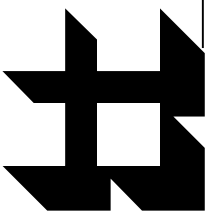

ГЛАВА 9

Отображение табличных данных на форме

В процессе работы с базами данных приходится делать запросы к таблицам и полученные результаты визуально отображать на экране. Поэтому Visual FoxPro должен иметь в своем арсенале средства для визуального отображения таблиц, курсоров и адаптеров курсоров. Такие средства в нем, разумеется, есть. Если в версиях FoxPro 2.x для отображения данных использовалась команда `BROWSE`, то в более поздних версиях используется элемент управления `Grid`, а `Browse` используется только для отладки. Использование `Grid` позволяет представлять данные в виде таблицы, при этом пользователь видит сразу несколько записей. Кроме объекта `Grid`, форма использует объект `DataEnvironment`, чтобы подключать таблицы к объектам, визуально их отображающим. Все таблицы, которые вы поместите в `DataEnvironment`, будут открываться при вызове формы, соблюдая установленные связи, и закрываться при ее закрытии.

DataEnvironment (Среда данных)

Среда данных (DE) открывается при создании формы, набора форм или отчета. Она позволяет добавить в форму, набор форм или отчет источник данных и определить для работы с ним свойства и методы. Среда данных открывается перед загрузкой самой формы, и в ней открываются источники данных. Если данные будут определены в `DataEnvironment`, вам больше не придется заботиться об открытии и закрытии таблиц, а также об организации отношений между ними — все необходимые действия будут выполнены автоматически.

Объект `DataEnvironment` — это контейнер, он может включать в себя другие объекты:

- курсоры;
- адаптеры курсоров;
- объекты `Relation`.

Объект `Cursor` создается при добавлении в `DataEnvironment` таблицы, представления или курсора. При добавлении таблицы (объекта данных) в DE новый табличный объ-

ект называется `CURSOR` + свободный порядковый номер (не обязательно последний), например, `Cursor1`. Объект `Cursor` обладает рядом собственных свойств и методов, среди которых можно отметить `Filter` и `Order`. Свойство `Filter` исключает из рассмотрения записи, которые не удовлетворяют критерию, определенному данным выражением.

Установка фильтра может быть произведена программно:

```
DataEnvironment.Cursor1.Filter=nVal
```

Свойство `Order` определяет управляющий индексный тег для объекта `Cursor`:

```
DataEnvironment.Cursor1.Order=TagName
```

У курсора на основе представления (view) имеется свойство `NoDataOnLoad`, которое позволяет в начале загрузки формы не запрашивать данные из источника (для уменьшения сетевого трафика).

Добавить таблицу в среду данных можно в Конструкторе форм, выбрав в контекстном меню пункт — **Data Environment**. В открывшемся окне конструктора среды данных, в контекстном меню выберите пункт **Add**. В открывшемся диалоге выберите таблицу или представление (view). Если база данных не открыта или проект не содержит базу данных, для добавления свободных таблиц нажмите **Other**.

Если есть несколько одинаковых по структуре таблиц, в курсор DE можно добавить только одну, а доступ к данным определять изменением значения `CursorSource`.

Удалить таблицу или представление из среды данных можно кнопкой **Delete** (предварительно выделив таблицу или представление) или в меню `Data Environment` выбрать команду `Remove`.

Объект `CursorAdapter` — это буферизованный курсор, представляющий данные одного или нескольких источников. В качестве `CursorAdapter` могут выступать собственные таблицы VFP, удаленные представления, ActiveX-источники данных и XML-документы.

Объект `Relation` создается при установлении отношений между курсорами среды данных.

Организовать связи (отношения) между курсорами можно, перетаскивая поле из первичного курсора на соответствующий индексный тег подчиненного курсора. При этом Visual FoxPro предложит сформировать индекс для подчиненного поля, если этот индекс будет отсутствовать. Когда вы устанавливаете связь (отношение) в конструкторе среды данных, на нее будет указывать графическая линия между таблицами. Если ее выделить мышью, и нажать кнопку **Delete**, связь будет удалена. Для редактирования свойств связей в окне свойств в списке объектов выберите связь (Relation) или при открытом окне свойств (Properties) выберите мышью связь между курсорами (линия должна выделиться).

Конструктор *Data Environment*

Для визуального создания и модификации окружения данных форм, наборов форм и отчетов можно использовать конструктор **Data Environment**. Для форм Visual FoxPro заполняет список свойств **ControlSource**, который можно просмотреть в окне **Properties** со всеми полями в вашем окружении данных. Для отчетов Visual FoxPro заполняет наборы данных, требуемые для вашего отчета и основанные на связанных таблицах или представлениях.

При открытом окне Конструктора **Data Environment** в главном меню появляется новый пункт **Data Environment**, который так же, как и контекстное меню, позволяет добавлять в среду данных таблицы и просматривать их.

Чтобы открыть окно конструктора **Data Environment**, на свободном от элементов управления участке формы нажмите правую кнопку мыши и в появившемся контекстном меню выберите пункт **Data Environment**. Окно среды данных будет открыто, и вам сразу же будет предложено указать таблицу для включения ее в DE.

Второй путь — использовать главное меню: **Data Environment | Add**.

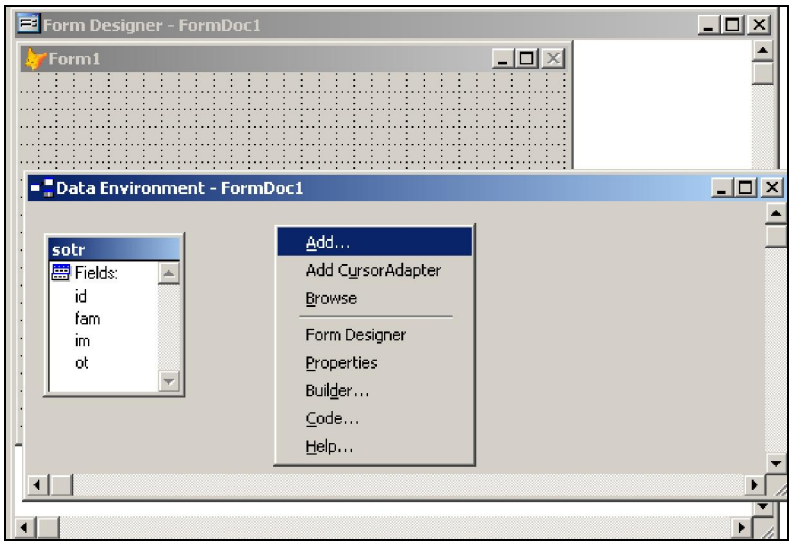

Рис. 9.1. Конструктор **Data Environment**

При этом будет открыто диалоговое окно **Add Table or View** (рис. 9.2), содержащее список таблиц открытой базы данных. Опция **Views** области **Select** позволяет выбрать созданные в базе данных представления. Если вы не обнаружили нужной таблицы в списке, нажмите кнопку **Other** и, используя диалоговое окно **Open**, найдите требуемый объект.

После того как нужные таблицы выбраны и размещены в среде данных, необходимо упорядочить данные, для этого нужно установить **Order**, т. е. управляющий индексный тег для объекта `Cursor`.

Для этого нужно выполнить следующие действия:

1. Установите курсор на таблицу, размещенную в среде данных.
2. Нажмите правую кнопку мыши и выберите из контекстного меню пункт **Properties**.
3. Выделите свойство `Order`.
4. В раскрывающемся списке выберите необходимый вам тег и установите его.

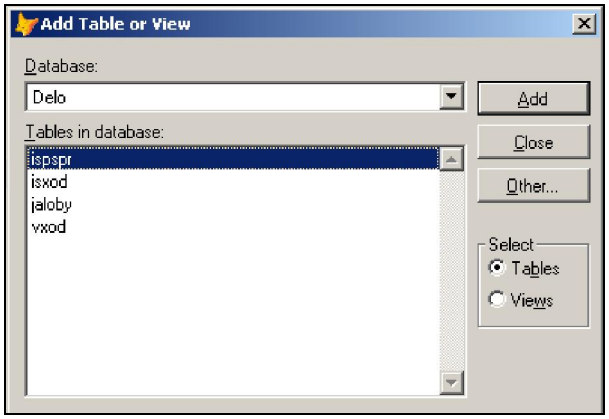

Рис. 9.2. Диалоговое окно **Add Table or View**

Свойства, события и методы объекта *DataEnvironment*

Если выбрать в контекстном меню **Data Environment** пункт **Properties**, мы получим доступ к свойствам, событиям и методам DE. Список свойств, событий и методов вы можете увидеть в табл. П4.19 *приложения 4*.

Свойства связи соответствуют командам `SET RELATION` и `SET SKIP`.

Свойство `RelationExpr` содержит по умолчанию имя первичного ключевого поля (primary) родительской таблицы. Если связанная таблица индексирована с использованием выражения, вы должны указать в свойстве `RelationExpr` это выражение.

Например, если связанная таблица проиндексирована с помощью `UPPER` (cust_id), вы должны установить `RelationExpr` в `UPPER` (cust_id). Если связь "один ко многим" не нужна, установите свойство `OneToMany = (.F.)`. Это соответствует использованию `SET RELATION` без применения `SET SKIP`. Установка свойства `OneToMany = (.T.)` соответствует команде `SET SKIP`.

