSN. Предлагаемый построитель VFP для объекта `CursorAdapter` так и делает.

Далее сказано, что это может быть пустая строка или `.Null.` и в этом случае `CursorAdapter` будет использовать источник данных из свойства `DataSource`. Вот она — тонкость. Если мы сделаем это, и не оставим значения этого свойства по умолчанию, а возьмем и явно присвоим пустую строку или `.Null.`,

```
lo_CA.UpdateCmdDataSource=""/.Null.
```

то при попытке обновления данных мы получим сообщение об ошибке (`Connection handle is invalid`).

Для того чтобы все это заработало, есть два варианта:

- мы вообще не упоминаем в коде ни `UpdateCmdDataSource`, ни `UpdateCmdDataSourceType`;
- используем в коде оба свойства и присваиваем в обязательном порядке свойству `UpdateCmdDataSourceType` значение `ODBC` и свойству `UpdateCmdDataSource` уже созданный заголовок соединения.

Пусть это будет так:

```
lo_CA.UpdateCmdDataSourceType="ODBC"
lo_CA.UpdateCmdDataSource=lnConnHandle
```

В итоге, для используемой нами базы данных и таблицы фрагмент кода, отвечающий за обновление данных, может выглядеть так:

```
lo_CA.Tables="Customers"
lo_CA.KeyFieldList="CustomerID"
lo_CA.UpdateNameList="CustomerID  Customers.Customerid, CompanyName
Customers.CompanyName"
lo_CA.UpdatableFieldList="Customerid,CompanyName"
lo_CA.UpdateCmdDataSourceType="ODBC"
lo_CA.UpdateCmdDataSource=lnConnHandle
```

или так:

```
lo_CA.Tables="Customers"
lo_CA.KeyFieldList="CustomerID"
lo_CA.UpdateNameList="CustomerID  Customers.Customerid,CompanyName
Customers.CompanyName"
lo_CA.UpdatableFieldList="Customerid,CompanyName"
```

Поскольку при работе с `CursorAdapter` используется буферизация, то необходимо позаботиться об установке `SET MULTILOCKS` и установить ее в `ON`.

Новый листинг программы в предпочтительном варианте будет выглядеть так (листинг 8.4).

```
SET MULTILOCKS ON
        lnConnHandle = sqlstringconnect('driver=SQL Server;server=(local);' + ;
                'database=Northwind;uid=sa;pwd=;trusted_connection=yes')

        IF lnConnHandle < 0
           = MESSAGEBOX('Cannot make connection', 16, 'SQL Connect Error')
        ELSE
           = MESSAGEBOX('Connection made', 48, 'SQL Connect Message')
        ENDIF
lo_CA=CREATEOBJECT("CursorAdapter")
lo_CA.Alias="cTestSQL"
lo_CA.DataSourceType="ODBC"
lo_CA.DataSource=lnConnHandle
lo_CA.SelectCmd="Select * FROM Customers"
lo_CA.Tables="Customers"
lo_CA.KeyFieldList="CustomerID"
lo_CA.UpdateNameList="CustomerID  Customers.Customerid,CompanyName
Customers.CompanyName"
lo_CA.UpdatableFieldList="Customerid,CompanyName"
llCursorFilled=lo_CA.CursorFill()
? "Is cursor filled? "
?? llCursorFilled
```

```
BROWSE
RELEASE lo_CA
SQLDISCONNECT(lnConnHandle)
```

А что же с удалением/вставкой? Измените явно тип обновления:

```
lo_CA.UpdateType=2
```

Таким образом, для операций над данными необходимо прописать только 4 свойства:

- `Tables`;
- `KeyFieldList`;
- `UpdatableFieldList`;
- `UpdateNameList`.

Работа с полем, имеющим свойство *IDENTITY*

`IDENTITY` — это не тип данных. Это дополнительное свойство, ограничение, накладываемое на целочисленные типы данных в MS SQL-сервере. Свойство может быть применено к полям следующего типа: `tinyint`, `smallint`, `int`, `bigint`, `decimal(p,0)` или `numeric(p,0)`.

Аналогом этого свойства в Visual FoxPro является тип данных `Integer(AutoIncrement)`. Только не надо считать, что `Integer-AutoIncrement` это и есть поле со свойством `Identity`, это именно аналог. В основном они похожи, но имеют ряд отличий.

Поля со свойством `IDENTITY` обладают следующими особенностями:

- в одной таблице допустимо существование только одного поля со свойством `IDENTITY`;
- поле со свойством `IDENTITY` нельзя редактировать. Они имеет свойство "только для чтения";
- значение полю со свойством `IDENTITY` присваивается автоматически в момент создания новой записи.

Есть еще некоторые особенности, но они уже являются следствием перечисленных особенностей.

Новое значение — это последнее использованное значение плюс некоторая фиксированная величина. Обратите внимание, новое значение опирается не на максимальное значение в существующих записях, а на последнее использованное значение. Это значит, что записи с последним использованным значением физически может не существовать, тем не менее это значение будет использовано.

Другими словами, в последовательности значений поля со свойством `IDENTITY` вполне допустимы "дыры". Список значений вовсе не непрерывный.

Как правило, в качестве шага приращения указывают 1, но это может быть и любое целое число, в том числе и отрицательное.

В связи с особенностями полей со свойством `IDENTITY` такие поля часто используют в качестве первичных ключей. Другими словами, в качестве полей, по значению которых всегда можно однозначно идентифицировать запись таблицы.

Следует иметь в виду, что свойство `IDENTITY` никак не контролирует уникальность данных. Например, если изначально поле имело тип `INTEGER`, и в него был введен ряд значений. А затем была изменена структура таблицы, и на данное поле было наложено свойство `IDENTITY`, то новые записи вполне могут иметь те же данные, что уже были введены ранее в эту таблицу. Поэтому если поле со свойством `IDENTITY` используется в качестве первичного ключа, то на это поле следует наложить дополнительное ограничение по уникальности.

Однако, несмотря на явные достоинства использования полей со свойством `IDENTITY` в качестве первичного ключа, они имеют и серьезный недостаток: значение полей со свойством `IDENTITY` невозможно узнать до того, как запись будет физически создана.

Проблема заключается в том, что для того, чтобы узнать значение поля какой-либо записи, эту запись надо сначала найти. А поиск записи как раз и осуществляется по значению первичного ключа. Того самого, значение которого необходимо определить. Замкнутый круг: чтобы прочитать значение, это значение надо знать!

Структура хранения данных в MS SQL-сервере принципиально отличается от структуры хранения данных в DBF-файлах. В нем нет таких понятий, как "физический номер записи", "следующая запись", "последняя запись" и т. п. То есть невозможно перейти к "последней записи", чтобы прочитать значение ее первичного ключа.

Более того, хотя новое значение поля со свойством `IDENTITY` всегда больше любого из существующих значений (если шаг приращения — положительное число), но определить это новое значение, просто вычислив максимальное из существующих значений, тоже нельзя. Нет, само максимальное значение, разумеется, будет получено. Просто нет никакой гарантии, что полученное значение — это значение именно той записи, которая и была создана.

Дело тут в том, что, как правило, MS SQL-сервер используется в многопользовательских приложениях. Это значит, что сразу несколько пользователей, одновременно, могут создавать новые записи. Получается, что один пользователь создал новую запись, затем начал вычислять максимальное значение, и в этот момент другой пользователь также создал новую запись. В результате, первый пользователь в качестве максимального значения получит значение записи, созданной вторым пользователем.

Так что же, отказаться от использования полей со свойством `IDENTITY` в качестве первичного ключа? Вовсе нет. Все-таки существуют способы определения значения поля со свойством `IDENTITY` у новой записи.

Собственно, есть три принципиальные стратегии определения значения поля со свойством `IDENTITY` в новой, только что созданной, записи:

- значение, возвращаемое системной переменной `@@IDENTITY`;
- значение, возвращаемое функцией `SCOPE_IDENTITY()`;
- нахождение новой записи по значению других полей.

Теперь рассмотрим более подробно достоинства и недостатки каждой стратегии.

Значение, возвращаемое системной переменной *@@IDENTITY*

В MS SQL-сервере есть ряд системных переменных, значение которых изменяется автоматически при наступлении определенных событий. В частности, значение системной переменной `@@IDENTITY` автоматически устанавливается равным значению поля со свойством `IDENTITY` последней созданной записи в текущем соединении. То есть создание новых записей в другом соединении (другим пользователем) никак не повлияет на ее значение в данном соединении.

Ну, так вот оно, решение. Просто после создания новой записи читаем значение системной переменной `@@IDENTITY` и имеем искомое значение.

В целом, верно. Проблема только в том, что системная переменная `@@IDENTITY` меняет свое значение при создании записи в любой таблице.

На практике это означает, что если на таблицу установлен триггер на вставку, в теле которого дается команда `INSERT` на создание записи в другой таблице, которая, в свою очередь, также имеет поле со свойством `IDENTITY`, то системная переменная `@@IDENTITY` получит значение поля из этой второй таблицы.

Другими словами, опираться на значение системной переменной `@@IDENTITY` можно, но помня о том, что эта переменная не привязана к значению поля одной таблицы.

