

**Автор: Horrific aka Фленов Михаил e-mail: [vr\\_online@cydsoft.com](mailto:vr_online@cydsoft.com)**

14.5 Поисковые поля.....	336
14.6 Улучшенный пример с поисковыми полями.....	344
14.7 Сортировка.....	346
14.8 Фильтрация данных .....	349
14.9 Язык запросов SQL .....	351

## 14.5 Поисковые поля

Настало время работать наше поле *Город* которое имеет числовой тип и никак не может пока хранить данные о городе. Для этого мы создадим отдельную таблицу в нашей базе данных с полями:

1. *Key1* – счётчик (ключевое поле);
2. *Название города* – текстовое поле размером в 30 символов.

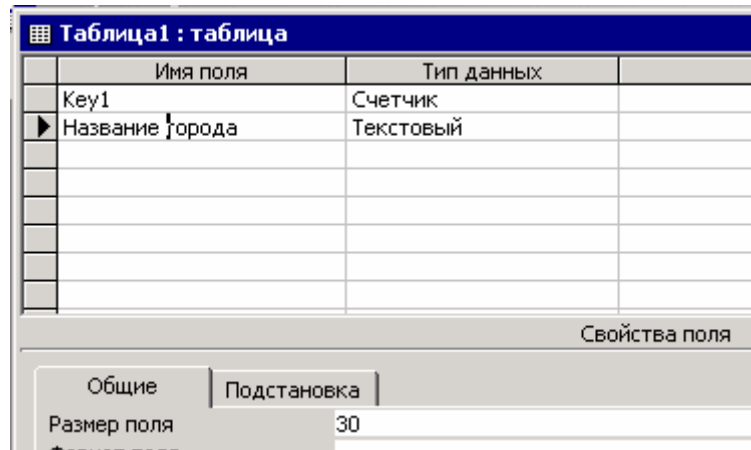


Рисунок 14.5.1 Таблица «Справочник городов»

Сохрани новую таблицу под именем «*Справочник городов*». Теперь твоя база данных должна состоять из двух таблиц:

1. Справочник;
2. Справочник городов.

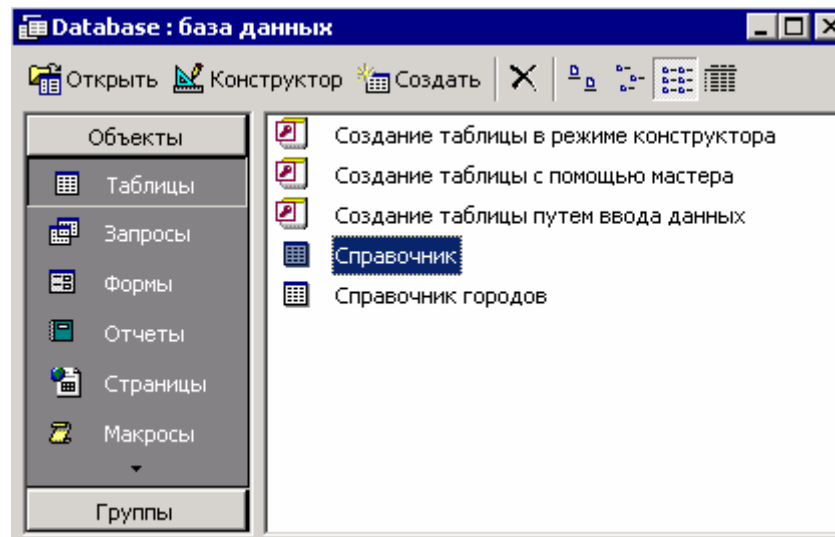


Рисунок 14.5.2 Обновлённый вид базы данных

Теперь давай откроем проект созданный в прошлой части главы и модуль *DataModuleUnit*. Добавь сюда компонент *DataSource* (назовём его *TownSource*) и *ADOTable* (его назовём *TownTable*). После этого у компонента *TownSource* в свойстве *DataSet* укажи таблицу *TownTable*.

Теперь давай настроим *TownTable* на отображение справочника городов. Для этого:

1. В свойстве *Connection* укажи компонент *ADOConnection1*, который указывает на нашу базу данных.
2. В свойстве *TableName* укажи таблицу *Справочник городов*.
3. Установи свойство *Active* в *True*, чтобы активизировать таблицу.

Войди в редактор полей таблицы *TownTable* и добавь все поля. Сделай поле *Key1* невидимым, потому что это счётчик и пользователю он абсолютно не нужен.

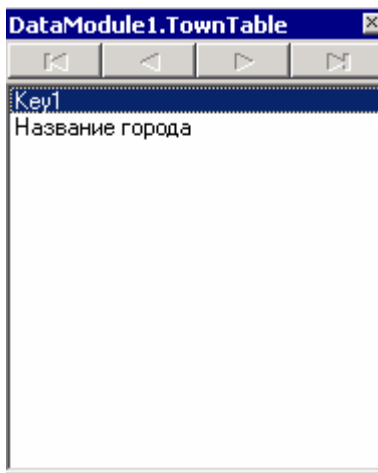


Рисунок 14.5.3 Редактор полей таблицы

Теперь создадим новую форму, для редактирования справочника и сохрани форму в модуле под именем *TownBookUnit*. Саму форму назовём *TownBookForm*. Подключи к новой форме модуль *DataModuleUnit*, чтобы отсюда можно было получить доступ к компонентам для работы с базами данных. Для этого из меню *File* выбери пункт *Use Unit* и в появившемся окне укажи модуль *DataModuleUnit* и нажми *OK*.

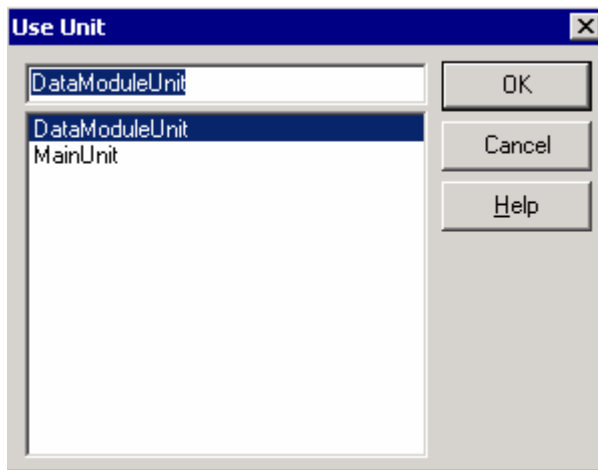


Рисунок 14.5.4 Добавление модуля *DataModuleUnit*.