Обратите внимание, если вы хотите установить отношения "один-ко-многим" (OneToMany) в форме с использованием базы данных, установите свойство OneToMany = (.T.), даже если постоянные отношения "один-ко-многим" уже существуют в базе данных.

Автоматическое открытие таблиц, включенных в DataEnvironment, определяется значением свойства AutoOpenTables, а автоматическое закрытие, соответственно, свойством AutoCloseTables. Когда свойство AutoOpenTables установлено в True (.T.), DataEnvironment автоматически вызывает метод AutoOpen любого объекта CursorAdapter, содержащегося в нем. DataEnvironment открывает любые курсоры или курсоры-адаптеры в том порядке, в котором они были введены в окружение данных.

Когда свойство AutoOpenTables установлено в False (.F.), вы можете вызвать метод OpenTables объекта DataEnvironment для программной загрузки данных в DataEnvironment.

Свойство InitialSelectedAlias указывает имя текущей рабочей области.

Во время выполнения могут быть установлены только следующие свойства объекта DataEnvironment: DataSource, DataSourceType, InitialSelectedAlias, Name, OpenViews и Tag. Чтобы новое значение свойства вступило в действие, необходимо вызвать методы CloseTables и OpenTables объекта DE. Попытка изменить другие свойства во время выполнения программы генерирует ошибку. Значения свойств AutoOpenTables и AutoCloseTables могут быть изменены только на этапе проектирования.

На базе класса DataEnvironment можно определить подкласс:

```
DEFINE CLASS DtEn As DataEnvironment
...
ENDDEFINE
```

Можно сохранять DataEnvironment как библиотеку классов. Для этого, находясь в DE, выберите пункт меню **Save As Class**, и среда данных будет сохранена в VCX-файле. Все свойства, методы и события DataEnvironment доступны в окне **Properties** после того, как вы откроете Конструктор классов (Class Designer). Однако вы можете обратиться к Конструктору среды данных (DataEnvironment Designer) только из Конструктора форм (Form Designer), но не из **Class Designer**.

Во время разработки вы можете добавить ранее существовавший класс DataEnvironment на форму из инструментальной панели библиотеки классов или в пользовательском коде, используя метод AddObject. Однако Visual FoxPro не рассматривает класс DataEnvironment, который добавлен любым способом как член класса, как собственное окружение формы.

Для того чтобы загрузить внешний класс DataEnvironment, можно использовать свойства формы DEClass и DEClassLibrary:

```
DEFINE CLASS My_Form AS Form
        DEClassLibrary="My_DE.prg"
```

```
        DECLASS = "My_DE"
ENDDEFINE
```

Пример 1. Создание среды данных для нескольких таблиц, связанных отношениями

Необходимо разместить в `DataEnvironment` таблицы Vxod.dbf, Ispol.dbf, Otvet.dbf, Oznak.dbf.

Первая таблица представляет собой перечень входящих документов и связана с остальными таблицами по полю `Nvxod`.

1. Создаем форму и открываем Конструктор среды данных.
2. Добавляем на форму таблицу **Vxod**, и следом за ней таблицы **Ispol**, **Otvet** и **Oznak**.
3. Устанавливаем отношения между таблицами. Более подробно об установке отношений *см. в главе 6*.
4. На рис. 9.3 изображен результат этих действий: таблицы добавлены в среду данных, отношения установлены, свойство `Order` установлено автоматически.

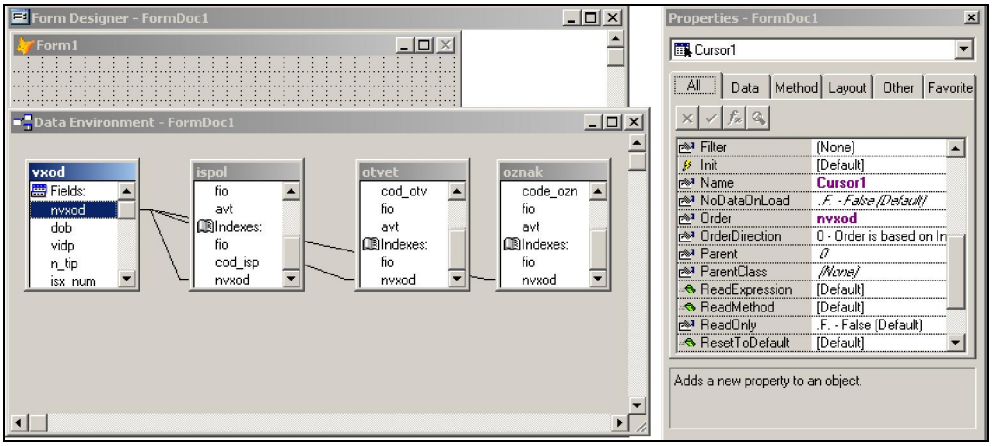

Рис. 9.3. Создание среды данных с несколькими таблицами, связанными отношениями

Пример 2. Создание среды данных для *CursorAdapter*

1. Создаем форму и открываем окно создания **CursorAdapter** (рис. 9.4).
2. Выбираем в контекстном меню пункт **Builder** (Построитель).
3. Адаптер курсора содержит три вкладки — **Properties**, **Data Access** и **Auto-Update**.
4. На первой вкладке указываем имя источника данных — адаптер курсора `CurVxod`.
5. На второй вкладке формируем команду `Select` для отображения данных:

```
SELECT nvxod, isx_num, data_isx, data_Reg, resol from vxod
```

6. На третьей вкладке формируем параметры автоматического обновления записей в адаптере курсора.
7. В свойствах `DataEnvironment` должен быть указан тип данных `RecordSourceType`. В нашем случае это `NATIVE`.

При необходимости переопределять источник данных для `CursorAdapter` для удаленных данных в методе `Init() Cursor Adapter` необходимо добавить код, для SQL Server код будет таким:

```
LOCAL lcSQLconnect
lcSQLconnect = "DRIVER=SQL Server;SERVER=" + gcServer + ";UID=" + gcUid + ;
";PWD=" + gcPwd + ";DATABASE=" + gcDatabase
THIS.DataSource = SQLSTRINGCONNECT(lcSQLconnect,.T.)
```

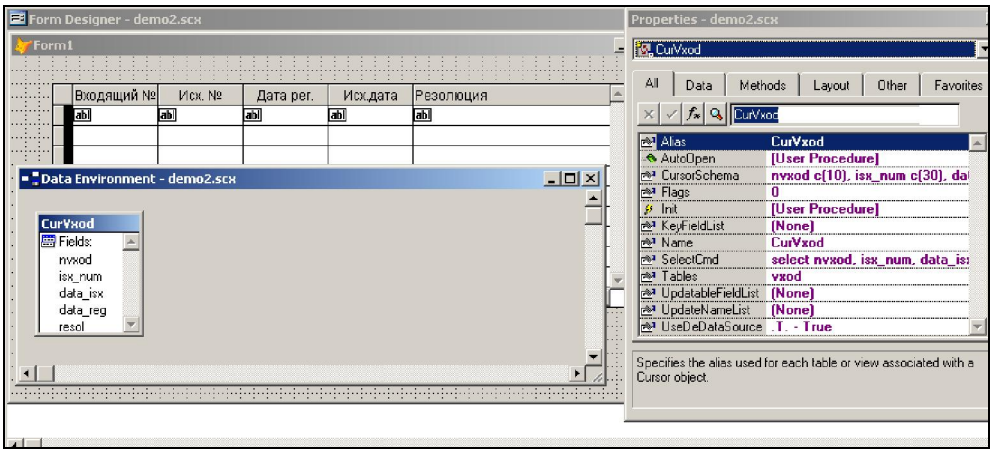

Рис. 9.4. Data Environment для **Cursor Adapter**

Здесь:

`gcServer` — имя сервера;

`gcUid` — логин;

`gcPwd` — пароль;

`gcDatabase` — имя базы данных.

Пример 3. Создание среды данных для ADO

Как уже говорилось в *главе 8*, можно ADO создавать интерактивным способом, но в этом случае месторасположение источника данных жестко "прописано" в коде. Поэтому чаще всего среда данных для ADO создается программно.

Пусть на сервере лежит база данных. Не важно, в каком формате (VFP, Access, SQL и т. п.) она реализована, главное, чтобы администратор сервера завел соответствующую запись и определил, какую команду нужно использовать для соединения

(Connection String). Обычно в команде указывается имя, заданное в настройках ODBC (`Data Source Name`, или `DSN`), а также идентификатор пользователя (`User ID`, или `UID`) и пароль для доступа к базе данных (`Password`, или `PWD`). Допустим, `Connection String` выглядит так: `DSN=fox; UID=sqlserver; PWD=sqlpassword`.

Эта строка указывает на некоторую базу данных, физическое расположение которой уже не имеет значения. В базе данных есть таблица **Customers**, имеющая много разных полей. Вставим в код следующий фрагмент:

```
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 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),"cust_id.txt", 1)
  STRTOFILE(TRANSFORM(oSet.fields(i-1).value), "cust_id.txt",1)
NEXT
rs.MoveNext
rs.Close
oConn.Close
```

Сессии данных

Есть еще один важный аспект, связанный с манипулированием данными, который нельзя игнорировать, — это организация сеансов работы с данными (Data Session). В Visual FoxPro по умолчанию создается так называемая сессия данных по умолчанию (Default Data Session), кроме того, на форме можно организовывать приватные (частные) сеансы работы с данными, каждый из которых может иметь до 32 767 рабочих областей. Установка частной сессии данных производится с помощью свойства формы `DataSession`: значение свойства для сессии данных по умолчанию равно 1, для частной сессии данных — 2. Открытие или закрытие таблиц в одной сессии данных никак не влияет на другую (рис. 9.5).