Значение, возвращаемое функцией *SCOPE_IDENTITY()*

В версии MS SQL 2000 была введена системная функция `SCOPE_IDENTITY()`. Эта функция также возвращает значение поля со свойством `IDENTITY` последней созданной записи, но созданной в пределах текущего `SCOPE`.

Адекватно перевести термин `SCOPE` на русский язык достаточно сложно. Но, приблизительно, можно сказать так: `SCOPE` — это одна процедура или функция. Другими словами, `SCOPE_IDENTITY()` вернет значение поля со свойством `IDENTITY` последней записи, созданной в пределах той процедуры, где эта функция была вызвана.

Триггер — это уже другой `SCOPE` (другая функция), поэтому он никак не повлияет на значение, возвращаемое `SCOPE_IDENTITY()`.

Даже если два пользователя одновременно вызвали одну и ту же процедуру, то каждый вызвал процедуру в своем `SCOPE`, т. е. опять-таки нет конфликта.

К недостаткам этой функции относится то, что ее следует вызывать в пределах того `SCOPE`, где и была создана новая запись интересующей нас таблицы. А это не всегда возможно.

Другими словами, для корректного использования `SCOPE_IDENTITY()` необходимо всегда следить за областью действия `SCOPE`, зачастую создавая специальные процедуры.

Нахождение новой записи по значению других полей

Если помните, то основная проблема с определением значения поля со свойством `IDENTITY` заключается в том, что данное поле используется в качестве первичного ключа, т. е. по его значению как раз и находят нужную запись.

Однако, зачастую, таблицы имеют поле или набор полей, по которым также можно однозначно определить запись. Например, если речь идет о справочнике, то, разумеется, справочник имеет поле "Название". Также очевидно, что это поле должно быть уникально в пределах справочника. Иначе просто теряется смысл использования самого справочника. Зачем вводить в справочник записи с одинаковым значением?

Почему же не использовать в качестве первичного ключа это самое "Название"? Зачем вообще нужно поле со свойством `IDENTITY`? Это тема отдельного разговора. Вкратце, "Название" — это для пользователя (внешние данные), а `IDENTITY` — это для обеспечения ссылочной целостности базы данных (внутренние данные).

Значение поля со свойством `IDENTITY` в новой записи неизвестно. Но значение поля "Название" в этой новой записи вполне известно. Пользователь сам же его и ввел! Значит, после создания новой записи можно найти эту новую запись по значению поля "Название" и прочитать значение поля со свойством `IDENTITY`.

Проблема только в том, что далеко не всегда существует такое поле или набор полей для однозначной идентификации записи. Это, кстати, одна из причин ввода так называемых суррогатных ключей, тех самых полей со свойством `IDENTITY`.

Если, тем не менее, вы решите использовать эту стратегию для поиска новой записи, то обязательно наложите на поле "Название" (или выбранный вами набор полей) ограничение по уникальности. То есть, чтобы случайно не оказалось двух записей с одинаковым значением в этом поле.

Как работать с полями со свойством *IDENTITY* в FoxPro

С теоретической частью закончили, теперь "попробуем со всем этим добром взлететь". То есть определимся, как же воспользоваться всеми этими знаниями в FoxPro. Еще раз уточним задачу, которую необходимо решить.

Добавляется запись в таблицу MS SQL-сервера, имеющую поле со свойством `IDENTITY`. Необходимо сразу после создания новой записи получить значение поля со свойством `IDENTITY` на стороне FoxPro.

У FoxPro есть три принципиальные возможности организации работы с MS SQL-сервером:

- прямое использование технологии Pass-Through через функцию `SQLEXEC()`;
- использование `Remote View`;

- использование `Cursor Adapter`.

Тут следует остановиться на том, какое именно событие собственно создает запись на MS SQL-сервер. С Pass-Trough все ясно, это собственно прямая команда серверу создать новую запись. А вот с `Remote View` и `Cursor Adapter` несколько иначе.

Результатом работы как `Remote View`, так и `Cursor Adapter` является курсор. То есть некая временная таблица, физически расположенная на машине клиента. По умолчанию этот курсор автоматически открывается в режиме оптимистической буферизации строк (3) и может быть переключен только в режим оптимистической буферизации таблиц (5). Переключиться в режим пессимистической буферизации или отключить буферизацию совсем для этого курсора невозможно.

Следовательно, новая запись сначала физически будет создана именно на клиентской машине в этом самом курсоре. Точнее, в буфере этого курсора. Физическое создание записи на MS SQL-сервере произойдет только после сброса буфера.

Для строковой буферизации сброс буфера может произойти автоматически при выполнении одного из следующих действий:

- переход (или попытка перехода) на другую запись;
- закрытие курсора;
- переключение в режим табличной буферизации;
- по команде `TableUpdate()`.

Для табличной буферизации сброс буфера может произойти только по команде `TableUpdate()`, и никак иначе.

Никакие другие действия и операции ни с `Remote View`, ни с `Cursor Adapter` не приведут к созданию новой записи на MS SQL-сервере. Если при выполнении какой-либо операции оказалось, что на MS SQL-сервере создалась новая запись, это означает, что курсор находился в режиме строковой буферизации, и произошло одно из событий, вызвавших автоматический сброс буфера.

Например, такое может произойти по команде `Requery()` для `Remote View`. Но это вовсе не означает, что команда `Requery()` сбрасывает буфер. Просто одним из условий выполнения команды `Requery()` является закрытие ранее существовавшего курсора. А вот это событие как раз и вызовет автоматический сброс буфера, если курсор находится в режиме строковой буферизации.

Чтобы избежать подобных недоразумений, переключайте курсор в режим табличной буферизации (5). В этом случае вы всегда сможете контролировать процесс сброса буфера.

Однако следует понимать, что даже если вы установите режим табличной буферизации, измените несколько записей в `Remote View` или `Cursor Adapter`, а потом дадите команду `TableUpdate()`, все равно сброс буфера будет происходить по одной записи за раз. То есть на сервер будет послана не одна команда, например, на модификацию, а набор команд по модификации каждой записи в отдельности.

Применительно к операциям создания новой записи из этого следует вывод, что во всех событиях объекта `Cursor Adapter` ***всегда*** происходит вставка только одной записи за раз.

Прямое использование технологии Pass-Through через функцию *SQLEXEC()*

При таком способе работы программист как бы напрямую работает с MS SQL-сервером. Сам формирует все команды, отсылаемые на сервер, получает результат, и сам же его обрабатывает. В этом случае не составляет никакого труда послать дополнительный запрос серверу на значение функции `SCOPE_IDENTITY`.

```
LOCAL lcNewValue, lnResut
lcNewValue = "Новое значение"
lnResut = SQLExec(m.lnConnectHandle,"INSERT INTO MyTab (Field1) VALUES
(?m.lcNewValue)")
IF m.lnResut>0
        SQLExec(m.lnConnectHandle,"SELECT NewIdent=SCOPE_IDENTITY()","NewIdent")
        ?NewIdent.NewIdent
ELSE
        LOCAL laError(1)
        =AERROR(laError)
        * Анализ массива laError для уточнения причины ошибки
ENDIF
```

В данном примере `m.lnConnectHandle` — это число, номер соединения с MS SQL-сервером, которое настраивается раньше. `MyTab` — это таблица, которая имеет поле со свойством `IDENTITY`.

После выполнения второго запроса в результирующем курсоре `NewIdent` в поле `NewIdent` первой записи и получим искомое значение. В данном синтаксисе и команда вставки, и вызов функции `SCOPE_IDENTITY()` происходят в одном `SCOPE`. Поэтому получаем нужное значение.

Использование *Remote View*

`Remote View` — это некая "надстройка" над технологией Pass-Through. По сути, при создании новой записи выполняется та же команда `INSERT INTO`. Однако проблема в том, что даже если прочитать номер соединения, в котором работает `Remote View`, а потом выполнить запрос для определения значения, возвращаемого `SCOPE_IDENTITY()`, то получим `NULL`, поскольку в этом случае команда вставки и `SCOPE_IDENTITY()` выполняются в разных `SCOPE`. Следовательно, остается только два способа определения значения поля со свойством `IDENTITY`.

```
* Определение значения системной переменной @@IDENTITY
LOCAL lnConnectHandle
lnConnectHandle = CursorGetProp("ConnectHandle","MyRemoteView")
```

```
SQLExec(m.lnConnectHandle,"SELECT NewIdent=@@IDENTITY","NewIdent")
?NewIdent.NewIdent

* Определение по значению другого поля
LOCAL lnConnectHandle, lcNickName
lnConnectHandle = CursorGetProp("ConnectHandle","MyRemoteView")
lcNickName = MyRemoteView.NickName
SQLExec(m.lnConnectHandle,"SELECT TabId FROM MyTab WHERE
NickName=?lcNickName","NewIdent")
?NewIdent.TabId
```

В обоих примерах `MyRemoteView` — это имя вашего `Remote View`. `NickName` — это имя поля в `Remote View`, по значению которого выполняется поиск новой записи, а `TabID` — это то самое поле со свойством `IDENTITY`, значение которого и надо определить.