Брось на форму сетку *DBGrid* и в свойстве *DataSource* укажи таблицу справочника городов - *DataModule1.TownSource*. Можешь всё это дело красиво оформить и добавить кнопку *OK*, для закрытия окна справочника. Моё окно редактора справочника городов ты можешь увидеть на рисунке 14.5.5.

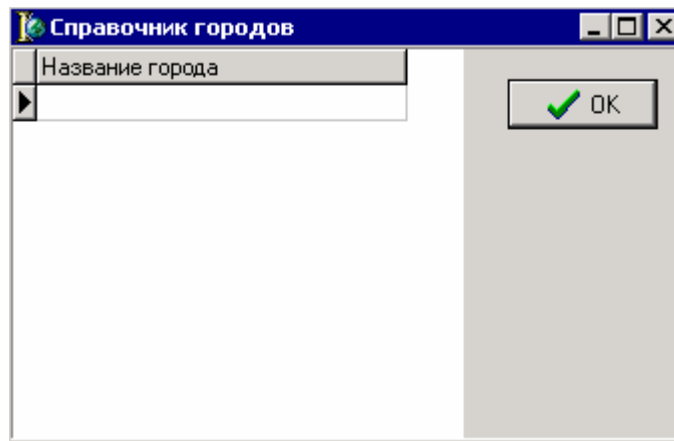


Рисунок 14.5.5 Окно справочника городов

Для большей красоты я ещё добавил на форму кнопки «Добавить», «Сохранить» и «Удалить» для добавления, удаления и сохранения строк справочника.

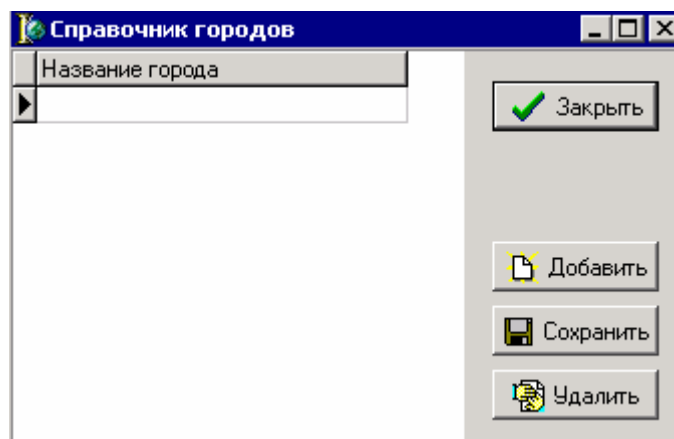


Рисунок 14.5.6 Обновлённая форма справочника

По нажатию кнопки «Добавить» пишем следующий код:

---

```
procedure TTownBookForm.AddBtnClick(Sender: TObject);
begin
    DataModule1.TownTable.Insert;
    DBGrid1.SetFocus;
end;
```

---

Метод *Insert* таблицы *TownTable* добавляет новую строку. Во второй строке я вызываю метод *SetFocus* нашей сетки, чтобы фокус ввода перешёл на него. После нажатия кнопки «Добавить» фокус попадает на неё, но после добавления новой строки, вполне логичным будет перенести фокус на сетку, потому что пользователь будет вводить имя города для новой строки.

По нажатию кнопки «Сохранить» пишем следующий код

---

```
procedure TTownBookForm.SaveBtnClick(Sender: TObject);
begin
```

---

```
if DataModule1.TownTable.Modified then  
    DataModule1.TownTable.Post;  
end;
```

---

Если текущая строка претерпела изменения, то в свойстве *Modifies* будет *true*, иначе *false*. А если произошли изменения, то их надо сохранить, иначе если пользователь закроет окно, то данные могут не сохраниться. Для сохранения изменений используется метод *Post*.

По нажатию кнопки «Удалить» пишем следующий код

---

```
procedure TTownBookForm.DeIBtnClick(Sender: TObject);  
begin  
    DataModule1.TownTable.Delete;  
end;
```

---

Метод *Delete* удаляет текущую строку из таблицы.

Всё, макияж для «Справочника городов» закончен. Теперь перейди к главной форме и создай меню или кнопку (я выбрал первое), для вызова справочника городов.

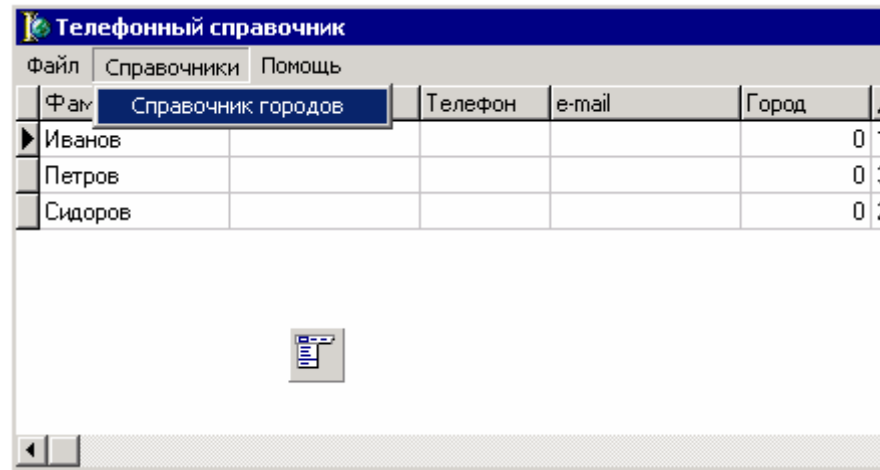


Рисунок 14.5.7 Меню вызова справочника городов

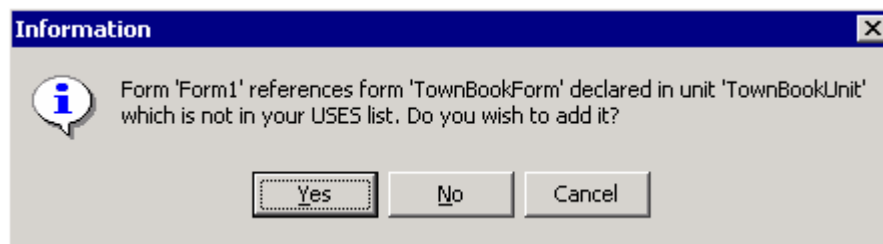
По событию *OnClick* от меню пишем код вызова окна справочника городов:

---

```
procedure TForm1.TownBookMenuItemClick(Sender: TObject);  
begin  
    TownBookForm.ShowModal;  
end;
```

---

Если ты не добавил модуль справочника городов к главной форме и попробуешь сейчас откомпилировать проект, то перед тобой появится следующий запрос:



Здесь говорится о том, что форма *TownBookForm* объявлена в модуле *TownBookUnit* и твоя форма не имеет ссылки на него. Тебе предлагается добавить её автоматически. Выбери *Yes* и модуль будет добавлен автоматически, после этого можно опять компилировать проект не внося никаких изменений. Теперь всё должно пройти удачно. Запусти проект и проверь работу программы.