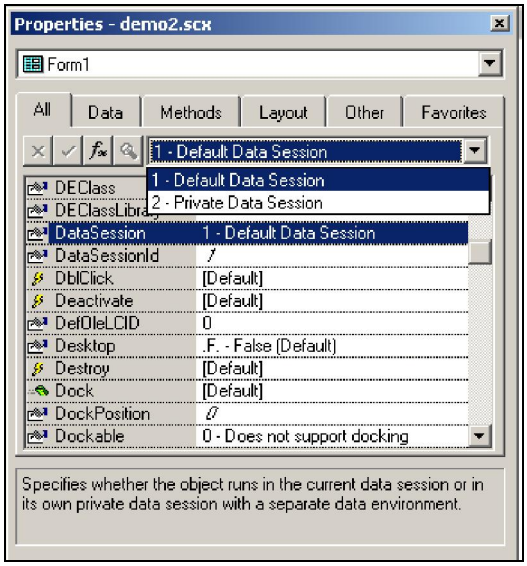

Рис. 9.5. Организация частной сессии данных на форме

Приватный сеанс работы с данными позволяет инкапсулировать все настройки, произведенные в окне **Data Environment**, для работы с конкретной формой.

Для чего могут понадобиться частные сессии данных? Например, при необходимости открыть два курсора с одинаковым именем. В этом случае без использования частной сессии данных курсор, созданный позднее, затирает более ранний.

Обратите внимание, что метод `AddObject()` контейнера и функция `CreateObject()` работают по-разному.

Метод `This.AddObject('MySession')`, добавляя экземпляр класса `Session` к контейнерам, имеющим свойство `DataSession`, никогда и никаких `DataSession` не создает, и действует так:

- если родительский объект имеет значение свойства `DataSession = 2-Private`, то экземпляр добавляется в `DataSession` родительского объекта;
- если родительский объект имеет значение свойства `DataSession = 1-Default`, то экземпляр добавляется в `DataSession Default(1)`.

В свою очередь, поведение при создании экземпляра объекта класса `Session` через `CreateObject('MySession')` несколько другое:

- если создаваемый объект имеет значение свойства `DataSession = 2-Private DS`, то всегда создается новая `DataSession`, причем значение свойства `MyForm.DataSessionID` переустанавливается в значение последней созданной сессии;
- если создаваемый объект имеет значение свойства `DataSession = 1-Default DS`, то в `DataSession Default(1)`.

Табличное представление данных

Основной задачей приложения для работы с базами данных является сохранение, удаление и редактирование данных. Основным средством работы с данными в Visual FoxPro является компонент класса `Grid`. `Grid` представляет данные в табличной форме, поскольку имеет строки и столбцы. Благодаря наглядности намного упрощается проведение любых расчетов, использующих данные, хранящиеся в таблицах.

Grid (Сетка)

`Grid`, несмотря на всю свою внешнюю простоту (рис. 9.6), является одним из самых сложных элементов управления формы. Сложен он с точки зрения понимания его работы.

Посмотрите, как много в нем составляющих! `Grid` содержит:

- строки и столбцы;
- полосы вертикальной и горизонтальной прокрутки;
- каждая колонка имеет заголовок и элемент управления для отображения и редактирования данных;
- указатель записи;
- пометка для удаленных записей.

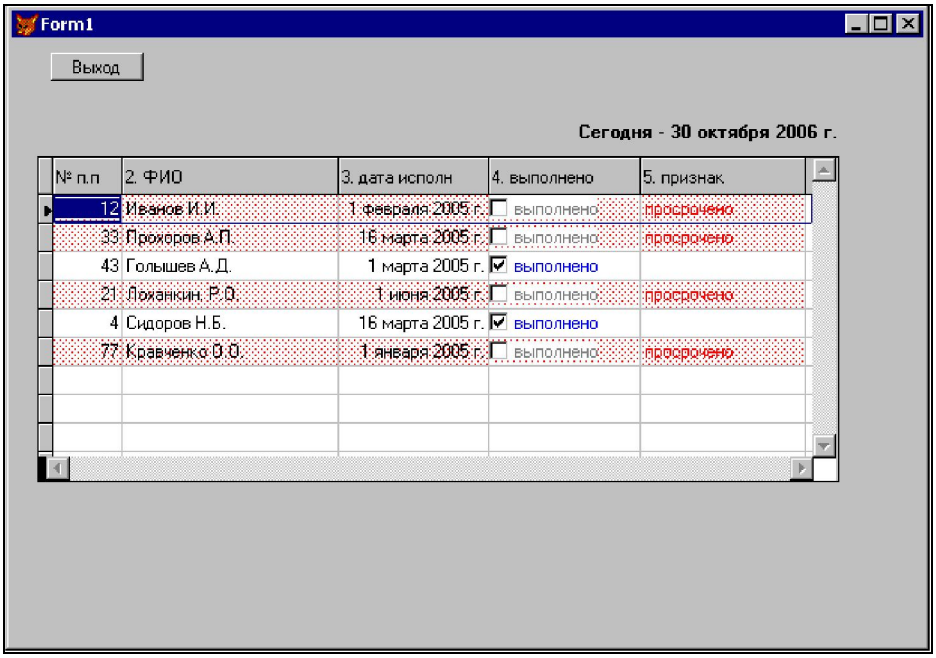

Рис. 9.6. Внешний вид `Grid`

`Grid` — это набор объектов, позволяющий представлять данные в виде таблицы-списка, который можно просматривать в двух направлениях при помощи полос прокрутки. `Grid` состоит из главного объекта `Grid` и набора колонок. Каждая колонка, в свою очередь, должна иметь объект заголовка и элемент управления, который позволяет редактировать и отображать данные в колонке `Grid`. По умолчанию таким объектом является `TextBox`, но возможно применение и других элементов управления: `CheckBox`, `Spinner`, `EditBox`, `OLEBoundControl` и др.

Внешне объект похож на часто применяемую в старых версиях FoxPro команду просмотра `Browse`, но в отличие от нее может быть включен в форму. Для создания объекта `Grid` интерактивным способом выполните следующие действия:

1. Создайте форму.
2. Выберите в **Form Controls** объект `Gird` (если **Form Controls** отсутствует, то выберите в главном меню **VFP View | Toolbars | Form Controls**), установите его в форму, нарисовав мышью прямоугольник нужного размера. По умолчанию свойство `ColumnCount` установлено в –1.
3. Нужно определиться, сколько и каких колонок вы хотите показать в `Grid`, и установить их число в свойстве `ColumnCount`. Для того чтобы определиться с колонками, придется заняться средой данных. На свободном от элементов управления месте формы нажимаем правую кнопку мыши, в появившемся контекстном меню выбираем **DataEnvironment** и подключаем все необходимые таблицы. Подсчи-

тываем количество колонок `Grid`, которые мы хотим показать в `Grid`, устанавливаем значение количества колонок таблицы в `ColumnCount`. Пусть в нашем примере их будет 5.

Среди объектов формы появляется не только объект `Grid`, но и 5 колонок, 5 заголовков колонок и 5 элементов управления, по умолчанию это `TextBox`.

4. В свойствах `Grid` находим `RecordSource` и прописываем в нем наш источник данных — имя таблицы или курсора, который мы хотим показать на экране.
5. Устанавливаем свойство `ControlSource` для каждого столбца. Это может быть конкретное поле курсора или таблицы, вычисляемое поле или поле, определяемое пользовательской функцией.
6. Определяем заголовки для каждого столбца, с этой целью изменяем в каждом заголовке (Header1...5) свойство `Caption`. Записываем те заголовки, которые считаем необходимыми. Заголовки можно сразу же отформатировать, используя свойство `Alignment`.

Для построения `Grid` можно использовать построитель — **Grid Builder**. Запустить **Grid Builder** можно одним из способов: выбрать объект `Grid`, а затем в его контекстном меню **Builder**; либо выбрать среди элементов `Form Controls` объект `Builder Lock`, а затем — `Grid`. В любом из случаев появится окно **Grid Builder**, имеющее четыре вкладки.

Таблица 9.1. *Вкладки* ***Grid Builder***

Вкладка	Назначение
Grid Items	Определяет выбор базы данных, таблиц и полей
Style	Определяет выбор стиля отображения таблицы
Layout	Определяет присвоение в поле редактирования **Caption** русских названий полям, а поле **Control type** — тип объекта, используемого в каждом столбце таблицы
Relationship	Устанавливает отношения между связанными таблицами

Объектная модель *Grid*

С точки зрения объектно-ориентированного приложения `Grid` представляет собой контейнер, который содержит вложенные объекты. К таким вложенным объектам относятся:

- колонка (`Column`);
- заголовок колонки (`Header`);
- элемент управления для отображения данных колонки (по умолчанию `TextBox`).

Соответственно, обратиться к свойствам объектов можно так, как это принято в объектно-ориентированном программировании:

<Объект>.<Свойство>=<Значение>,

например:

```
Grid1.Column1.Header1.Caption="Заголовок столбца"
```

Существуют свойства, определяющие сам объект `Grid`, и свойства, определяющие вложенные в него объекты (табл. 9.2).