Предполагается, что создание новой записи уже произошло. Либо явно была дана команда `TableUpdate()`, либо `Remote View` находится в режиме строковой буферизации, и была попытка перехода на другую запись.

А почему во втором случае не было использовано, казалось бы, более очевидное решение поиска после `Requery()`? Что-то вроде

```
=REQUERY("MyRemoteView")
SELECT MyRemoteView
LOCATE FOR NickName = "Новое значение"
?MyRemoteView.TabID
```

На это есть несколько причин.

Во-первых, `Requery()` предполагает выполнение запроса заново, т. е. предполагается, что произошел сброс буфера всех записей `Remote View`. Но ведь это может быть не так. Например, сбрасывают буфер по одной записи за раз в цикле.

Во-вторых, обычно `Remote View` содержит в себе дополнительные условия отбора записей, и только что созданная запись может вообще не попасть в условия отбора. То есть после `Requery()` запись хотя и будет физически создана на сервере, но в самом `Remote View` отображена не будет.

Использование *Cursor Adapter*

Объект `Cursor Adapter` был введен в FoxPro, начиная с версии Visual FoxPro 8. Он призван облегчить работу с удаленными данными при выполнении некоторых стандартных операций, в частности, операции создания новой записи. Через `Cursor Adapter` можно реализовать все три варианта получения значения поля со свойством `IDENTITY` в новой записи.

Значение, возвращаемое системной переменной *@@IDENTITY*

После того как будет выполнен сброс буфера и произойдет физическая создание новой записи на MS SQL-сервере, в объекте `Cursor Adapter` сработает событие

AfterInsert. Вот в нем как раз и надо выполнить дополнительный запрос на сервер и прочитать значение системной переменной @@IDENTITY.

```
PROCEDURE AfterInsert
LPARAMETERS cFldState, lForce, cInsertCmd, lResult
IF lResult=.T. && вставка произошла успешно. Надо получить значение ID
        LOCAL currentArea
        currentArea=SELECT()

        TRY
                IF 1=SQLEXEC(this.DataSource,"SELECT
NewIdent=@@IDENTITY","NewIdent")
                        REPLACE TabId WITH NewIdent.NewIdent IN (This.Alias)
                        USE IN IDRes
                ELSE
                        LOCAL laError(1)
                        =AERROR(laError)
                        * Анализ массива laError для уточнения причины ошибки
                ENDIF
        FINALLY
                SELECT (currentArea)
        ENDTRY
ENDIF
ENDPROC
```

После успешного создания новой записи на сервере при помощи SQLExec() для того же самого соединения мы определили значение @@IDENTITY и записали это значение в текущую запись курсора в ключевое поле.

В версии Visual FoxPro 9 в объект Cursor Adapter было добавлено свойство InsertCmdRefreshCmd. Это команда, которая посылается на сервер только после успешной вставки новой записи, т. е. здесь нет необходимости убеждаться в самом факте создания новой записи.

В паре с другим новым свойством InsertCmdRefreshFieldList можно выполнить обновление, опираясь на значение системной переменной @@IDENTITY проще. Для этого всего лишь надо сделать следующие настройки объекта CursorAdapter.

```
CursorAdapter.InsertCmdRefreshCmd       = "SELECT @@IDENTITY"
CursorAdapter.InsertCmdRefreshFieldList = "TabId"
```

Здесь TabId — это не имя поля со свойством IDENTITY в таблице собственно на MS SQL-сервере, а имя поля курсора, полученного на стороне клиента, в котором и отображается содержимое поля со свойством IDENTITY. Как правило, эти имена совпадают, но, в общем случае, могут и отличаться. Более подробно о свойстве InsertCmdRefreshFieldList читайте ниже, в разделе, посвященном нахождению новой записи по значению других полей.

Значение, возвращаемое функцией *SCOPE_IDENTITY()*

Здесь все несколько сложнее. Дело в том, что действовать так же, как и в случае определения системной переменной @@IDENTITY, нельзя. SQLExec() и Cursor Adapter работают в разных SCOPE. Поэтому при подобном подходе SCOPE_IDENTITY() всегда будет возвращать NULL. Чтобы преодолеть это противоречие, используют специальные процедуры.

Для начала надо создать на самом MS SQL-сервере вот такую хранимую процедуру:

```
CREATE PROCEDURE Get_ValueInt
                @ValueIn Int,
                @ValueOut Int OUTPUT
AS
SET NOCOUNT ON
SELECT @ValueOut=@ValueIn
```

Как видите, цель данной процедуры просто присвоить значение входного параметра выходному параметру. Теперь можно использовать эту процедуру для определения значения SCOPE_IDENTITY() в Cursor Adapter. Для этого в событии BeforeInsert делается такая подмена:

```
PROCEDURE BeforeInsert
LPARAMETERS cFldState, lForce, cInsertCmd
cInsertCmd = cInsertCmd + ;
                "; DECLARE @id int" + ;
                "; SELECT @id=SCOPE_IDENTITY()" + ;
                "; EXEC Get_ValueInt @id, ?@MyTab.TabId"
ENDPROC
```

Только следует иметь в виду, что в данной подмене MyTab — это уже не имя таблицы на MS SQL-сервере, а имя курсора, создаваемого у клиента. Точнее, то имя, которое записано в свойстве Alias Cursor Adapter. Соответственно, TabId — это имя поля курсора, созданного у клиента и содержащего значение поля со свойством IDENTITY.

В данном случае, по сути, формируется динамическая хранимая процедура. То есть не просто одна команда INSERT, а последовательность команд. Команды отделяются друг от друга символом точки с запятой, хотя это вовсе не обязательно. Достаточно отделять команды друг от друга простым пробелом.

Нахождение новой записи по значению других полей

Начиная с версии Visual FoxPro 9, в объекте Cursor Adapter появились дополнительные свойства, позволяющие автоматизировать эту процедуру без написания дополнительного кода.

InsertCmdRefreshFieldList — список полей, значение которых будет обновлено после Insert.

`InsertCmdRefreshKeyFieldList` — поле или набор неключевых полей, также однозначно идентифицирующих запись, по которым и будет осуществлен поиск новой записи.

Следует отметить, что в этих свойствах указываются не имена полей таблицы собственно MS SQL-сервера, а соответствующие им имена полей курсора, полученного на стороне клиента. Применительно к тому же примеру это будет примерно так:

```
InsertCmdRefreshFieldList       = "TabID"
InsertCmdRefreshKeyFieldList    = "NickName"
```

Другими словами, опираясь на значение поля `NickName`, будет найдена запись в таблице на MS SQL-сервере и прочитано значение поля `TabID`, после чего это значение будет записано в новую запись на стороне клиента.

Особенности работы с полями типа *Blob*

В SQL-сервере есть несколько типов полей, которые допускают хранение в них больших массивов информации (**BLOB** поля — **binary large object**). Это поля типа `text`, `ntext` и `image`. Максимальный размер информации, который можно сохранить в полях таких типов, составляет 2^{31}– 1 (2,147,483,647) байт для полей типа `text` и `image` и 2^{30} — 1 (1,073,741,823) символов для полей типа `ntext`. Как правило, данные полей этих типов хранятся в отдельных страницах базы данных по структуре, схожей с той, которая используется для хранения индексов, а в строке таблицы хранится только указатель размером в 16 байт.

Хранить большие массивы информации в поле `text` опасно, это может привести к потере информации, поскольку текстовые поля вовсе не предназначены для хранения больших объемов данных. Стало быть, надо применять тип `Image`. Но поля типа `Image` имеют особенность — с сервера данные выбираются вовсе не в `Blob`, а в поле типа `General`. Как же быть? Решение есть. Если вам необходимо выбрать данные именно в `Blob`, нужно перед `SQLEXEC` установить `CURSORSETPROP(MapBinary, .T., 0)`, в этом случае `SQLEXEC` будет использовать `Blob` поля для `Image`.

Поддержка XML

Общие сведения о языке XML

Язык XML (eXtensible Markup Language) принадлежит к так называемой группе языков разметки. К языкам разметки, в частности, относятся такие широко известные языки, как SGML, HTML, XSLT. В последнее время появляется большое количество разновидностей языков, базирующихся на основе XML. Например, такие как SOAP, WML (Wireless Markup Language), SVG (Scalebel Vector Grafic), XHTML, RDF (Resorce Description Framwork), MathML (Mathematical Markup Language), UML.

Данные в XML-документе хранятся в виде текста, обрамленного специальными символами, управляющими разметкой документа. XML — это набор специальных инструкций, предназначенных для формирования в документах древовидной структуры и определения отношений между различными элементами этой структуры. Данные,

обладающие древовидной структурой, оформляются в XML посредством ряда базовых конструкций языка:

- элементы (теги) и их атрибуты;
- текст (между открывающим и закрывающим тегами);
- секции символьных данных (CDATA);
- неразделяемые сущности (entities);
- комментарии (<!-- ... -->).

Отличие XML от похожего синтаксически на него языка разметки HTML в том, что в HTML существует жестко ограниченный набор тегов, направленных в основном на то, чтобы определить способ отображения в Internet Browser. В XML можно использовать любую разметку, которая может потребоваться для описания данных, причем при этом там нет никаких указаний на способы отображения — это только данные плюс их структура.