Запусти программу, вызови «Справочник городов» и добавь туда несколько строк. Это будет полезно на будущее, заодно и проверишь правильность работы программы.

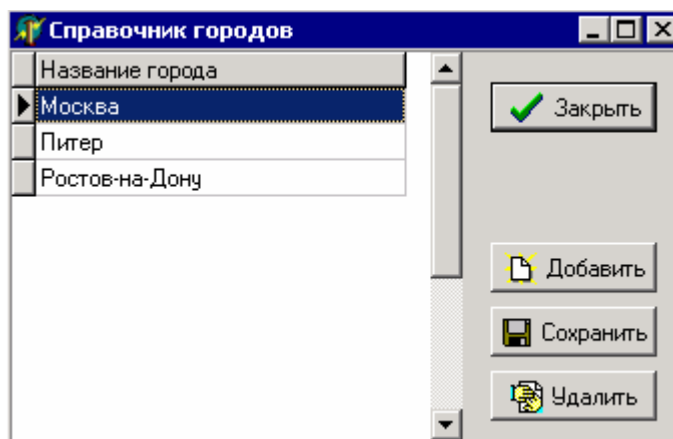


Рисунок 14.5.8 Справочник в действии

Теперь у нас есть справочник городов и мы можем подвязать его данные к основной таблице. Но перед этим немного улучшим форму. Выдели сетку *DBGrid1* на главной форме и в свойстве *Options* отключи возможность редактирования данных в сетке – в *dgEditing* присвой *false*:

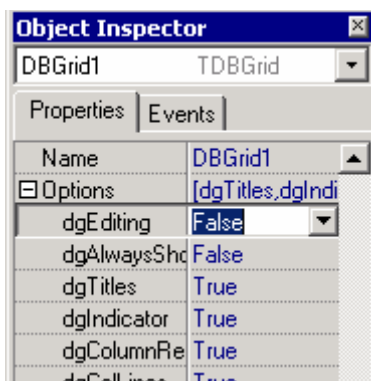


Рисунок 14.5.9 Отключение возможности редактирования

Теперь редактирование данных в сетке невозможно, поэтому мы сделаем для этого отдельные окна. В главном меню создай пункт «Редактирование» со следующими подпунктами:

1. Добавить запись;

2. Редактировать запись;
3. Удалить запись.

На рисунке 14.5.10 ты можешь увидеть результат этого меню. Лично я ещё создал панель с кнопками, чтобы к этим командам можно было быстро получить доступ и назначил командам клавиши быстрого вызова.

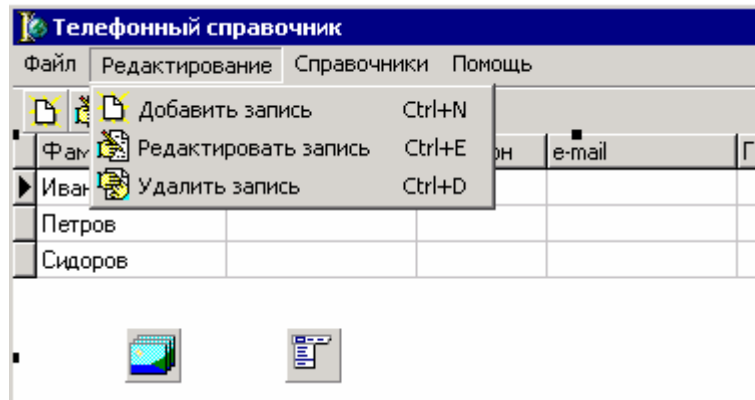


Рисунок 14.5.10 Меню «Редактирование»

Теперь создадим новую форму, которая будет использоваться для редактирования данных каждой записи. Создай форму и сохрани её под именем *EditFormUnit*. Саму же форму назовём *EditRecordForm*. Теперь измени у формы следующие свойства:

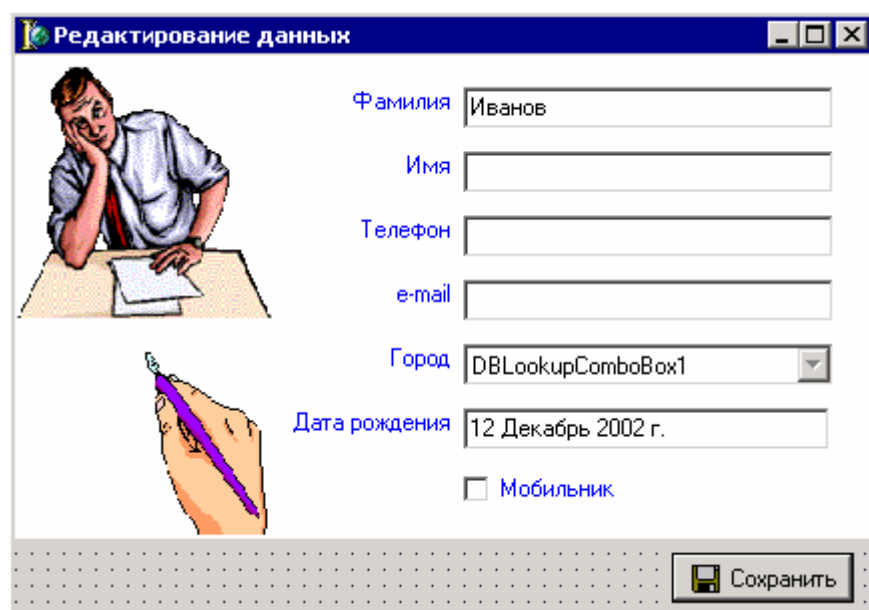
**BorderStyle** - *bsSingle*

**Position** – *poMainFormCenter*.

Ну и хватит. Этого достаточно, чтобы форма выглядела солидно.