Таблица 9.2. *Основные свойства объекта* `Grid`

Наименование	Описание
`ColumnCount`	Количество столбцов в объекте `Grid`
`DeleteMark`	Если значение `=.T.`, в левой части `Grid` появляется столбец для маркировки удаленных записей
`GridLineWidth`	Толщина сетки `Grid`
`GridLineColor`	Цвет сетки
`GridLines`	Стиль оформления сетки: 0 — отсутствуют горизонтальные и вертикальные разделительные линии 1 — только горизонтальные линии 2 — только вертикальные линии
`HeaderHight`	Определяет высоту заголовка
`Scrollbars`	Полосы прокрутки: 0 — нет полос прокрутки 1 — горизонтальная полоса 2 — вертикальная полоса 3 — горизонтальная и вертикальная полосы
`RecordSource`	Источник данных для размещаемой информации
`RecordSourceType`	Тип источника данных: 0 — таблица 1 — псевдоним 2 — запрашиваемое значение 3 — запрос 4 — SQL-выражение
`RecordMark`	Если значение свойства `=.T.`, то в левой части `Grid` появится указатель текущей записи

Любой объект Grid имеет составную часть в виде столбца (Column), хотя бы один. Этот столбец тоже является объектом и определяет информацию, выводимую в столбце (табл. 9.3). С помощью свойств столбца можно выделить наиболее важную информацию в Grid.

Таблица 9.3. *Свойства объекта* Column

Свойство	Описание
Alignment	Выравнивает информацию в столбце
Bound	Определяет, привязывается ли элемент управления в объекте Column к источнику объекта
ColumnOrder	Задает порядок следования столбцов
ControlSource	Связывает столбец с источником данных
Sparse	Определяет, затрагивает свойство CurrentControl все ячейки объекта Column или только активную ячейку
ToolTipText	Задает текст, который появляется в виде подсказки для указанного столбца

Каждый столбец имеет заголовок, который является объектом Header. Вы можете изменять вид заголовка, меняя его свойства (табл. 9.4).

Таблица 9.4. *Свойства объекта* Header

Свойство	Описание
Alignment	Выравнивает информацию в заголовке столбца
Caption	Определяет заголовок столбца
WordWrap	Определяет, по вертикали или по горизонтали расширяется заголовок, чтобы вместить текст, заданный его свойством Caption

Объект Text определяет вид информации, выводимой в ячейке сетки, когда курсор будет установлен на эту ячейку (табл. 9.5).

Таблица 9.5. *Свойства объекта* Text

Свойство	Описание
ControlSource	Задает источник данных
Format	Определяет формат ввода данных
InputMask	Определяет маску ввода
NullDisplay	Задает значение, которое будет выведено, если поле пусто

PasswordChar	Определяет, что отображается в поле — символы, введенные пользователем, или метки-заполнители; определяет символ, используемый в качестве метки-заполнителя

Методы и события объекта Grid отображены в табл. 9.6.

Таблица 9.6. *Некоторые методы и события объекта Grid*

Наименование	Описание
ActivateCell	Активизирует ячейку в элементе управления Grid
AddColumn	Добавляет объект Column в элемент управления Grid
AfterRowColChange	Имеет место, когда пользователь переходит в другую строку или другой столбец Grid, после того как новая ячейка получит фокус, а также после того как в новой строке или новом столбце произойдет событие When для данного объекта
BeforeRowColChange	Имеет место перед тем, как пользователь изменяет активную строку или столбец, прежде чем новая ячейка получит фокус
DeleteColumn	Удаляет объект Column из элемента управления Grid
Deleted	Имеет место, когда пользователь удаляет пометку записи на удаление или когда выдается команда DELETE
DoScroll	Прокручивает элемент управления Grid, моделируя работу пользователя с полосами прокрутки
Error	Имеет место при возникновении в методе ошибки этапа выполнения
KeyPress	Имеет место, когда пользователь нажимает и отпускает клавишу
Moved	Имеет место, когда объект перемещается на новую позицию или когда установки свойства Top или Left объекта-контейнера изменяются программным способом
RightClick	Имеет место, когда пользователь нажимает и затем отпускает правую кнопку мыши и при этом указатель мыши находится над Grid
SetAll	Присваивает установку некоторого свойства всем элементам управления данного объекта
SetFocus	Устанавливает фокус в Grid
Valid	Имеет место перед тем, как элемент управления теряет фокус
When	Имеет место перед тем, как элемент управления получает фокус

Следует заметить, что при установке свойства Grid AllowCellSelection=.T. (а оно устанавливается в истину по умолчанию) Visual FoxPro игнорирует событие KeyPress сетки и использует событие на уровне ячейки. Включив команду NODEFAULT в программный код события KeyPress, вы предотвратите или отмените ввод каких-либо символов в соответствующий буфер клавиатуры. Тем самым вы можете контролировать специально созданной процедурой для события KeyPress вводимые с клавиатуры символы и соответствующую обработку вводимых символов.

Знание последовательности выполняемых событий трудно переоценить, т. к. порой необъяснимые ошибки работы приложения бывают связаны с тем, что программный код в событиях выполняется не в той последовательности, в какой хотелось бы разработчику, а в той, какую задает порядок самих событий. Например, сначала инициализируется страница в объекте `PageFrame`, и только потом сам `PageFrame`. При этом общая последовательность событий подчиняется правилу "от частного к общему".

В приведенной ниже таблице (табл. 9.7) показана общая последовательность событий, происходящих на форме, имеющей `Grid` в качестве объекта, с учетом того, что свойство `AutoOpenTables` окружения данных установлено в `True (.T.)`.

Таблица 9.7. *Последовательность событий в форме, имеющей* `Grid`

Событие	**Описание**
Data environment	OpenTables
Data environment	BeforeOpenTables
Form	Load
Data environment cursors	Init
Data environment	Init
Header1	Init
Text1	Init
Column1	Init
<и т. д. в этом порядке все колонки Grid>	Init
Grid1	Init
Form	Init
Form	Show
Form	Activate
Grid1	When
Form	GotFocus
Grid1	GotFocus
Grid1	LostFocus
Form	QueryUnload
Form	Destroy
Form	Unload
Data environment	AfterCloseTables
Data environment	Destroy
Data environment cursors	Destroy

Последовательность событий легко получить самостоятельно с помощью отладчика.

Создание *Grid* программным способом

Представьте себе программу, выполняющую какие-либо сложные выборки данных или расчеты. Порой в таких программах до окончания вычислений невозможно определенно сказать, сколько колонок будет иметь итоговый курсор, какие будут заголовки у колонок, какие элементы управления нужно разместить в `Grid` для отображения полученных данных. Такими программами, к примеру, являются сложные системы подготовки отчетов, ведь формы отчетности часто изменяются.

В зависимости от даты подготовки отчета он может иметь разные колонки, заголовки и пр. Так что же — отказаться от применения `Grid`? Не стоит. В этом случае `Grid` может быть создан программно.

```
PUBLIC oForm
oForm=NEWOBJECT("Условия")
oForm.Show()

DEFINE CLASS Условия AS Form
    ADD OBJECT ogrid1 AS grid WITH RecordSource="MyCust",ColumnCount=1,;
    titlebar=0

        ADD OBJECT oButton1 AS CommandButton WITH TOP = 220,;
           Caption="Выход"

    PROCEDURE Load
      SELECT usl.naim FROM "data\usl" iNTO CURSOR MyCust

    PROCEDURE Init
        THISFORM.ogrid1.Column1.Width = 250
        THISFORM.ogrid1.Column1.FontSize=9

    PROCEDURE oButton1.Click
         ThisForm.Release()
ENDDEFINE
```

```
create cursor MyCursor (F1 C(3))
  PUBLIC MYFRM
  MYFRM=CREATEOBJECT('MYFORM')
  MYFRM.VISIBLE=.T.
  read events
```

```
DEFINE CLASS MYFORM AS FORM
PROCEDURE INIT
THISFORM.AddObject('MYOB','GRID')
THISFORM.MYOB.top=40
THISFORM.MYOB.VISIBLE=.T.
ENDPROC
ENDDEFINE
```

Программным способом также можно изменять объекты, входящие в состав `Grid` (`Header`, `Column`, `Text`).

```
Create cursor MyCursor (F1 C(3), f2 c(20))
Insert INTO  MyCursor(f1, f2)  VALUES ("001","Наименование")
Create CURSOR MyCursor1 (name c(20))
Insert INTO MyCursor1 (name) Values("Название1")
PUBLIC MYFRM
MYFRM=CREATEOBJECT('MYFORM')
MYFRM.VISIBLE=.T.
Read Events
DEFINE CLASS MYFORM AS FORM
PROCEDURE INIT
THISFORM.AddObject('GRID1','GRID')
THISFORM.Grid1.top=40
THISFORM.Grid1.VISIBLE=.T.
WITH thisform.Grid1
      .ColumnCount=2
      .Recordsource="MyCursor"
      .Scrollbars=2
      .Column1.header1.Caption="Поле1"
      .Column2.Header1.Caption="Поле2"
      .Column1.width=75
      .Column2.width=125
ENDWITH
WITH thisform.grid1.Column2
      .Removeobject('Text1')
      .AddObject('Combo1', "Combobox")
      .ControlSource='MyCursor.f2'
      .Sparse=.f.
      .Bound=.f.
      .Combo1.visible=.t.
endwith
WITH thisform.grid1.Column2.combo1
      .Boundto=.t.
      .Boundcolumn=2
      .ControlSource='MyCursor.F2'
```

```
        .RowSource="MyCursor1.Name"
        .RowSourceType=2
        .Style=2
  ENDWITH

  ENDPROC
  ENDDEFINE
```

В этом примере программно создается форма и объект `Grid` на ней. Свойство `Scrollbars` сетки, если оно установлено в значение 2, обеспечивает отображение только вертикальной полосы прокрутки объекта `Grid1`. Также программно устанавливается число столбцов (`ColumnCount`), их ширина (`Width`), названия заголовков (`Header.Caption`), во втором столбце удаляется элемент управления `TextBox` и устанавливается `ComboBox`. Для `ComboBox` назначается источник данных в виде поля курсора `MyCursor1.Name`.