Теги, или, как их иногда называют, управляющие дескрипторы, в документах выделяются относительно основного содержимого документа и служат в качестве инструкций для программы, производящей показ содержимого документа на стороне клиента. Для обозначения инструкций используются символы < и >, внутри которых помещаются их названия и параметры. Понятие того, как элемент может включать в себя другие элементы, требует уточнения: элементы должны быть "правильно" вложенными. То есть начальный тег, будучи помещенный в некоторый родительский, должен иметь в этом же теге и концевой. Другими словами, конструкции с "общими пересечениями", типа:

```
<A><B></A></B>
```

недопустимы.

Но теги только зададут структурирование текста или выполнение команд обработки. Для более подробной информации об отображении элемента необходимы атрибуты. Что это такое? Атрибуты — это простые символьные конструкции, которые добавляются к элементам для большей информативности, т. е. параметры, уточняющие характеристики тега.

Например,

```
<person Name="Customer" type="C"
/person>
```

Сущность (entities) можно определить как часть документа с разметкой, не связанную с каким-либо структурным понятием. Например, сущностью может быть файл, строка текста, иллюстрации, специальные символы.

Комментариями является любая область данных, заключенная между последовательностями символов <!-- и -->. Комментарии пропускаются анализатором и поэтому при разборе структуры документа в качестве значащей информации не рассматриваются.

Все XML-документы могут быть разделены на две группы:

- правильно оформленные документы (well-formed);
- состоятельные (допустимые) или (valid).

Все документы, соответствующие спецификациям XML, называются правильно оформленными, если они могут быть использованы без декларации типа документа (DTD) или XML-схем (XML Schema). Другими словами, такие документы не могут использовать внешние декларации, а атрибуты не подвергаются специальной обработке и не имеют значений по умолчанию. Таким образом, чтобы XML-документ был правильно/хорошо оформленным, должны быть выполненными следующие условия:

- синтаксис должен соответствовать спецификациям XML;
- элементы образуют иерархическое дерево с одним простым корневым узлом;
- при отсутствии внешних DTD также не существует ссылок на сущности и определений атрибутов (сущности-параметры могут быть использованы только в блоках внутренних DTD).

Документ называется состоятельным, если он правильно оформлен и имеет внешние определения в виде деклараций типа документа (DTD), т. е. не может быть корректно обработан без привлечения "внешней информации", или у него могут быть предопределены сущности и/или атрибуты с использованием внутренних деклараций типа документа.

Любой XML-документ должен всегда начинаться с инструкции <?xml?>, внутри которой можно задавать номер версии языка, номер кодовой страницы и другие параметры, необходимые программе-анализатору в процессе разбора документа. Первая строка `<?xml version="1.0" standalone="yes"?>` является частью пролога документа и представляет XML-декларацию (XML Declaration), состоящую из инструкции по обработке документа (processing instruction) — `<?xml ... ?>`. Каждый XML-документ начинается именно с XML-декларации, включающей в себя обязательный атрибут `version`, указывающий на используемую версию языка.

Второй атрибут `encoding` представляет собой способ кодировки данных (`encoding=UTF-8`). Третий атрибут `standalone` указывает на то, что весь документ содержится именно в этом файле и не требует импорта других файлов (например, `standalone="yes"`). Все, что располагается ниже пролога, представляет собой содержание документа.

Простейший XML-документ может выглядеть так, как это показано в примере:

```
<?xml version="1.0" encoding="Windows-1251"  standalone="yes"?>
<databases>
<tables>Table1</tables>
</databases>
```

В прологе указано:

- версия языка;
- данные документа находятся в кодировке, содержащие русские символы для OS Windows;

- документ следует обрабатывать без всяких ссылок на внешние определения, причем если такие внешние определения появятся в ходе обработки, то должно возникнуть сообщение об ошибке.

Список распространенных значений атрибута `encoding` для России может быть таким:

- Windows-1251;
- KOI8-R;
- Cp866;
- ISO-8859-5.

Другая информация, помещаемая в пролог, — ссылка на таблицу стилей, с помощью которой будет обрабатываться содержание документа:

```
<?xml-stylesheet type= "text/[css|xsl]" href="..." [charset="..."] [title="..."]
[media="..."]?>
```

Например:

```
<?xml-stylesheet type="text/xsl" href="my.xsl" charset="Windows-1251"?>
```

Таким образом, любой XML-документ имеет следующую структуру:

```
<?xml version="1.0" encoding="Windows-1251"?>
<?xml-stylesheet type= "text/xsl" href="my.xsl"?>
<databases>
<tables>Table1</tables>
</databases>
```

Вы видите, что XML-документ представляет собой древовидную структуру, т. е. состоит из узлов. XML требует обязательного наличия открывающего и закрывающего тегов, обрамляющих содержание документа, которые образуют главный корневой узел. Этот узел может быть только в единственном числе. В данном файле это тег `<databases> ... </databases>`, являющийся корневым узлом документа. Внутри корневого узла располагаются дочерние узлы, которые в свою очередь могут иметь дочерние. Внутри корневого узла нашего простейшего документа размещен узел `<tables>...</tables>`. В VFP главный корневой узел имеет название VFPData и является обязательным. Можно изменить его название с помощью

```
XMLAdapter.XMLName=STRCONV("MainTag",5)
```

Однако полностью отменить его нельзя. Корневой тег — это необходимая часть XML-документа.

С точки зрения Visual FoxPro XML-документ должен представлять собой либо одиночную таблицу, либо несколько таблиц, связанных отношением "один-ко-многим". Не забывайте, что курсор — это тоже таблица, хотя и временная, поэтому в качестве таблицы может выступать и курсор, и `CursorAdapter`.

Попробуем сформировать XML-документ с помощью VFP. Для формирования XML-документа (Cursor — XML) и обратного преобразования (XML — Cursor) служат функции `CURSORTOXML` и `XMLTOCURSOR`., которые имеют следующий синтаксис:

```
CURSORTOXML(nWorkArea | cTableAlias, cOutput [, nOutputFormat
[, nFlags [, nRecords [, cSchemaName [, cSchemaLocation [, cNameSpace ]]]]]])
```

Здесь:

`nWorkArea` — это рабочая область, из которой создается XML;

`cTableAlias` — псевдоним таблицы, из которой создается XML;

`cOutput` — путь и имя файла или переменная памяти, в которую посылается XML-документ;

`nOutputFormat` — определяет формат вывода: 1 — Elements, 2 — Attributes, 3 — RAW (подробнее о форматах вывода см. ниже);

`nFlags` — определяет форматирование XML-документа;

`nRecords` — указывает число записей, подлежащих выводу в XML, по умолчанию

`nRecords`=0 — в этом случае выводятся все записи;

`cSchemaName` — указывает имя и местонахождение схемы информации о данных в `cOutput`, например `MySchema.xsd`;

`cSchemaLocation` — определяет местонахождение, в котором приложение, читающее данные из XML, должно искать файл схемы. Параметр используется только в случае, когда месторасположение схемы отличается от места расположения файла .XML;

`cNameSpace` — указывает пространство имен XML или производимой схемы и по умолчанию устанавливается в пустую строку ("");

```
XMLTOCURSOR(eExpression | cXMLFile [, cCursorName [, nFlags ]])
```

`eExpression` — определяет выражение для XML-данных. Это может быть переменная памяти, memo-поле, HTTP-запрос, возвращение результата SOAP-методом, XML из XMLDOM, ADO-поток данных;

`cXMLFile` — определяет имя и путь XML-документа;

`cCursorName` — определяет имя курсора для хранения результата;

`nFlags` — определяет форматирование XML-документа.

При формировании документа мы можем определить структуру документа на основе трех типов:

- `Element centric` (XML на базе элементов). В этом случае каждое поле представляется дочерним элементом родительского элемента;
- `Attribute centric` (XML на базе атрибутов). Каждое поле представляется атрибутом элемента VFPData;

- `RAW` (Общий, XML на базе атрибутов). Каждая строка курсора представляется идентификатором "row" и столбец — атрибутом row-элемента, при этом имя атрибута совпадает с именем столбца.

Для вывода данных нам потребуется таблица или курсор, пусть это будет таблица Person (рис. 8.29).