Дальнейшее оформление зависит от твоих пристрастий, а я только покажу самое необходимое. Для начала подключи к новой форме модуль с данными, потому что нам необходимо будет иметь к ним доступ. Для этого выбери из меню *File* пункт *Use Unit*, в появившемся окне выбери *DataModuleUnit* и нажми *OK*.

Теперь посмотри на вид моей формы для редактирования данных (рисунок 14.5.11)



В этом окне у меня несколько компонентов для украшения вида (картинка, панель белого цвета), но это не главное. Ты можешь повторять и делать подобную форму, а можешь ограничиться только основными компонентами. Основными тут являются – кнопка сохранить, надписи и компоненты доступа к данным.

Напротив надписей *Фамилия*, *Имя*, *Телефон*, *e-mail* и *Дата рождения* находятся компоненты *DBEdit* с закладки *Data Controls*. Эти компоненты представляют собой простые строки ввода типа **TEdit**, только они умеют автоматически редактировать указанные поля в базе данных. Чтобы компонент видел данные из нужного поля нужно указать у него в свойстве *DataSource* нужную таблицу (*DataModule1.DataSource1*, как мы это делали с сеткой редактирования), а в свойстве *DataField* указать поле, которое надо редактировать. Обязательно попробуй сделать это сам, чтобы подробно разобраться с процессом установки полей.

Для свойства «*Мобильник*» лучше использовать компонент *DBCheckBox*. У него также надо указать поле в таблице, как и у компонентов *DBEdit*.

Самое интересное – поле «*Город*». Названия городов у нас хранятся в отдельном справочнике, а в основной таблице должны храниться только числа – номера строк из справочника городов. Допустим, что у выделенной записи нужно указать город «Москва», который идёт под номером 2 (поле *Key1* для этой строки в справочнике городов равно 2). В этом случае, в основном справочнике в поле «*Город*» нужно указать только цифру 2, а название города в любой момент можно найти в справочнике городов, по полю *Key1*. Так как оно уникально (счётчик), то проблем не возникнет.

Чтобы всё это реализовать достаточно поставить компонент *DBLookupComboBox* с закладки *Data Controls*. Теперь нужно указать у него в свойстве *DataSource* основную таблицу (*DataModule1.DataSource1*) которая будет редактироваться, а в свойстве *DataField* указать поле, которое надо редактировать – «*Город*».

Компонент *DBLookupComboBox* выглядит как выпадающий список (похож на **TComboBox**). В качестве элементов выпадающего списка можно указать таблицу. В свойстве *ListSource* нужно указать таблицу, из которой будут браться элементы для выпадающего списка. Давай укажем наш справочник городов - *DataModule1.TownSource*. В свойстве *ListField* укажем поле из этой таблицы, которое будет использоваться для заполнения выпадающего списка – «*Название города*». В свойстве *KeyField* нужно указать поле, значение которого будет вноситься в указанное поле основной таблицы - *Key1*.

Если что-то непонятно, то попробуй перечитать всё заново. Если не поможет, то подожди немного, скоро мы всё увидим на практике.

По нажатию кнопки «*Сохранить*» напиши:

---

```
procedure TEditRecordForm.BitBtn1Click(Sender: TObject);
begin
  if DataModule1.BookTable.Modified then
    DataModule1.BookTable.Post;
  Close;
end;
```

---

В первой строке я проверяю, если таблица была изменена (*DataModule1.BookTable.Modified* равна *true*), то принять изменения *DataModule1.BookTable.Post*.

Теперь перейди в основную форму и по нажатию пункта меню «*Добавить запись*» напиши следующее:

---



```
DataModule1.BookTable.Insert;  
EditRecordForm.ShowModal;
```

Здесь в первой строке я вставляю в основную таблицу новую строку. Во второй строке я отображаю окно редактирования данных.

По нажатию пункта меню «*Редактировать запись*» напиши просто код отображения окна редактирования - *EditRecordForm.ShowModal*;

Теперь запустим программу. Создай новую запись. На рисунке 14.5.12 ты можешь увидеть параметры, которые я забил.

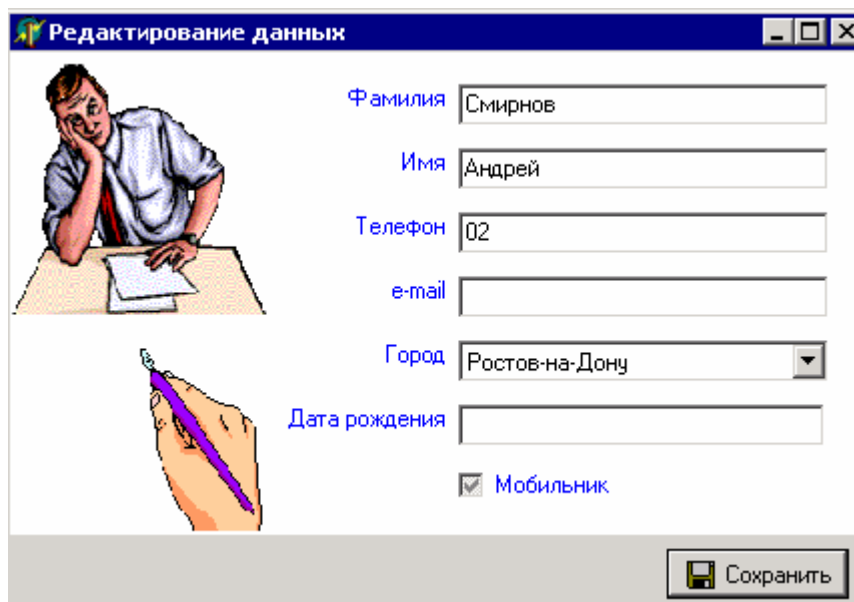
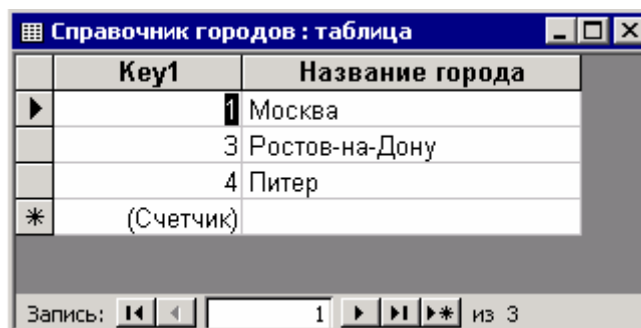


Рисунок 14.5.12 Окно редактирования данных

В поле город я выбрал «Ростов-на-Дону». После нажатия кнопки «*Сохранить*» окно закрывается. Посмотри на сетку. В поле «Город» новой строки ты можешь увидеть цифру 3. Это значит, что в справочнике городов есть запись со значением в поле Key1 равным 3 и в поле «Название города» указано название, которое мы выбрали. Давай откроем таблицу через Access и посмотрим:



Key1	Название города
1	Москва
3	Ростов-на-Дону
4	Питер
*	(Счетчик)

Рисунок 14.5.13 Таблица открытая с Access

Поле с названием «Ростов-на-Дону» действительно имеет в поле Key1 значение 3. Таким образом, в основном справочнике не надо указывать полный текст имени города.