Источники данных для *Grid*

Источниками данных для `Grid` могут быть:

0 — `Table` (Таблица)

1 — `Alias` (Псевдоним)

2 — `Prompt` (Запрашиваемое значение, подсказка)

3 — `Query` (.QPR) (Запрос)

4 — `SQL Statement` (SQL-выражение)

В качестве источника данных для `Grid` нельзя использовать массив. Источником данных для `Grid` может быть исключительно таблица (курсор и `View` — это временные таблицы). Однако можно, используя команду `CREATE CURSOR`, создать временную таблицу при помощи одной из команд:

```
APPEND FROM ARRAY ...
  или
INSERT INTO ... FROM ARRAY ...
```

Созданную таким образом таблицу или курсор можно использовать как источник данных для `Grid`.

Если вы используете в качестве источника данных для `Grid` SQL-выражение, не забывайте добавлять в конце выражения `Into Cursor` или `Into Table`, в противном случае вы увидите на экране окно **Browse** — по умолчанию именно в это окно выводится результат SQL-выражения.

Настройка и редактирование объекта *Grid*

Итак, мы создали `Grid`, и даже попробовали запустить форму на выполнение. Все замечательно, все работает. Но как только мы обнаружим, что один столбец лишний, появляется первый вопрос: как удалить этот столбец?

Чтобы удалить колонку из Grid, нужно:

- нажать левую кнопку мыши на заголовке ненужной колонки и, удерживая кнопку, перетащить ненужную колонку в конец Grid. Теперь в свойстве объекта уменьшить количество колонок (ColumnCount) Grid на 1. Последняя колонка исчезнет.
- Можно использовать и другой способ: нажать правую кнопку мыши, в появившемся меню выбрать **Edit**, затем нажать мышью на ненужную колонку (только не на заголовок, а на саму колонку!) и нажать на клавиатуре клавишу <Delete>. Лишняя колонка после подтверждения запроса будет удалена.

Если необходимо добавить колонку, проще всего это сделать увеличением значения свойства ColumnCount. Новая, добавленная, колонка станет последней колонкой Grid. Возможно, она не будет видна. Вы можете отредактировать Grid, если, установив на него курсор, нажмете правую кнопку мыши и в появившемся контекстном меню выберете пункт **Edit**. В этом случае станет возможно изменять ширину колонок, менять колонки местами. Для того чтобы изменить ширину колонки, выберите в контекстном меню пункт **Edit**, затем установите курсор на правую границу колонки (должен появиться крестик с двунаправленными стрелками), нажмите левую кнопку мыши и, не отпуская ее, двигайте границу вправо или влево, так, как вам необходимо. Чтобы поменять колонки местами, выберите в контекстном меню **Edit**, установите курсор на заголовок колонки, нажмите левую кнопку мыши и, не отпуская ее, перетащите колонку вправо или влево на нужное место.

Размещение в *Grid ComboBox*, *CheckBox* и других элементов управления

По умолчанию в колонках Grid для ввода и редактирования данных используется TextBox. Если вам необходимо разместить в одной из колонок другие элементы управления, например, ComboBox или CheckBox, выполните следующие действия:

1. Добавьте на форму объект Grid.
2. Установите значение его свойства ColumnCount больше нуля.
3. Активизируйте Grid, установив на него курсор мыши. Правой кнопкой мыши вызовите контекстное меню **Grid** и выберите пункт **Edit**. В старших версиях FoxPro в этот режим можно перейти, нажав и удерживая клавишу <Ctrl>, щелкнув по Grid левой кнопкой мыши.
4. В ToolBar с именем Form Controls щелкните левой кнопкой мыши по нужному объекту.
5. Левой кнопкой мыши щелкайте на нужном столбце в любом месте под заголовком.

Все, нужный объект вставлен. Убедиться в этом можно в окне **Properties**. В раскрывающемся списке в соответствующем столбце Grid кроме объекта Text1 появится и имя вставленного вами объекта. Чтобы именно вставленный объект отображался в

столбце, необходимо убедиться, что свойство `CurrentControl` этого столбца содержит имя вставленного объекта. А чтобы вставленный вами объект отображался не только в текущей строке `Grid`, но и во всех остальных строках, необходимо для того же столбца сделать дополнительную настройку

```
Sparse=.F.
```

Можно также использовать другой метод вставки:

1. Добавьте на форму объект `Grid`.
2. Установите его свойство `ColumnCount` больше нуля.
3. Добавьте экземпляр любого нужного вам класса, который вы хотите вставить в столбец `Grid` непосредственно на форму вне `Grid`.
4. Выделите добавленный объект (щелкнув по нему левой кнопкой мыши) и скопируйте его в буфер обмена (комбинация клавиш <Ctrl>+<C> или пункт системного меню **Edit**, подпункт **Copy**).
5. Правой кнопкой мыши щелкните на `Grid` и выберите пункт **Edit**. В старших версиях FoxPro в этот режим можно перейти, нажав и удерживая клавишу <Ctrl>, щелкнув по `Grid` левой кнопкой мыши.
6. Левой кнопкой мыши щелкните на нужном столбце в любом месте под заголовком.
7. Сделайте вставку из буфера обмена (комбинация клавиш <Ctrl>+<V> или пункт системного меню **Edit**, подпункт **Paste**).

Можно удалить уже не нужный объект `Text1` из столбца `Grid` следующим образом:

1. В окне **Properties** в раскрывающемся списке выберите объект, который хотите удалить.
2. Однократно щелкните левой кнопкой мыши по любому месту формы. Лучше щелкнуть по заголовку формы, хотя это не принципиально.
3. Нажмите клавишу <Delete> на клавиатуре.

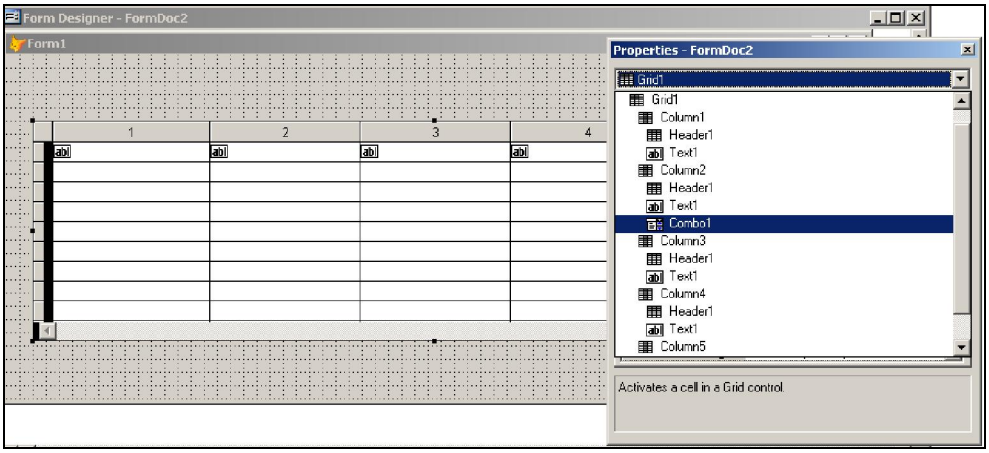

Рис. 9.7. Размещение `ComboBox` в колонке `Grid`

Иногда в объекте `Grid` нужно показать поле типа `Memo`. Обычно для такого типа поля используют класс `EditBox`. Проделаем только что приведенную процедуру, заменив текстовую коробку в колонке `Grid` на `EditBox` и увеличив высоту колонок.

Если запустить форму, то в колонке с `Memo`-полем будет видна всего одна строчка, хотя этот же класс показывает это же поле правильно, будучи помещен на форму.

Исправить ситуацию помогает небольшой трюк. Сделаем свой класс типа `Container`, поместив в него `EditBox`. Если теперь в колонку `Grid` поместить вместо `Editbox` наш класс, и заполнить свойство `Controlsource` колонки и `EditBox` внутри контейнера на имя поля типа `Memo`, в ячейке колонки будет видно столько строк, сколько позволит видеть высота колонки.

Визуальные эффекты *Grid*

Объект `Grid` имеет свои особенности, которые необходимо знать и учитывать при проектировании. Несмотря на все хлопоты, связанные с настройкой `Grid`, `Grid` — весьма полезный объект, и очень популярен среди программистов, поэтому существует много обходных путей и решений для `Grid`, которые позволяют пробиться сквозь ограничения и создавать действительно замечательные вещи. Ограничения, связанные со странностями поведения `Grid`, не должны быть ключевым аспектом в принятии решения об использовании его в приложении.