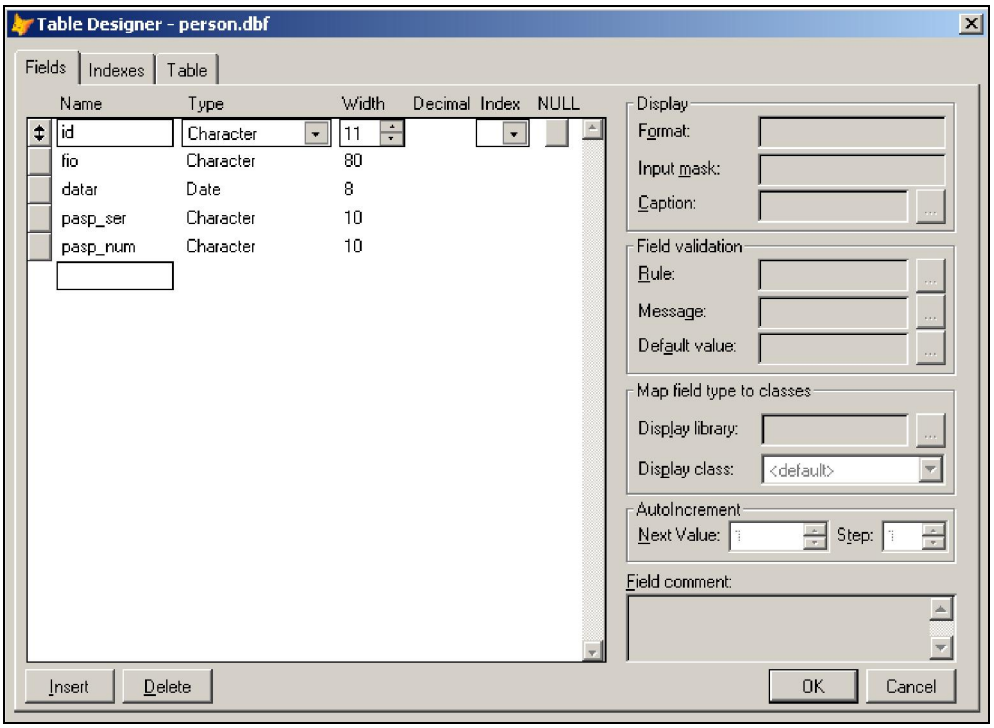

Рис. 8.29. Структура таблицы, из которой экспортируются данные

Используем функцию `CURSORTOXML` для экспорта данных в XML. Испробуем все три варианта.

1. Для `Element centric`:

```
USE person
GO top
CURSORTOXML('person','d:\char8\primer2.xml', 1, 16+512, 2, '')
```

Полученный результат (рис. 8.30).

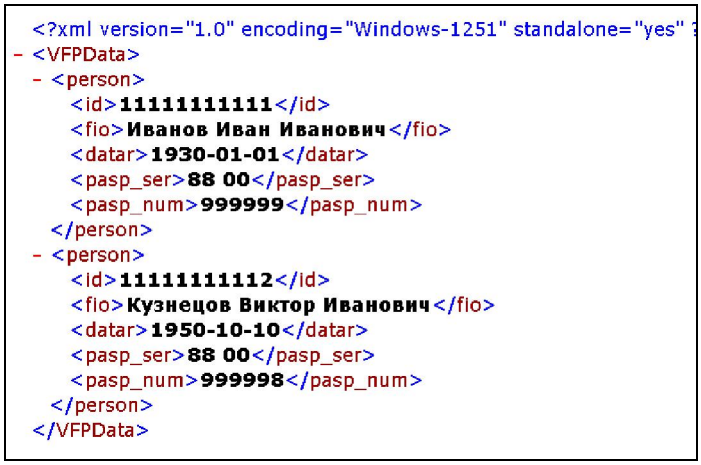

Рис. 8.30. Результат выполнения CursorToXML для Element Centric

2. Для `Attribute centric`:

```
USE person
GO top
CURSORTOXML('person','d:\char8\primer2.xml', 2, 16+512, 2, '')
```

Результат — на рис. 8.31.

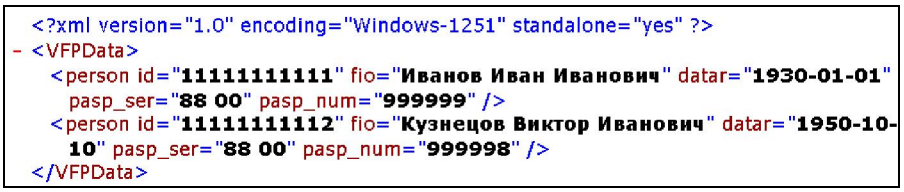

Рис. 8.31. Результат выполнения CursorToXML для Attribute Centric

3. Для `RAW`:

```
USE person
GO top
CURSORTOXML('person','i:\kniga\cd-disk\char8\primer2.xml', 3, 16+512, 2, '')
```

Результат выполнения — на рис. 8.32.

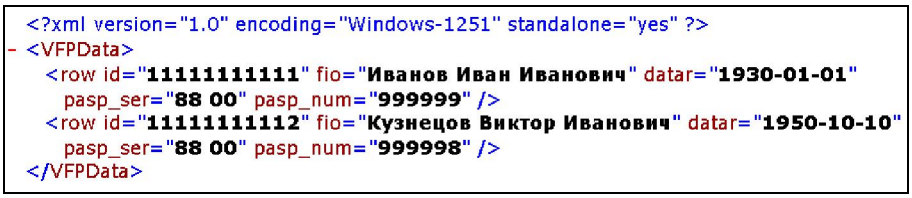

Рис. 8.32. Результат выполнения CursorToXML для RAW

Как видно из примеров, ни один из типов структур документа не содержит типов экспортируемых данных. Чтобы получить тип данных, необходимо использовать схему (Schema).

Схемы

Чтобы XML-данные все-таки отображать, часто используют ассоциированные "таблицы стилей" (css|xsl), для чего используют инструкции по обработке типа:

```
<?xml-stylesheet type= "text/[css|xsl]" href="..." [charset="..."] [title="..."]
[media="..."]?>
```

Для логического описания структуры в XML-документах (т. е. правил, которым должна удовлетворять структура XML-данных) используют язык DTD (Data Type Definition) или XSD-схемы.

Схема — это объявление структуры документа. При импорте данных используется либо внешняя схема, либо внутренняя. Если схема не указана, то VFP выполняет два прохода по XML-документу: первый — для определения схемы, второй — для выполнения преобразований. VFP не может импортировать данные, если их невозможно преобразовать в таблицу.

VFP поддерживает XML Schema Definition (XSD), который определяет элементы, а также их атрибуты и типы данных. Чтобы определить файл схемы, можно задать команду

```
=CURSORTOXML('person','d:\char8\primer2.xml', 1, 16+512, 2,
0,'d:\char8\primer2.xsd')
```

Мы экспортируем таблицу Person в файл d:\char8\primer2..xml с типом `Element centric`, т. е. каждое поле будет выведено в файл как самостоятельный узел, при этом вывод кодируется в кодовой странице курсора (16) и результат посылается в файл, указанный в параметре `Coutput` (512); в файл XML будут выведены *все записи* и схема будет содержаться в файле `d:\primer2.xsd`.

Полученная в результате этих действий схема выглядит так:

```
 <?xml version="1.0" encoding="Windows-1251" standalone="yes" ?>
- <xsd:schema id="VFPData" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:msdata="urn:schemas-microsoft-com:xml-msdata">
- <xsd:element name="VFPData" msdata:IsDataSet="true">
- <xsd:complexType>
- <xsd:choice maxOccurs="unbounded">
- <xsd:element name="person" minOccurs="0" maxOccurs="unbounded">
- <xsd:complexType>
- <xsd:sequence>
- <xsd:element name="id">
- <xsd:simpleType>
- <xsd:restriction base="xsd:string">
  <xsd:maxLength value="11" />
```

```
  </xsd:restriction>
  </xsd:simpleType>
  </xsd:element>
- <xsd:element name="fio">
- <xsd:simpleType>
- <xsd:restriction base="xsd:string">
  <xsd:maxLength value="80" />
  </xsd:restriction>
  </xsd:simpleType>
  </xsd:element>
  <xsd:element name="datar" type="xsd:date" />
- <xsd:element name="pasp_ser">
- <xsd:simpleType>
- <xsd:restriction base="xsd:string">
  <xsd:maxLength value="10" />
  </xsd:restriction>
  </xsd:simpleType>
  </xsd:element>
- <xsd:element name="pasp_num">
- <xsd:simpleType>
- <xsd:restriction base="xsd:string">
  <xsd:maxLength value="10" />
  </xsd:restriction>
  </xsd:simpleType>
  </xsd:element>
  </xsd:sequence>
  </xsd:complexType>
  </xsd:element>
  </xsd:choice>
  <xsd:anyAttribute namespace="http://www.w3.org/XML/1998/namespace"
processContents="lax" />
  </xsd:complexType>
  </xsd:element>
  </xsd:schema>
```

XMLAdapter Class

Этот класс в полном объеме поддерживает импорт и экспорт курсоров VFP и XML-данных. Вы можете импортировать XML и содержащуюся в нем схему или сгенерировать XML из таблиц (табл. 8.15 и 8.16).

Объект `XMLAdapter` хранит объектную ссылку на схему XML (XML Schema) и узлы объектной модели документа (DOM). То есть он не хранит информацию о реальной схеме или содержании. Объект `XMLAdapter` содержит коллекцию `Tables`, которая, в свою очередь, содержит один или более объектов `XMLTable` и описывает XML, как курсор Visual FoxPro вместе с любой связанной информацией. Каждый объект

XMLTable может также содержать дочерний XMLTable, и, самое большее, коллекцию Fields с одним или более объектами XMLField.