Достаточно только указать нужный ключ справочника городов и мы в любой момент сможем найти название.

Что-то эта глава уже сильно раздулась и пора закругляться. В следующей части мы улучшим пример, а пока попробуй поиграть с уже созданным примером. Попробуй создать несколько строк и потом редактировать их. Обрати внимание, что когда ты открываешь окно редактирования, в компоненте *DBLookupComboBox* отображается правильное название города для указанной записи.

 На компакт диске, в директории \Примеры\Глава 14\Link ты можешь увидеть пример этой программы.

## 14.6 Улучшенный пример с поисковыми полями

Прежде чем двигаться дальше, давай сначала сделаем обработчик для меню «Удалить запись». По этому событию нужно вывести запрос на подтверждения удаления и если ответ утвердительный, то можно удалять. Создай такой обработчик и напиши в нём следующее:

---

```
begin
if Application.MessageBox(PChar('Ты действительно хочешь удалить '
+DataModule1.BookTableDSDesigner.AsString), 'Внимание!!!',
MB_OKCANCEL)=id_OK then
DataModule1.BookTable.Delete;
end;
```

---

Здесь я вывожу сообщение уже знакомой функцией *MessageBox*. В первом параметре (текст сообщения) я пишу текст 'Ты действительно хочешь удалить ' плюс значение поля «Фамилия» выделенной строки - *DataModule1.BookTableDSDesigner.AsString*. Самое сложное здесь - *DataModule1.BookTableDSDesigner.AsString*. Чтобы понять эту конструкцию перейди в модуль *DataModule*. Здесь щёлкни дважды по компоненту *BookTable*, где у нас подключён основной справочник, и затем по полю «Фамилия». Посмотри в объектном инспекторе имя этого поля, оно должно быть *BookTableDSDesigner*. Теперь ясно? Я пишу имя модуля данных (*DataModule1*), затем через точку имя поля (*BookTableDSDesigner*) и метод *AsString*, который возвращает значение поля в виде строки.

Можешь подняться в раздел *type* модуля *DataModule1* и убедиться, что внутри нашего объекта *TDataModule1* есть объявление свойства *BookTableDSDesigner* типа *TWideStringField*.

Надеюсь, что с первой строкой теперь всё ясно. Если нет, то запусти пример и посмотри, что произойдёт если попытаться удалить строку.

Во второй строке я просто удаляю текущую строку с помощью вызова метода *Delete* таблицы - *DataModule1.BookTable.Delete*.

Теперь пример практически готов. Единственный недостаток – в сетке просмотра данных вместо названия города отображается индекс строки в справочнике. Это очень неудобно, поэтому давай исправим этот недостаток.

Перейди в модуль *DataModule1*, и выдели компонент *BookTable*. Сделай его неактивным – в свойстве *Active* установи *false*. Теперь дважды щёлкни по этому компоненту и перед тобой откроется уже знакомый редактор полей. Давай создадим новое поле, которое будет содержать текстовое название города для строк таблицы. Для этого

щёлкини внутри окна редактора и в появившемся меню выбери пункт *New Field*. Перед тобой должно открыться окно, как на рисунке 14.6.1.

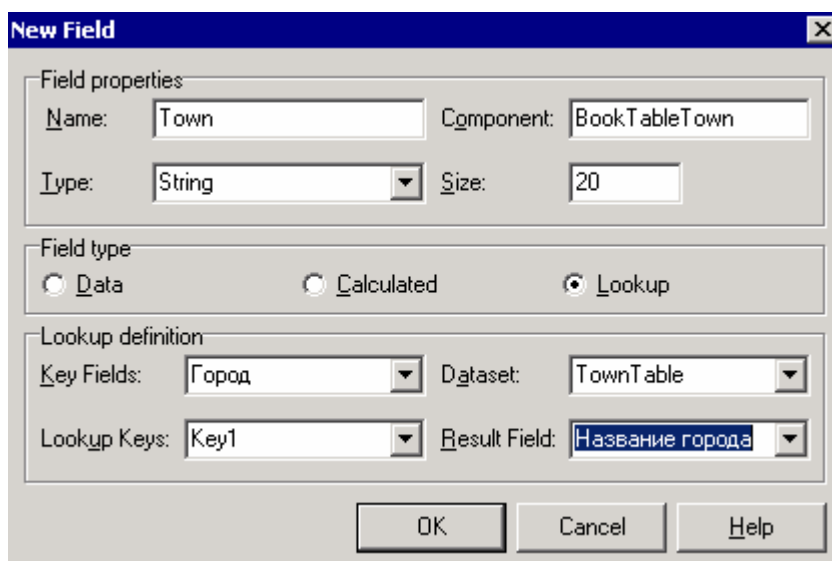


Рисунок 14.6.1 Окно создания нового поля



Создание нового поля возможно только при неактивной таблице, поэтому мы и выставили в свойстве *Active* значение *false*.

Заполни поля этого окна следующим образом:

В поле **Name** введи «*Town*»

В поле **Type** укажи тип *String* – строка.

В поле **FieldType** выбери *Lookup* – поисковое поле.

В поле **KeyField** (ключевое поле) выбери поле «Город». Это поле основной таблицы, по значению которого надо будет искать текст в другой таблице.