Изменение цвета активной строки *Grid*

По умолчанию в активной строке `Grid` (той, на которой установлен курсор) выделяется цветом одна активная ячейка. Как же быть, если нужно выделить цветом всю строку целиком? Нужно воспользоваться свойством `Grid HighLightStyle`

```
ThisForm.Grid.HighLightStyle = 2
```

Если полученный цвет вас не устраивает, то используйте свойства

```
ThisForm.Grid.HighLightBackColor — цвет фона
ThisForm.Grid.HighLightForeColor — цвет шрифта
```

В принципе, того же эффекта можно добиться, используя другое свойство

```
ThisForm.Grid.AllowCellSelection = .F.
```

Однако если свойство `HighLightStyle` отвечает только за способ (стиль) выделения цветом, то свойство `AllowCellSelection` отвечает также и за возможность выбора отдельной ячейки.

Замечание

Если какая-либо колонка `Grid` имеет свойство `Sparce = .F.`, то на нее не распространяется выделение цветом всей строки. В этом случае используется другой способ:

1. Необходимо создать дополнительное свойство формы `ThisForm.nRecno`. Можно сделать собственный класс `Grid` и создать это свойство у собственного класса, а затем присвоить этому свойству числовое значение, наприме, nRecno=1.

   Новое свойство можно добавить через пункт меню **Form** (для класса `Class`), подпункт **New Property** или **Edit Property/Method**.

2. В событии `Grid.Init` пишем следующий код:

```
IF RECNO(This.RecordSource)#0
      THIS.SetAll("DynamicBackColor",;
      "IIF(RECNO(This.RecordSource)=ThisForm.nRecno,RGB(255,255,0),RGB(
255,255,255))",;
"COLUMN")
ENDIF
```

3. Если свойство `nRecno` создано непосредственно в классе `Grid`, то, разумеется, можно вместо `ThisForm.nRecno` писать `This.nRecno`.

4. В событии `Grid.AfterRowColChange` запишем следующий код:

```
LPARAMETERS nColIndex
ThisForm.nRecno = RECNO(This.RecordSource)
This.Refresh()
```

Примечание

Если в качестве источника данных для `Grid` используется не псевдоним, т. е. `Grid.RecordSourceType<>1 — Alias`, то необходимо ввести еще одно свойство, которое будет содержать имя псевдонима, из которого следует считывать значение `Recno()`. Или же просто явно прописывать имя псевдонима непосредственно в коде.

Изменение цвета колонок Grid

Вы можете менять не только цвет отдельной строки или ячейки, но и колонки. У колонок `Grid` есть ряд динамических свойств. Нас будут интересовать свойства `DynamicBackColor` (это цвет фона ячейки) и `DynamicForeColor` (это цвет букв в ячейке). Если установить в этих свойствах условие раскраски, то при изменении значения ячейки автоматически изменится ее цвет и цвет шрифта. Например, имеется колонка, содержащая значение даты исполнения задания.

`ДатаИсполнения=` — значение, когда задание должно быть выполнено.

`Date()` — сегодняшняя дата.

Прописав в свойстве `DynamicBackColor` этой колонки `Grid` следующее условие:

```
IIF(ДатаИсполнения>Date(), RGB(255,0,0), IIF(Date()-ДатаИсполнения=1,
RGB(255,255,0), RGB(255,255,255))),
```

Получаем всего три варианта:

- `ДатаИсполнения > Date()` — выполнение задания просрочено, ячейка выделяется красным цветом;
- `Date() – ДатаИсполнения = 1` — сегодня последний день срока, ячейка будет желтой;
- `Date() < ДатаИсполнения` — в этом случае срабатывает RGB(255,255,255), до исполнения еще есть время, ячейка окрашена в белый цвет.

Если же таких ячеек много, то проще использовать `SetAll`.

Например, так: записываем в `Init()` формы следующий код:

```
ThisForm.Grid1.SetAll("DynamicBackColor","IIF(ДатаИсполнения>Date(),
RGB(255,0,0), IIF(Date()-ДатаИсполнения=1, RGB(255,255,0),
RGB(255,255,255)))","Column")
```

Пример раскраски вы можете посмотреть на компакт-диске в демонстрационном примере.

Блокировка (замораживание) колонки

Если надо заблокировать (заморозить) некоторую колонку или колонки `Grid`, чтобы они были всегда видны слева даже после горизонтальной прокрутки, первая мысль приходит — это разделить `Grid` на две части: левая показывает первые колонки, которые нам нужны, а правая используется для прокрутки. Но для пользователя это выглядит странно и требует объяснений: все эти лишние полосы прокрутки, слева часто больше, чем одна колонка и т. п.

`Grid` имеет хорошее свойство `LeftColumn`, содержащее порядковый номер колонки, которая сейчас отображена как самая левая колонка среди отображаемых колонок `Grid` при текущей горизонтальной прокрутке. Это значение изменяется при прокрутке. Можно его использовать, чтобы переместить нужную нам колонку на место левой

отображаемой позиции, в событии Scrolled Grid, заблокировав эту колонку таким образом:

```
if nDirection>3
this.Columns(1).ColumnOrder = this.LeftColumn
 endif
```

И для автоматической прокрутки по перемещению текущей клетки грида в событии AfterRowColChange добавьте строку:

```
this.Columns(1).ColumnOrder = this.LeftColumn
```

Теперь первая колонка Grid всегда отображается первой!

Автоматическая перепривязка колонок *Grid*

Если вы уже попробовали создавать формы с объектом Grid, возможно, вы уже знакомы с таким явлением, как автоматическая перепривязка колонок Grid. Оно заключается в том, что Grid отображает совсем не те колонки, которые вы ему задали в свойстве ControlSource. Данные в колонках располагаются в порядке их физического размещения в таблице, которая является источником данных для Grid, а вы задавали совсем иной порядок следования этих полей. Чем это чревато? Во-первых, если типы данных в колонке Grid и выводимом поле источника данных не совпадут (к примеру, описано как Logical, а выводится символьное поле), вы тут же получите сообщение об ошибке. Во-вторых, если при выводе поля вы использовали выражение, оно будет утеряно, и, например, вместо PADR(My_Memo,100) вы увидите на экране Memo. На экране могут появиться поля, о существовании которых пользователи и не подозревали, например, поле главного ключа таблицы. Неприятно? Конечно. Причиной такого странного поведения Grid является изменение свойства RecordSource во время разработки формы или на этапе выполнения. После такого изменения всегда очищается свойство ControlSource колонок, и Grid самостоятельно назначает поля колонкам. Если это не ошибка программы, и такое переопределение вам необходимо, придется позаботиться об устранении последствий. Как правило, значение ControlSource колонок сохраняется в свойстве Comment, а после изменения RecordSource восстанавливается обратно.

```
&& сохранить ControlSource каждой колонки
with Grid1
   local nColumnIndex
   for m.nColumnIndex = 1 to .ColumnCount
      .Columns(m.nColumnIndex).Comment = .Columns(m.nColumnIndex).ControlSource
   endfor
endwith
```

```
&& восстановить ControlSource каждой колонки
with Grid1
   local nColumnIndex
   for m.nColumnIndex = 1 to .ColumnCount
      if !empty(.Columns(m.nColumnIndex).Comment)
         .Columns(m.nColumnIndex).ControlSource =
.Columns(m.nColumnIndex).Comment
      endif
   endfor
endwith
```

Автоматическая перестройка *Grid*

Еще более неприятный эффект дает автоматическая перестройка `Grid`. Она заключается в том, что переоткрывается источник данных для `Grid`. Если источником данных была таблица, скорее всего, она была закрыта и открыта заново. Если источником данных было SQL-выражение с выбором данных в курсор, возможно, оно было повторно выполнено (с теми же или другими условиями). Также возможен вариант с выполнением команды `Pack`. Визуальный эффект этих действий довольно неприглядный: в колонках исчезают элементы управления, которые вы назначили для отображения информации в этих колонках. Вместо них появляется стандартный `TextBox`. Возможно появление на экране вместо `Grid` пустого прямоугольника. Как правило, автоматической перестройки `Grid` не случается, если источником данных является представление (View), а обновление данных происходит по команде `Requery()`. Если подобное переопределение все-таки необходимо, нужно перед переоткрытием источника данных сбросить его:

```
ThisForm.Grid1.RecordSource="Table1"
ThisForm. Grid1. RecordSource=""
<...> переоткрытие таблицы, выполнение необходимых действий
ThisForm.Grid1.RecordSource="Table1"
ThisForm.Grid1.Column1.ControlSource="Table1.Field1"
ThisForm.Grid1.Column2.ControlSource="Table1.Field2"
ThisForm.Grid1.Column3.ControlSource="Table1.Field3"
```

Заголовок колонки *Grid* в несколько строк

При практическом программировании встречаются такие заголовки колонок, которые не помещаются в одну короткую строчку, которая отведена в стандартном заголовке `Grid`. `Grid` позволяет увеличить высоту заголовка колонки и разместить заголовок в несколько строк. Для того чтобы разрешить размещение заголовка в несколько строк, установите свойство

```
Header.WordWrap=.T.
```

Однако при этом визуально заголовок останется таким же, каким был, потому что мала высота заголовка. Чтобы увеличить высоту заголовка, установите

```
Grid.HeaderHight
```

в нужное значение, чтобы поместился весь многострочный заголовок.

Вычисляемое поле в *Grid*

Что такое "вычисляемое поле"? Это поле, содержимое которого зависит (является функцией) от содержимого других столбцов. Например, таблица имеет два поля A и B, требуется показать в `Grid` значения полей A, B и сумму значений этих полей в третьем столбце. Вот этот третий столбец и будет вычисляемым полем `Grid`. Конечно, проще всего, наверное, формировать курсор с уже вычисленным третьим полем, и указать в качестве источника данных для `Grid` этот курсор, тогда не придется ломать голову. Но программисты не ищут легких путей (это, конечно, шутка, на самом деле такая необходимость бывает). Решений может быть несколько.