Определить класс можно так же, как и любой другой класс:

```
LOCAL loXA as XMLAdapter
loXA = NEWOBJECT('XMLAdapter')
```

ЗАМЕЧАНИЕ

Для работы XMLAdapter требуется установка Microsoft XML Core Services (MSXML) 4.0 Service Pack 1 (SP1) или более поздней версии.

Таблица 8.15. *Свойства класса XMLAdapter*

Свойство	Описание
CodePage	Устанавливает кодовую страницу для объектов импорта или экспорта. Например: oXMLAdapter.CodePage = nValue oXMLTable.CodePage = nValue oXMLField.CodePage = nNvalue
DeclareXMLPrefix	Объявляет объект XMLAdapter XMLPrefix для объекта, включенного в синтаксический анализ
DisableEncode	Разрешает или запрещает кодирование base64 или hexBinary для memo или символьных полей
ForceCloseTag	При установке свойства в "истину" позволяет замещать тег одиночного элемента набором открывающего и закрывающего тегов, например — <elem1></elem1>. По умолчанию установлено в .F.
FormattedOutput	Позволяет выводить либо форматированные строки XML (.T.), либо в виде единственной непрерывной строки. По умолчанию используется форматированный вывод
IsDiffGramm	Указывает, связан объект XMLAdapter с .NET DiffGram
IsLoaded	Указывает, связан ли объект XMLAdapter с документом XML
IXMLDomElement	Содержит объектную ссылку на объект IXMLDOMElement после успешного исполнения методов LoadXML или Attach объекта XMLAdapter
Mapbinary	Разрешает разметку типов данных из типов данных XML в типы данных Visual FoxPro Varbinary Blob
MapN19_4ToCurrency	По умолчанию Visual FoxPro размечает xsd:decimal в Numeric. Однако вы можете сохранить точность типа данных "currency" разметкой xsd:decimal в Currency вместо разметки в Numeric. Например, поля валюты SQL Server представлены в XML как десятичные числа 19,4 общим числом в 19 цифр и четырьмя числами после запятой
MapVarchar	Свойство MapVarchar объекта CursorAdapter разрешает используемую по умолчанию разметку типов данных из типов данных ODBC и ADO в Visual FoxPro Varchar тип данных

NoCpTrans	Определяет — будут ли поля символьного и memo-типов создаваться как поля типов Character (Binary) или Memo (Binary)

***Таблица 8.15** (продолжение)*

Свойство	Описание
PreserveWhiteSpace	Сохраняет или удаляет пробелы из XML-документа
RespectCursorCP	Определяет — будет ли принята во внимание кодовая страница курсора
RespectNesting	Определяет — должен ли метод ToXML использовать вложение таблиц на основе значения свойства NestedInto Property, и должно ли вложение учитываться в течение разбора XML-документа
SelectionNamespaces	Определяет декларацию XMLNamespace для выражений XPath
SOM	Содержит объектную ссылку на Объектную Модель Схемы XML-документа (SOM) (ISchema) после успешного выполнения метода LoadXML или метода Attach
Unicode	Указывает логическое значение, которое определяет, как будут обрабатываться строки в зависимости от исполняемого метода или как они будут храниться
UseCodePage	Определяет — должен ли объект XMLAdapter использовать специальную логику для определения — какая кодовая страница должна быть использована для кодирования или декодирования данных
UTF8Encoded	Определяет, будет ли выполнена кодировка установки в Unicode Transformation Format-8 (UTF-8). UTF8Encoded работает совместно со свойством RespectCursorCP
WrapCharInCDATA	Определяет перенос на следующую строку символьных полей в секции CDATA. WrapCharInCDATA применяется только при выполнении следующих методов: • LoadXML • Attach • AddTableSchema
WrapMemoInCDATA	Определяет перенос на следующую строку memo-полей в секции CDATA
XMLConstraints	Содержит объектную ссылку на ISchemaItemCollection-объект после того, как успешно выполнен LoadXML или Attach-метод
XMLName	Имеет различное значение для разных элементов. XMLAdarpter XMLName содержит имя элемента, который определяет табличный элемент. • XMLTable: XMLName содержит имя элемента — таблицы в документе XML. • XMLField: XMLName содержит элемент или имя атрибута этой таблицы в документе XML

XMLNameIsXPath	Определяет, будет ли свойство XMLName содержать выражение XPath

Таблица 8.15 *(окончание)*

Свойство	Описание
XMLNamespace	XMLAdapter: XMLNamespace содержит XML Namespace, которому принадлежит XMLAdapter XMLName, если любое. XMLTable: XMLNamespace содержит XML Namespace, которому принадлежит XMLTable XMLName. Если свойство пустое, то вместо него используется значение из свойства XMLAdapter Namespace
XMLPrefix	Содержит префикс, используемый, чтобы ссылаться на свойство XMLNamespace для соответствующего объекта
XMLSchemaLocation	Указывает информацию о схеме

Таблица 8.16. *Методы класса XMLAdapter*

Метод	Описание
AddTableSchema	Добавляет новый объект XMLTable в коллекцию XMLAdapter Tables и необходимые объекты XMLField в коллекцию XMLTable Fields на основе указанного псевдонима таблицы
Attach	Прикрепляет существующий документ XML Document Object Model (DOM) или DOM элемент, и необязательно, XML-схему к объекту XMLAdapter. Attach, кроме того, устанавливает свойство IXMLDOMElement объекта XMLAdapter
LoadXML	Загружает XML из файла или строки в Объектную Модель Документа (DOM) и подключает это к объекту XMLAdapter
ReleaseXML	Освобождает прикрепленный объект IXMLDOMElement
ToXML	Выводит XML-строку в файл или переменную памяти

Пример работы с классом XMLAdapter

ЗАДАНИЕ. Создать XML-документ.

Необходимо вывести данные в XML-документ в следующем формате:

Тег	Вложенный тег	Значение
Company		Данные организации, подающей сведения
	Person	
Person		Персональные данные о лице, на которое подаются сведения

	Zarpl	
Zarpl		Заработная плата

Описание атрибутов тегов:

Имя тега	Имя атрибута	Назначение
Company		
	Name	Полное наименование организации-отправителя
	INN	ИНН организации-отправителя
	Data_otpr	Дата отправки данных
Person		
	ID	Цифровой идентификатор физического лица
	FIO	Фамилия, имя, отчество
	Adres	Адрес
	Datar	Дата рождения
	Pasp_ser	Серия паспорта
	Pasp_num	Номер паспорта
Zarpl		
	Vid	Вид дохода
	Sum	Сумма дохода
	Month	Месяц дохода
	Year	Год дохода

Поскольку курсора Company у нас нет, создадим его:

```
CREATE CURSOR company (name c(60), inn c(10), date_otpr d(8))
INSERT INTO Company name, inn, date_otpr;
Value ("Организация_наименование",;
INN="9999999999",date())
```

Курсоры Person и Zarpl у нас имеются, они выбраны из источника данных:

Person, содержащий поля

```
ID
FIO
Adres
Birthday
Pasp_Ser
Pasp_Num
```

(INN — добавляется для связи с курсором Company)

и курсор Zarpl с полями

```
ID
Vid
Sum
Month
Year
Dolgn
Dolgn_history
```

Прежде всего, должны быть установлены отношения между курсорами. Если связь между курсорами Person и Zarpl просматривается (в виде идентификатора ID), то поле, по которому можно связать курсоры Company и Person, отсутствует. Мы создадим его искусственно, добавив поле INN в курсор Person и заполнив это поле значением 9999999999, равным Company.INN. Это делается для более простого преобразования данных из XML-документа обратно в курсор (листинг 8.5).

```
SET RELATION TO ID   INTO Zarpl   IN person
SET RELATION TO inn INTO person  IN company
```

```
LOCAL loXA as XMLAdapter
loXA = NEWOBJECT('XMLAdapter')  && Создаем объект XMLAdapter
WITH loXA
.RespectCursorCP=1251
.RespectNesting = .T.
        .AddTableSchema('company',.f.)
        .AddTableSchema('person',.f.)
        lcName= .Tables(2).Fields(7).xmlname
        .Tables(2).fields.remove(lcName)
        .AddTableSchema('zarpl',.f.)
        lcName1=.Tables(3).Fields(7).xmlname
        .Tables(3).fields.remove(lcName1)
        lcName2=.Tables(3).Fields(6).xmlname
        .Tables(3).fields.remove(lcName1)
        .ForceCloseTag= .T.
        .XMLSchemaLocation = 'xmlout.xsd'
        .ToXML('d:\xml\out\XmlOut', 'XmlOut', .T.)
        .XMLSchemaLocation=''
ENDWITH
```

Это означает следующее (табл. 8.17).