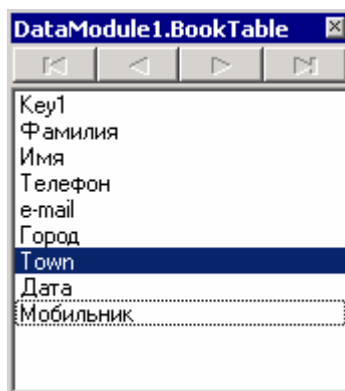
В поле **DataSet** надо указать *TownTable* – это таблица-справочник городов, где нужно искать.

В поле **Lookup Keys** укажи *Key1* – это поле в таблице справочнике, по которому надо искать.

В поле **Result Field** укажи поле «Название города» - это поле, текст которого будет подставляться.

Теперь нажми ОК.

В окне редакторе полей появиться новое поле с именем *Town*. В самой базе данных такого поля не будет, потому что оно динамическое и существует только в памяти машины, когда программа запущена. Перетащи его мышкой повыше, ближе к полю *Город*.



Снова сделай таблицу *BookTable* активной и попробуй теперь запустить программу. Посмотри на поле *Town* и ты увидишь теперь там тестовое название города (рисунок 14.6.2).

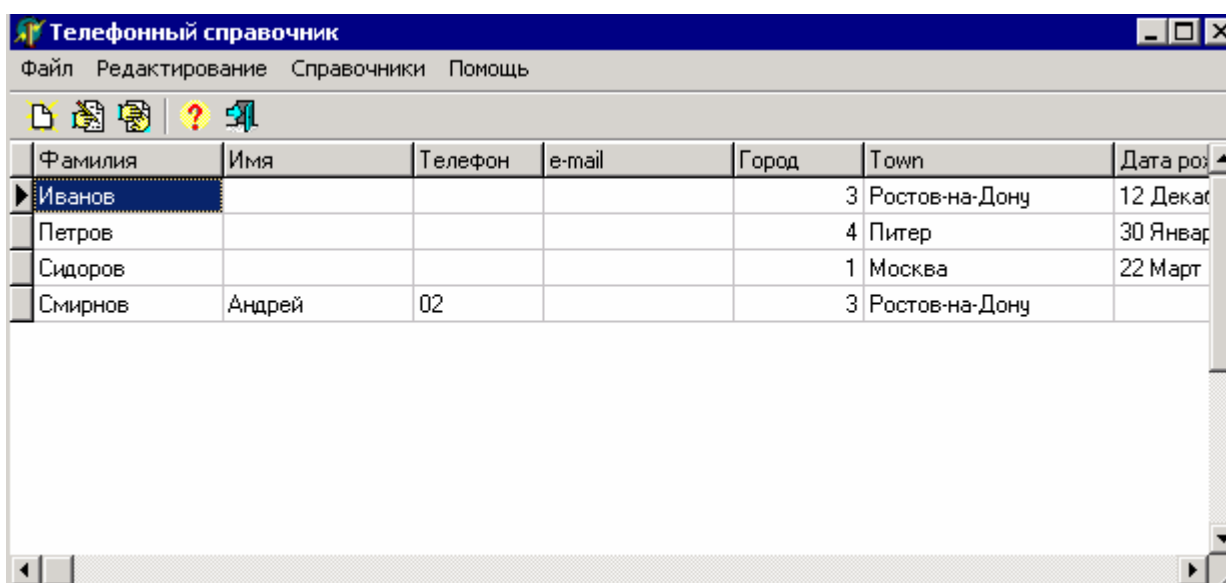


Рисунок 14.6.2 Результат работы программы

Теперь программа выглядит намного красивее и интереснее. Единственное – можно сделать поле «Город» невидимым, чтобы пользователь не видел эти непонятные числа, а над полем *Town* написать надпись «Город». Для этого дважды щёлкни по компоненту *BookTable*, выдели поле «Город» и установи в свойстве *Visible* значение *false*. Теперь выдели поле *Town* и в свойстве *DisplayLabel* напиши «Город».

Всё!!! С поисковыми полями покончено, можно двигаться дальше, глубже, шире :).

 На компакт диске, в директории [\Примеры\Глава 14\Link1](#) ты можешь увидеть пример этой программы.

## 14.7 Сортировка

Наш телефонный справочник уже достаточно хорош. В него можно добавлять новые записи, редактировать или удалять существующие. Но было бы удобней научить нашу программу сортировать записи по определённому

полю. Допустим, что тебе надо отсортировать данные по полю «Фамилия» - как это сделать? Очень просто, для этого существуют индексные поля.

В любой базе данных существует понятие индексного поля. Мы пока создавали только один индекс – главный для поля счётчика. Это обязательный индекс и существует всегда, но ты можешь создавать любое количество дополнительных индексов. Но это не значит, что надо все поля базы данных сделать индексными, ведь индексирование отнимает дополнительное место на диске и если переборщить, то можно наоборот сделать хуже. Поэтому нужно находить золотую середину.

Какие же поля индексировать? Я советую это делать только с теми полями, по которым будет чаще всего происходить поиск. В телефонном справочнике чаще ищут по номеру телефона или по фамилии. В домашнем справочнике это делают чаще по фамилии, а если у тебя будет справочник всего города, то возможно, что чаще будут искать по телефону или даже по адресу. Ну представь себе записную книжку с телефонами твоих друзей. По каким параметрам ты будешь искать номер своего друга? Ну конечно же по фамилии, ведь ты её знаешь, а номер телефона можешь забыть. Вот именно поэтому в нашем справочнике желательно сделать поле «Фамилия» индексным.

Индексы увеличивают скорость поиска данных и позволяют сортировать все записи. Если первое практически невозможно увидеть на примерах моей книги (потому что количество записей у нас очень маленькое, а для ощущения скорости нужно большое количество данных), то сортировку можно ощутить без проблем.