Если вычисляемое поле — это колонка `Grid`, то можно записать арифметическое выражение в `ControlSource` колонки:

```
ThisForm.Text1.ControlSource = "(tbl.fld1 + tbl.fld2)"
```

Для колонки `Grid` можно также применить `SELECT`:

```
SELECT table1.field3, (table1.field1 + table1.field2) as summa
```

> ***ПРИМЕЧАНИЕ***
>
> Если вычисляемое поле — это результат каких-либо арифметических действий в `TextBox`, то можно записать следующий код в событие `Refresh TextBox`:
>
> ```
> ThisForm.Text3.value=Thisform.text1.value+ThisForm.text2.value
> ```

Ниже приводится листинг примера, демонстрирующего создание вычисляемого столбца `Grid`. Вы можете запустить его с компакт-диска: Char9\calc_field.prg.

```
CREATE TABLE test (a n(2), b n(2))
  FOR i=1 TO 5
  INSERT INTO test (a,b) VALUES (i,i+i)
  ENDFOR
  GO TOP
  ox=CREATEOBJECT("form1")
  ox.visible=.T.
  READ events

  DEFINE CLASS form1 AS form
        DoCreate = .T.
        Caption = "Form1"
        Name = "FORM1"
```

```
        ADD OBJECT grid1 AS grid WITH ;
                ColumnCount = 3, ;
                Height = 197, ;
                Left = 60, ;
                RecordSource = "test", ;
                RecordSourceType = 1, ;
                Top = 24, ;
                Width = 329, ;
                Name = "Grid1"
          PROCEDURE grid1.init
                  This.column1.header1.Caption = "Столбец 1"
          This.column1.ControlSource = "a"
                  This.column2.header1.Caption = "Столбец 2"
          This.column2.ControlSource = "b"
                  This.Column3.ControlSource = "(a+b)"
                  This.Column3.Name = "Column3"
                  This.column3.header1.Caption = "Столбец3 — выч."
          ENDPROC
            PROCEDURE grid1.AfterRowColChange
                  LPARAMETERS nColIndex
                  ThisForm.Text1.Value=This.Column3.Text1.Value
                  ThisForm.Text1.Refresh()
        ENDPROC
        ADD OBJECT text1 AS textbox WITH ;
                ControlSource = "ThisForm.Grid1.Column3.Text1.Value", ;
                Height = 25, ;
                Left = 72, ;
                Top = 244, ;
                Width = 61, ;
                Name = "Text1"
      PROCEDURE unload
          CLEAR EVENTS
      ENDPROC
  ENDDEFINE
```

Изменение индекса при щелчке мыши на заголовке колонки *Grid*

В некоторых случаях весьма удобно изменять порядок просмотра данных в `Grid`, изменяя индекс щелчком мыши по заголовку колонки. В этом случае в `Header.Click()` нужно записать следующий код (листинг 9.7).

```
LOCAL lnRecNo
lnRecNo = IIF(!EOF(), RECNO(), 0)
ThisForm.DataEnvironment.cursor1.Order = "Column_Order"
```

```
WITH This.Parent.Parent
    .ActivateCell(1, .ActiveColumn)
    IF lnRecNo # 0
        GO (lnRecNo)
    ELSE
        GO TOP
    ENDIF
    .Refresh()
ENDWITH
```

OLE-объекты в таблицах

Объекты, которые можно размещать в таблицах Visual FoxPro, называются OLE-объектами, потому что они предполагают объектную компоновку и встраивание объектов (OLE — Object Linking and Embedding, дословно переводится как связывание и внедрение).

OLE-объектами являются графические изображения, фотографии, звуковые файлы, документы Word, таблицы Excel и другие, созданные с применением сторонних приложений. Для хранения таких объектов в таблицах предусмотрено поле `General`.

При компоновке объект хранится там, где он был создан, т. е. в файле, который был создан в приложении, а в таблице FoxPro содержится только ссылка на объект. Изменять объект можно только в том приложении, которое его создало. При перемещении или переименовании файла ссылка на объект теряется.

При вложении в таблице хранится копия объекта. Изменения, внесенные в объект в исходном приложении, не влияют на встроенный объект.

Внести данные в поле типа `General` программным путем можно, используя следующую команду:

```
APPEND GENERAL GeneralFieldName [FROM FileName]
   [DATA cExpression] [LINK] [CLASS OLEClassName
```

Опция `FROM` представляет собой имя файла-вложения, например myfoto.bmp.

Опция `DATA` включает символьное выражение, включаемое в OLE-объект. Эта опция неприменима к графическим объектам.

Если не используется опция `LINK`, то OLE-объект встраивается в поле типа `General`, в противном случае в поле типа `General` размещается ссылка на объект.

Опция `CLASS` определяет класс объекта, если класс не используется по умолчанию.

Пример:

```
USE sort
APPEND GENERAL foto FROM bmp\foto25.bmp
```

Удобно хранить в поле типа `General` фотографии изделий или рекламу. Программу можно написать так, что при передвижении по полям таблицы, хранящей, например, код и описание товара, будет меняться и фотография товара.

Информацию, находящуюся в поле `General`, можно просматривать программно, если использовать для просмотра команду

```
MODIFY GENERAL GeneralField1 [, GeneralField2 ...] [NOMODIFY]
   [NOWAIT] [[WINDOW WindowName1] [IN [WINDOW] WindowName2 | IN SCREEN]]
```

Чтобы просмотреть введенное изображение, нужно дважды щелкнуть на поле кнопкой мыши. А изменить его можно, вызвав то приложение, которое его создало, для этого нужно дважды щелкнуть кнопкой мыши, и править объект в уже появившемся приложении.

Объект можно встроить в поле `General` из другого приложения, воспользовавшись командами главного меню Visual FoxPro:

- **Edit | Paste**;
- **Edit | Paste Special**;
- **Edit | Insert Object**.

Команда **Edit | Paste** встраивает объект из буфера, куда его нужно поместить, либо выполнив **Cut** (Вырезать) и **Copy** (Скопировать), либо использовав <Ctrl>+<C> для копирования и <Ctrl>+<V> для ввода.

При использовании **Edit | Paste Special** открывается окно для выбора вставки (рис. 9.8).

Чаще всего для вставки OLE-объектов используется команда **Edit | Insert Object**. При выборе этой команды появляется следующее диалоговое окно (рис. 9.9).

Возможен выбор одного из следующих вариантов:

- создать новый.

  При этом тип объекта выбирается из представленного списка;

- создать из файла.

  При этом имя файла вводится в специальном окне (рис. 9.10).

Если вы хотите установить связь с объектом, установите флажок **Связь**. Если флажок снять, то объект становится неизменяемым.

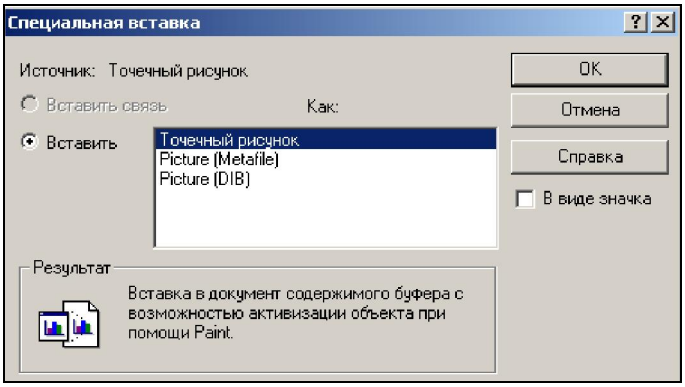

Рис. 9.8. Специальная вставка, выполняется команда **Edit | Paste Special**

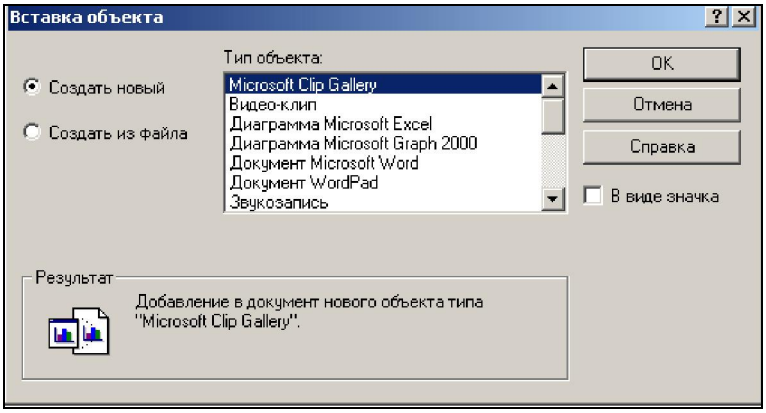

Рис. 9.9. Вставка объекта командой **Edit | Insert Object**

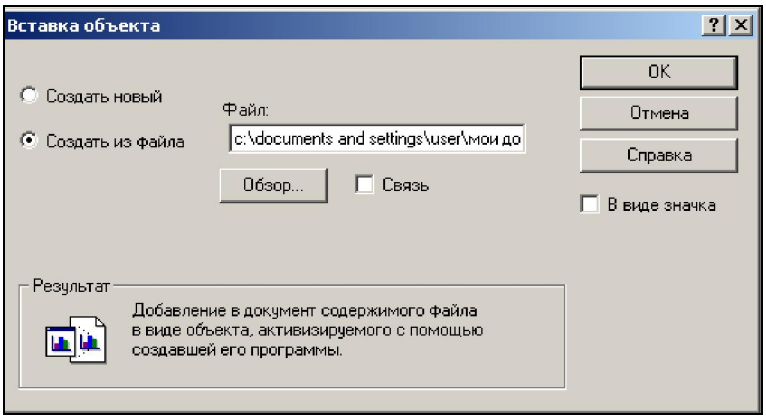

Рис. 9.10. Вставка из существующего файла

Для редактирования OLE-объекта нужно выполнить команду меню **Edit | Точечный рисунок BMP | Изменить**.