Таблица 8.17. *Описание команд листинга 8.5*

Команда	Описание
.RespectCursorCP=1251	Устанавливает кодировку согласно кодовой страницы курсора
.RespectNesting = .T.	Если таблица является вложенной, ее узлы должны быть дочерними к узлу внешней таблицы

***Таблица 8.17** (окончание)*

Команда	Описание
.AddTableSchema('company',.f.)	Добавляет курсор Company
lcName= .Tables(2).Fields(7).xmlname	Определяется поле для удаления
.Tables(2).fields.remove(lcName)	Добавочное поле INN не подлежит выводу в выходной XML-документ, поэтому удаляется
.AddTableSchema('zarpl',.f.)	Добавляет курсор Person
lcName1=.Tables(3).Fields(7).xmlname	Определяется поле для удаления
.Tables(3).fields.remove(lcName1)	Удаляем ненужное в выходном документе поле Dolgn_History
lcName2=.Tables(3).Fields(6).xmlname	Определяет поле для удаления
.Tables(3).fields.remove(lcName2)	Удаляем ненужное в выходном документе поле Dolgn
.ForceCloseTag= .T.	Для улучшения читабельности документа устанавливаем соответствие открывающего и закрывающего тега в виде <Zarpl ..... />
.XMLSchemaLocation = 'xmlout.xsd'	Указывает информацию о схеме
.ToXML('d:\xml\out\XmlOut','XmlOut', .T.)	Генерирует XML в файл или переменную памяти
.XMLSchemaLocation=''	Если .XMLSchemaLocation="", то LoadXML не выполняет: • обработку любых схем во время загрузки; • изменение коллекции Tables; • установку свойства IsDiffGram; • ToXML не генерирует никаких схем

Вывод XML-данных в Internet Explorer

Вопрос, который возникает у многих программистов, изучающих XML: какие действия нужно предпринять, чтобы вывести данные из XML-документа в Internet Explorer? Для того чтобы отобразить XML-файл в IE, необходимо определить формат отображения данных. Как правило, производится преобразование DBF — XML — HTML.

Программа преобразования написана Михаилом Дроздовым (листинг 8.6).

```
*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
*~~ Author:  Michael Drozdov, ICS Soft, Perm, Russia
*~~ My Page: http://vfpdev.narod.ru/
*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
*
#DEFINE C_PATH          "d:\char8\dbftohtm\"  &&<- change as your need
#DEFINE C_TABLE         "person" && "supplier"/"orders"/"employee"/"customer" ...
#DEFINE C_HTM           "out.htm"
#DEFINE N_ROWFLDS       1  && or what you need, but <> 0
#DEFINE C_LANG          "ru" && "en" or "ru" &&<- change as your need
*
*/////////////////////////////////////////////////////////////////
*-- Change constants below, according to specific of your language
#IF C_LANG = "ru"
        *
        *-- For Russian only:
        #DEFINE C_XMLPI                 '<?xml version="1.0" encoding="Windows-
1251" standalone="yes" ?>'
        #DEFINE C_TITLEPREF_LOC "Таблица "
#ELSE
        #DEFINE C_XMLPI                 '<?xml version="1.0" encoding="Windows-
1252" standalone="yes" ?>'
        #DEFINE C_TITLEPREF_LOC "Table "
#ENDIF
*=============== Attention!!!
* If yours language is not Russian,
* change attribute encoding="windows-1251" in element <xsl:output .../>
* according to specific of your language in files: xmltohtml.xslt,
getfields.xslt
* and see also [CTRL+F3 Russian] in file xmltohtml.xslt
*=============== Attention!!!
*/////////////////////////////////////////////////////////////////

#DEFINE C_XSLHTML       "xmltohtml.xslt"
#DEFINE C_XSLFLDS       "getfields.xslt"
```

```
#DEFINE C_XSD           "curSchema.xsd"
#DEFINE C_XML           "lcXml"
#DEFINE SW_SHOWNORMAL   1

*/////////////////////////////////////////////////////////////////
*-- For debug: save current data as files
#DEFINE C_XMLFLDS       ""      &&<- not empty for debug only ("curFields.xml" as
example)
#DEFINE C_XMLOUT        ""      &&<- not empty for debug only ("curOut.xml" as
example)
*/////////////////////////////////////////////////////////////////

IF !DIRECTORY(C_PATH)
        MESSAGEBOX("Path: '" + C_PATH + "' not found.", 16, _SCREEN.Caption)
        RETURN .F.
ENDIF

PUBLIC gbIsApiDefs As Boolean
*
*-- MSXML PROGIDs
LOCAL lcMsxmlVersion As String ;
        ,lcPROGIDmsxml As String ;
        ,lcPROGIDmsxsl As String ;
        ,lcPROGIDmstemp As String
lcMsxmlVersion = "4.0"
*********************************************************
* lcMsxmlVersion = "3.0" && Also you can use 3.0 versions
* but in file: xmltohtml.xslt change
* finction msxsl:format-date() to user:get_Date()
* according to comments there
*********************************************************
lcPROGIDmsxml = "Msxml2.DOMDocument." + lcMsxmlVersion
lcPROGIDmsxsl = "Msxml2.FreeThreadedDOMDocument." + lcMsxmlVersion
lcPROGIDmstemp = "Msxml2.XSLTemplate." + lcMsxmlVersion

*
*-- Open table
SET DEFAULT TO (C_PATH)
IF !USED(C_TABLE)
        USE (C_TABLE + '.dbf') IN 0 SHARED
        IF !USED(C_TABLE)
                RETURN .F.
        ENDIF
ENDIF

*
*-- Clear old-result
```

```
IF FILE(C_HTM)
        ERASE (C_HTM)
ENDIF

*
*-- Convert DBF to XML
LOCAL lnLength as Number;
        ,lcVarXml as String
lcVarXml = C_XML
&lcVarXml = ""
IF FILE(C_XSD)
        ERASE (C_XSD)
ENDIF
lnLength = CURSORTOXML(C_TABLE, C_XML, 1 , 16, 0, C_XSD)
IF !EMPTY(C_XMLOUT)
        *
        *-- For debug only
        IF FILE(C_XMLOUT)
                ERASE (C_XMLOUT)
        ENDIF
        STRTOFILE(EVALUATE(C_XML), C_XMLOUT)
ENDIF
IF USED(C_TABLE)
        USE IN (C_TABLE)
ENDIF

*
*-- Create MSXML objects
LOCAL loMsXml As Object ;
        ,loXmlFlds As Object ;
        ,loMsXsl As Object ;
        ,loMsPrc As Object ;
        ,lobjXSLTProc As Object
loMsXml = CREATEOBJECT(lcPROGIDmsxml)
IF VARTYPE(loMsXml) # 'O'
        RETURN .F.
ENDIF
loXmlFlds = CREATEOBJECT(lcPROGIDmsxml)
loMsXsl = CREATEOBJECT(lcPROGIDmsxsl)
loMsPrc = CREATEOBJECT(lcPROGIDmstemp)

*//////////////////////////////////////////
*// Step1: get list of fields from XSD-file
*//
LOCAL lnLen as Number;
        ,cOut as String
lcOut = ""
```

```
lnLen = 0
IF FILE(C_XSD)
        loMsXml.async = .F.
        loMsXml.load(C_XSD)
        loMsXsl.async = .F.
        loMsXsl.load(C_XSLFLDS)
        loMsPrc.stylesheet = loMsXsl.documentElement
        lobjXSLTProc = loMsPrc.createProcessor()
        IF VARTYPE(lobjXSLTProc) = "O"
            lobjXSLTProc.input = loMsXml
            *
            *-- Transform (XML + XSLT) -> XML
            IF lobjXSLTProc.transform()
                lcOut = lobjXSLTProc.output
                lnLen = LENC(lcOut)
            ENDIF
            lobjXSLTProc = NULL
        ENDIF
        loMsXsl = CREATEOBJECT(lcPROGIDmsxsl)
        IF EMPTY(C_XMLFLDS)
                ERASE (C_XSD)
        ELSE
                *
                *-- For debug only
                IF FILE(C_XMLFLDS)
                        ERASE (C_XMLFLDS)
                ENDIF
                STRTOFILE(C_XMLPI + lcOut, C_XMLFLDS)
        ENDIF
ENDIF
LOCAL lnRowForFields as Number
lnRowForFields = N_ROWFLDS
IF lnLen > 0
        loXmlFlds.loadXML(C_XMLPI + lcOut)
        lnRowForFields = 0
ENDIF

*////////////////////////////////
*// Step2: convert XML to HTML
*//
lcOut = ""
lnLen = 0
loMsXml.async = .F.
loMsXml.loadXML(EVALUATE(C_XML))
&lcVarXml = ""
loMsXsl.async = .F.
loMsXsl.load(C_XSLHTML)
loMsPrc.stylesheet = loMsXsl.documentElement
```

```
lobjXSLTProc = loMsPrc.createProcessor()
IF VARTYPE(lobjXSLTProc) = "O"
    lobjXSLTProc.input = loMsXml
    *
    *-- Set parameters values
    lobjXSLTProc.addParameter("prmLang", C_LANG)
    lobjXSLTProc.addParameter("prmTitlePrefLoc", C_TITLEPREF_LOC)
    lobjXSLTProc.addParameter("prmRowForFields", lnRowForFields)
    IF lnRowForFields = 0
        lobjXSLTProc.addParameter("prmDocFields", loXmlFlds)
    ENDIF
    *
    *-- Transform (XML + XSLT) -> HTML
    IF lobjXSLTProc.transform()
        lcOut = lobjXSLTProc.output
        lnLen = LENC(lcOut)
    ENDIF
ENDIF
STORE NULL TO ;
        loMsXml, loMsXsl, loMsPrc, lobjXSLTProc, loXmlFlds

*////////////////////////////////
*// Step3: show result
*//
IF lnLen > 0
        *
        *-- Save result
    IF STRTOFILE(lcOut, C_HTM) > 0
        lcOut = ""
        *
        *-- Show result
                LOCAL lnCode As Number
                IF !gbIsApiDefs
                        DECLARE LONG ShellExecute IN shell32.dll ;
                                LONG hwnd;
                                ,STRING lpOperation;
                                ,STRING lpFile;
                                ,STRING lpParameters;
                                ,STRING lpDirectory;
                                ,LONG nShowCmd
                        gbIsApiDefs = .T.
                ENDIF
                lnCode = ShellExecute(0, "open", C_HTM, NULL, NULL,
SW_SHOWNORMAL)
    ENDIF
ENDIF
*
```

```
*-- Clear all
IF gbIsApiDefs
        CLEAR DLLS
        gbIsApiDefs = .F.
ENDIF
*
```

Пример преобразования таблицы Person находится на компакт-диске в каталоге CHAR8\dbftohtm.