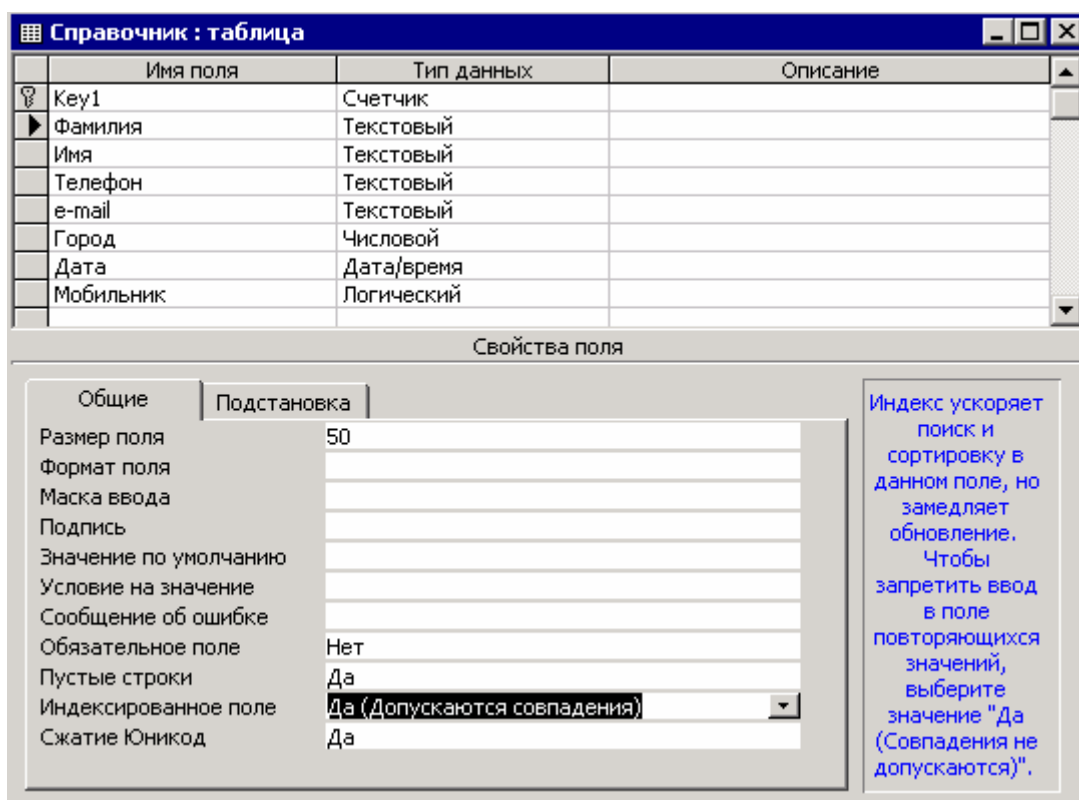


Рисунок 14.7.1 Редактирование свойств полей

Открой нашу базу данных в Access, выдели таблицу «Справочник» и нажми кнопку «Конструктор». Перед тобой откроется окно редактирование свойств полей (рисунок 14.7.1). Здесь выдели поле «Фамилия» и посмотри на свойство «Индексированное поле». Свойство имеет выпадающий список для выбора нужного параметра. Тебе доступны три варианта:

1. *Нет* – поле не индексировано.

2. *Да (Допускаются совпадения)* – поле индексировано и две (или более) записи могут иметь одно и то же значение. Это значит, что у тебя может быть две записи, где в поле «*Фамилия*» указано значение «*Сидоров*». Для нашего случая это идеальный вариант, потому что много людей могут быть под одной и той же фамилией.

3. *Да (Совпадения не допускаются)* – если выбран этот параметр, то в проиндексированном поле нельзя хранить одинаковое значение в разных записей. База данных будет сама следить за уникальностью поля. Если выбрать этот параметр, то две записи в нашем справочнике не смогут иметь значение «*Сидоров*». Этот параметр очень удобен, когда тебе нужно, чтобы поле было действительно уникальным. Такое бывает в офисных приложениях, например, номер документа часто должен быть уникальным.

Давай сделаем два поля индексными: «*Фамилия*» и «*Телефон*». У обоих нужно выбрать в свойстве «*Индексированное поле*» параметр «*Да (Допускаются совпадения)*».

Всё, закрываем базу данных и переходим к программированию. Загружай в Delphi пример, который мы написали в прошлой части, чтобы мы могли добавить к нашему справочнику возможность сортировки.

Для начала улучшим нашу форму. Для этого добавь в меню нашей программы пункт «*Сортировка*» и два его подпункта «*По фамилии*» и «*По телефону*» (рисунок 14.7.2).

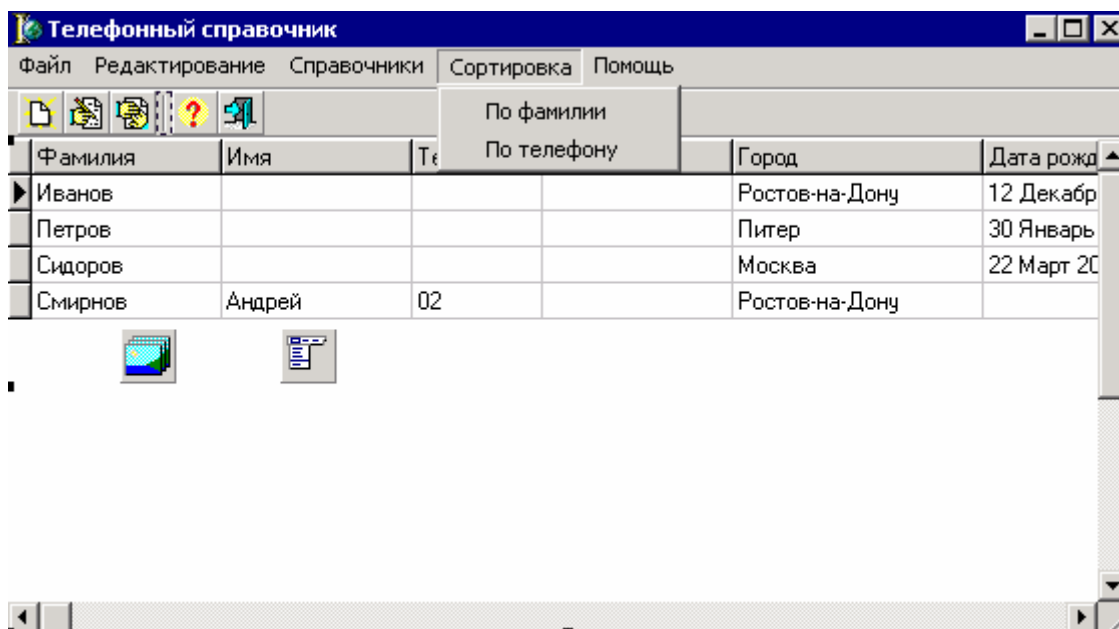


Рисунок 14.7.2 Улучшенная форма нашей программы

По нажатию пункта меню «*По фамилии*» напиши следующий код:

```
procedure TForm1.N8Click(Sender: TObject);
begin
  DataModule1.BookTable.IndexFieldNames:= 'Фамилия';
end;
```

По нажатию пункта меню «*По телефону*» напиши следующий код:

```
procedure TForm1.N8Click(Sender: TObject);
begin
```

```
DataModule1.BookTable.IndexFieldNames:= 'Телефон';  
end;
```

---

В обоих случаях я присваиваю свойству *IndexFieldNames* таблицы *BookTable* значение поля, по которому нужно сортировать записи.