Для очистки открытого поля типа `General` используется команда меню **Edit | Clear**.

OLE-объекты можно также перетаскивать в поле типа `General`, для этого оба приложения должны находиться на экране одновременно. Объект нужно выделить и, не отпуская кнопку мыши, тащить его на поле `General` Visual FoxPro. Объект при этом будет встроен.

Кроме OLE-объектов, в поле типа `General` можно хранить и обычный текст.

Считывание изображений из полей *Blob* таблицы

Для считывания изображений из полей типа `Blob` можно применять свойство `PictureVal`. Для этого в событии `AfterRowColChange Grid` пишем:

```
ThisForm.Image1.PictureVal=MyTable.Myblobfield
```

Передвигаясь по полям `Grid`, наблюдаем изменение картинки. Свойство `PictureVal` не открывает файлы и не читает из них информацию, она берется из переменных и полей таблиц.

Небольшой пример (листинг 9.8) демонстрирует считывание изображения из поля типа `Blob` и размещение картинки в `Grid`. Он также находится на компакт-диске в Char9\toblob.prg.

```
Create Cursor TestPic (name  c(100), picture blob)
Insert into TestPic values ("file1",CAST(filetostr("book.bmp") as blob))
Insert into TestPic values ("file2",CAST(filetostr("shwamped.bmp") as blob))
 GO top
With CreateObject("MyForm")
      .Show(1)
EndWith

Define Class MyForm as Form
      Width=400
      Height=300

      Add Object Image1 as Image with ;
              top=10,;
              left=10,;
              Width=100,;
              Height=100

      Add Object Grid1 as Grid with ;
              top=10,;
              Left=260,;
```

```
            Width=100,;
            Height=100,;
            ColumnCount=1,;
            ReadOnly=.T.

    Procedure Grid1.AfterRowColChange(nIndex)
thisform.image1.PictureVal = testPic.Picture
EndDefine
```

Отображение данных в *Grid*

Таблицы и курсоры Visual FoxPro могут содержать следующие типы данных: Blob, Character, Character (Binary), Currency, Date, DateTime, Double, Float, General, Integer, Integer (Autoinc), Logical, Memo, Numeric, Varbinary, Varchar, Varchar (Binary).

Некоторые из указанных типов данных требуют дополнительных усилий для их корректного отображения в Grid. Например, если вы попытаетесь отобразить в Grid содержимое поля типа Memo, General или Blob, то увидите не содержимое этих полей, а лишь надписи Memo, Gen или Blob.

Для отображения Memo-поля в Grid можно применить элемент управления EditBox.

Замените TextBox на EditBox в колонке Grid, увеличьте высоту и ширину строки, чтобы удобно было читать содержимое Memo-поля. Для изменения высоты строки Grid воспользуйтесь свойством RowHeight. Изменить ширину колонки можно, изменяя свойство Width колонки. Не забудьте установить свойства Bound=.F. и Sparse=.F. для колонки, содержащей элемент управления. Если же по каким-то причинам вы не хотите использовать элементы управления в Grid, просто заключите содержимое ControlSource колонки в дополнительные круглые скобки, например, вот так:

```
ControlSource = (MyTable.MyMeno)
```

Чтобы посмотреть содержимое поля типа Blob, можно использовать EditBox или TextBox в соответствующем контейнере Grid, или при обычном использовании команды MODIFY MEMO, указав соответствующее имя поля Blob; однако содержимое этого поля находится в режиме "только-чтение". В EditBox данные типа Blob отображаются в соответствующей 16-ричной кодировке, без лидирующих символов "0h". В объекте Grid данные типа Blob отображаются как строка Blob, если значения отсутствуют, или в виде строки "Blob", если данные в этом поле имеются. При выполнении "double-click" на данном поле открывается дополнительное окно редактирования для отображения соответствующих данных в режиме "только-чтение".

Индексные ключи для типа данных Blob не поддерживаются. Кодовые страницы для типа Blob также не выполняют никаких преобразований.

Кроме того, для отображения графических изображений, хранящихся в полях типа Blob, можно использовать элемент управления Image.

Логический тип данных (Logical) удобнее всего отображать с помощью элемента `CheckBox`.

Для отображения числовых данных в нужном формате используйте свойство `InputMask`.

Навигация данных

Навигация — это принцип перемещения по указателям. `Grid` имеет указатель записи, перемещая его, мы передвигаемся по данным таблицы или курсора.

Для перемещения по `Grid` можно использовать полосы прокрутки: вертикальную для перемещения по записям, и горизонтальную — для перемещения по колонкам (если они не отменены разработчиком, разумеется). Отменить полосы прокрутки можно, установив в `Grid` свойство `Scrollbars=0`. Для перемещения по записям можно также использовать клавиши <PageUp>, <PageDown> и "стрелки вверх" и "вниз". Для перемещения по колонкам используйте клавиши <Tab>, <стрелки влево> и <вправо>. Перемещение в начало и конец `Grid` возможно при нажатии комбинации клавиш <Ctrl>+<Home>, <Ctrl>+<End>. Однако `Grid` не может автоматически устанавливать указатель записи в начало и конец данных при нажатии клавиш <Home> и <End>. Чтобы организовать такое перемещение, нужно в событии `KeyPress` формы поместить следующий код:

```
Do Case
  CASE LASTKEY()=1
Select Table
       GO top
       thisform.Grid1.Refresh()
   CASE LASTKEY()=6
  Select Table
        GO bott
        thisform.Grid1.Refresh()
Endcase
```

Если в событии `KeyPress` формы использована команда `NODEFAULT`, то будет отменено выполнение внутреннего программного кода Visual FoxPro, и будут обработаны только те клавиши, обработка которых включена в программный код события `KeyPress` формы.

Добавление, изменение и удаление записей в *Grid*

Удаление записи

Для удаления данных в `Grid` можно использовать пометку для удаленных записей, для этого можно нажать левую кнопку мыши на такой пометке (слева от записи). Однако разработчики чаще всего скрывают поле пометки, устанавливая

Grid.DeleteMark=.F., чтобы избежать ошибочного удаления. В этом случае в событии KeyPress формы можно поместить следующий код для обработки удаления:

```
DO CASE
  CASE LASTKEY()=7
    IF MESSAGEBOX("Удаляется запись со входящим номером  -";
       +CHR(13)+PADC(alltrIM(vxod.nvxod),40)+CHR(13)+" Удалить запись ?",
       4+32, " ") = 6
     SELECT vxod
     REPLACE vxod.avt WITH m.avt1 && записываем автора удаления и дату
     DELETE
     WAIT wind " Запись удалена ! " Nowait
       SET BELL on
       SET BELL TO "wav\deleted.wav"
       ?? CHR(7)
    thisform.Refresh()
ENDIF
ENDCASE
```

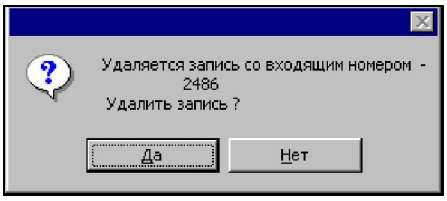

Рис. 9.11. Подтверждение удаления записи

Добавление записи

С точки зрения программиста, таблица не может иметь начала и конца, поэтому нельзя добавить данные в начало или конец таблицы, записи в нее добавляются по мере поступления, вне зависимости от их физического расположения в таблице. Кроме того, поскольку Grid является лишь отображением некой таблицы или курсора, то данные добавляются в таблицу, а не в Grid. При добавлении новой записи необходимо присваивать уникальный ключ записи. Возможные варианты решений:

- присваивать значение ключа в хранимой процедуре NewID;
- присваивать свой уникальный ключ каждой записи (можно использовать для хранения уникального ключа отдельную таблицу, в которой хранить последнее присвоенное или еще не присвоенное значение ключа, можно создавать ключ, используя функцию SYS(2015), и т. д.);
- использовать автоинкрементное поле.

Дальнейшие действия по добавлению записи описаны в *главе 7*. Это может быть использование как навигационных, так и реляционных команд.

Замечание

При добавлении записи в курсор возможно появление ошибки "`Cursor is read-only`". Это значит, что, создавая курсор командой `Select`, вы забыли добавить опцию `ReadWrite`. Если курсор создается командой `Create Cursor`, то в нем эта опция работает автоматически.

Изменение записей в *Grid*

Прежде всего, изменение записи в самом `Grid` считается среди программистов плохой идеей. Чаще всего для редактирования создается отдельная форма, в которой для выбранной записи имеются элементы управления для каждого редактируемого поля, и уже в этой отдельной форме происходит изменение данных. После окончания модификации данные сохраняются при нажатии кнопки **Сохранить**.

Команды, позволяющие изменить и сохранить модифицированную запись, также описаны в *главе 7*.