Парсеры (Анализаторы)

Парсер (Parser) — программа, считывающая введенный текст и анализирующая его значение. Такие программы еще называют анализаторами. Существуют два типа анализаторов:

- не проверяющие на допустимость (Non-validating). Такие анализаторы просто контролируют, чтобы документы были правильно оформленными (well-formed);
- проверяющие на допустимость (Validating). С помощью декларации типа документа (DTD) такие анализаторы проверяют допустимость формы и содержания правильно оформленного XML-документа.

По способу своей работы анализаторы также делятся на два разных типа:

- событийно управляемые;
- древовидные.

Событийно управляемые анализаторы

Принцип работы такого типа анализатора основан на обращениях к функциям обратного вызова (call-back), обрабатывающего приложения всякий раз, когда при обработке документа начата либо завершена обработка какого-нибудь структурного объекта документа: элемента, атрибута, символьных данных, команды обработки, нотации или комментария. Программирование же приложения заключается в написании реализации этих функций обратного вызова, чтобы выполнить требуемую обработку. Такие анализаторы не обслуживают работу с иерархической структурой данных как таковой, а просто последовательно читают документ, генерируя соответствующие события и передавая информацию об объекте, который был почитан.

К их достоинствам можно отнести как компактность, так и очень высокую скорость работы, с которой они способны обработать даже очень большие XML-документы.

Древовидные анализаторы

Данные XML-документа по структуре представляют собой иерархическое дерево. Поэтому при путешествии по узлам, поиске его содержания или редактировании можно использовать хорошо разработанные алгоритмы. Для осуществления этой за-

дачи консорциум W3C разработал объектную модель документа Document Object Model (DOM), (**http://www.w3.org/DOM/**). Эта модель представляет собой независимый от языка или платформы интерфейс, позволяющий совершать весь необходимый набор манипуляции со структурой XML-документа.

Особенность древовидных анализаторов такова, что дерево XML-документа они выстраивают в памяти компьютера, и для больших документов для этого требуется высокопроизводительная поддержка работы с виртуальной памятью компьютера. Далее, через соответствующее API-приложение может осуществлять навигацию по иерархическим данным документа, производить поиск, редактирование и другие операции над данными.

Использование DomDocument

Существует общая *объектная модель*, применимая для любого XML-документа, вне зависимости от его *структуры*. Эта модель называется *DOM (Document Object Model)*. Объектная модель XML-документа состоит из ряда объектов, представляющих различные компоненты XML-документа. DOM хранит данные в древообразной структуре, отражающей иерархическую структуру XML-документа. Свойства и методы объектов DOM позволяют предоставлять доступ к любым компонентам XML-документа, включая элементы, атрибуты, инструкции по обработке, комментарии и объявления нотаций и примитивов.

Программные объекты, представляющие XML-документ, называются узлами. DOM использует различные типы узлов для представления различных компонентов XML (табл. 8.18).

Таблица 8.18. *Типы узлов DOM*

Тип узла	Описание	NodeName	NodeValue
`Document`	Корневой узел документа	`#document`	`null`
`Element`	Элемент	Имя типа элемента	`null`
`Text`	Текст элемента	`#text`	Текст родительского XML-компонента
`Attribute`	Атрибут, имя и значение	Имя атрибута	Значение атрибута
`Processing-Instruction`	Инструкция по обработке	Инструкция	Содержимое инструкции
`Comment`	Комментарий	`#comment`	Комментарий
`CDATASection`	Раздел `CDATA`	`#cdata-section`	Содержимое раздела `CDATA`
`DocumentType`	Объявление типа документа	Имя корневого элемента	`null`

Entity	Объявление примитива в DTD	Имя примитива	null
Notation	Объявление нотации в DTD	Имя нотации	null

Вы можете получить имя каждого узла из свойства узла nodeName. Также вы можете получить значение каждого узла из свойства узла nodeValue. Если компонент XML непосредственно не имеет значения, DOM устанавливает в качестве значения null.

Каждый узел, как программный объект, имеет свойства и методы. Все типы узлов, помимо специфического собственного набора свойств и методов, используют общий набор свойств и методов. Вот некоторые полезные свойства, поддерживаемые всеми типами узлов (табл. 8.19).

Таблица 8.19. *Некоторые свойства узлов DOM*

Свойство	Описание	Пример
attributes	Коллекция NamedNodeMap всех дочерних узлов-атрибутов	AttributeNode = Element.attributes.getNamedItem ("ItemName")
childNodes	Множество NodeList всех дочерних узлов-неатрибутов данного узла	FirstNode = Element.childNodes(0)
dataType	Тип данных узла-атрибута, если этот тип данных определен в DTD	AttributeType = Attribute.dataType
firstChild	Первый дочерний узел данного узла, не являющийся атрибутом	FirstChildNode = Element.firstChild
lastChild	Последний дочерний узел данного узла, не являющийся атрибутом	LastChildNode = Element.lastChild
nextSibling	Следующий узел (любого типа) на том же уровне данного узла	NextElement = Element.nextSibling
nodeName	Имя узла	ElementName = Element.nodeName
nodeType	Тип узла (числовое значение)	NodeTypeCode = Element.nodeType
nodeTypeString	Тип узла (строка)	NodeTypeString = Element.nodeTypeString
nodeValue	Значение данного узла или null, если узел не имеет значения	AttributeValue = Attribute.nodeValue
ownerDocument	Корневой узел документа, содержащего данный узел	Document = Element.ownerDocument

parentNode	Узел, для которого данный узел является дочерним (не действует для атрибутов)	parentElement = Element.parentNode
previousSibling	Предыдущий узел (любого типа) на том же уровне данного узла	PreviousElement = Element.previousSibling
Text	Все текстовое содержимое данного узла, включая подчиненные узлы	AllCharacterData = Element.text
xml	Все XML-содержимое данного узла, включая подчиненные узлы	XMLContent = Element.xml

Пример.

```
*Создаем объект парсера:
oXMLDom=CREATEOBJECT("MSXML2.DOMDocument.4.0")
* Загружаем данные
oXMLDom.load("prim1.xml") && или
*? oXMLDom.load("Prim1.xml")
*- кол-во узлов
oIDs=oXMLDOM.getElementsByTagName('id')
? oIDs.length
oIDs=oXMLDOM.selectNodes('VFPData//id')
? oIDs.length
? oIDs.item(0).text
? oIDs.item(1).text
<…>
```

XSLT

XSLT *(Extensible Stylesheet Language Transformations)* — часть спецификации XSL, задающая язык преобразований XML-документов. Спецификация XSLT является рекомендацией консорциума W3C.

При применении *таблицы стилей* XSLT, состоящей из набора *шаблонов*, к XML-документу *(исходное дерево)*, образуется *конечное дерево*, которое может быть как XML-структурой, так и обычным текстом. Запросы выбора данных из исходного дерева пишутся на языке запросов XPAth.

XSLT применяется в основном в области Web-программирования.

Консорциум W3 определяет три составные части языка XSL (от англ. eXtensible Stylesheet Language — Расширяемый Язык Стилей): XSLT, XPAth (язык путей и выражений, используемый в XSLT для доступа к отдельным частям XML-документа) и

XSL Formatting Objects — словарь, определяющий семантику форматирования документов.

XSLT позволяет трансформировать одни документы в другие, пользуясь простыми наборами правил преобразования.

Использование XSLT может обеспечить легкую и гибкую интеграцию в проекты. Если одним из этапов процедуры обмена XML-данными будет XSLT-преобразование, расширение количества форматов, известных системе, будет производиться не дописыванием исходного кода, а добавлением преобразований. XSLT обеспечивает простой, удобный и легко настраиваемый вывод фрагментов HTML.

Получение выходного HTML-кода получается путем преобразования XSLT-процессором входных XML-данных по XSL-шаблону. Соответственно, умение организовать правильный выходной поток, т. е. HTML-код, состоит из умения правильно организовать XML-данные и умения правильно писать шаблоны.