 На компакт диске, в директории \Примеры\Глава 14\Index ты можешь увидеть пример этой программы.

## 14.8 Фильтрация данных

Наш телефонный справочник достаточно быстро набирается новыми возможностями, но в нём до сих пор нет самого главного – поиска. Представь себе, что у тебя в базе данных находятся телефоны ста человек. Как ты будешь искать телефон определённого человека? А если в базе данных будет 1000 записей? Без возможности поиска тут очень тяжело.

Для поиска в компоненте TADOTable есть свойство *Filter*. В нём можно указывать условие, по которому будут отображаться данные. Например, ты можешь указать там отображение только записей, в которых поле «*Фамилия*» содержит значение «*Сидоров*». Но для того, чтобы фильтр заработал, надо ещё установить свойство *Filtered* нашей таблицы в *true*. После этого можно изменять свойство *Filter* и все изменения сразу же будут вступать в силу.

Свойство *Filter* – это строка. В ней нужно писать текст условия в виде:

**Поле [Оператор сравнения] ‘Значение’**

Например, если ты хочешь отобразить все записи, в которых поле «*Фамилия*» равно значению «*Сидоров*», то нужно написать:

```
AdoTable1.Filter:='Фамилия='Сидоров';
```

Обрати внимание, что значение нужно указывать в одинарных кавычках. Но так как одинарные кавычки используются в Delphi для ограничения строк, то тут приходится немного включить соображение. Чтобы внутри строки поставить одинарную кавычку, её нужно поставить дважды:

**‘После этого текста будет одинарная кавычка’ это продолжение текста’**

Именно таким способом я ставлю перед значением одинарную кавычку. После значения мне нужно поставить одинарную кавычку и закрыть строку, поэтому я ставлю три одинарных кавычки (две для того, чтобы поставить кавычку для значения и одна для конца строки).

Это был пример простейшего условия на равенство. Ты можешь использовать любые другие операторы сравнения (больше или меньше). А можно даже создавать составные операторы сравнения, в которых сравнивается сразу два или более значений. Например:

```
AdoTable1.Filter:='Фамилия='Сидоров' or Телефон='3326523';
```



В этом пример я ищу все записи, в которых поле «Фамилия» равно значению «Сидоров» и поле «Телефон» имеет значение «3326523». Для объединения двух условий используется оператор **or**. Можно также использовать оператор логического «и», т.е. **and**.



*При программировании фильтров будь внимателен к кавычкам. Помни, что значения должны выделяться одинарными кавычками и внутри строки для этого приходится ставить две одинарных кавычки, а не одну двойную!!!*

Теперь переходим к программированию. Открывай пример из предыдущей части и будем добавлять к нему возможность поиска.

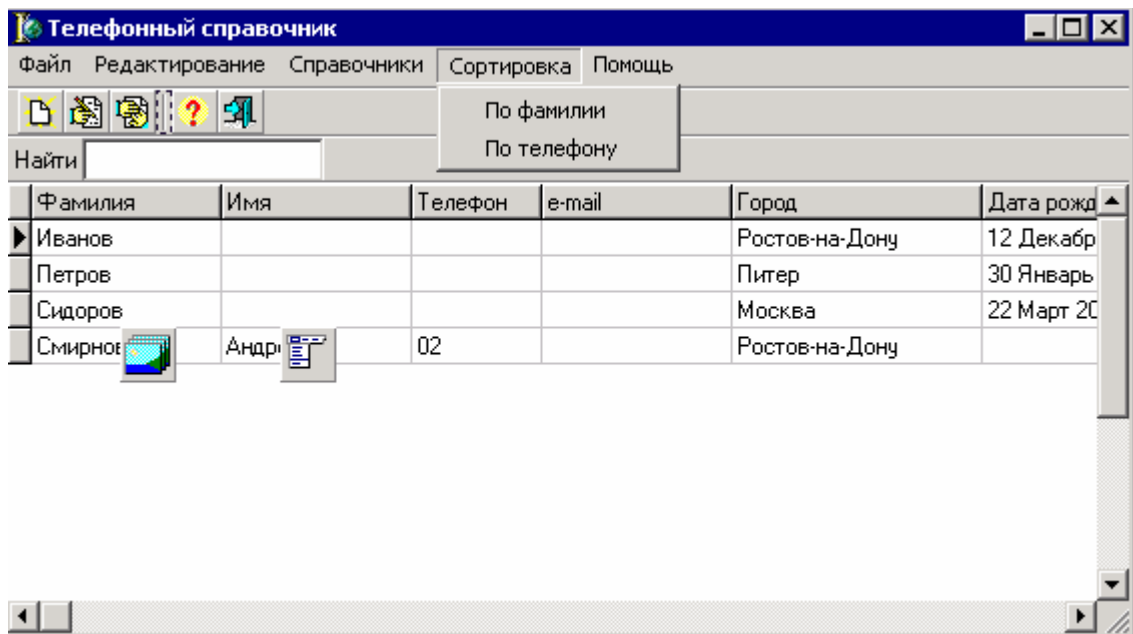


Рисунок 14.8.1 Улучшенная форма нашей программы

Для начала улучшим нашу форму. Для этого добавь панельку, на которой будет располагаться текст «Найти» и строка ввода **TEdit** с именем *FindEdit* (рисунок 14.8.1). Теперь создай обработчик события *OnChange* для строки ввода. Когда пользователь изменил текст в строке ввода, мы должны изменить и фильтр. Напиши в этом обработчике следующий код:

```
procedure TForm1.FindEditChange(Sender: TObject);
begin
  if Length(FindEdit.Text)>0 then
    DataModule1.BookTable.Filtered:=true
  else
    DataModule1.BookTable.Filtered:=false;

  DataModule1.BookTable.Filter:='Фамилия>'+'FindEdit.Text+''';
end;
```



Вначале я проверяю, если в строке поиска что-то есть, то включаю фильтр иначе его можно отключить, чтобы показать всю таблицу. После этого создаю условие фильтра: 'Фамилия>' + FindEdit.Text + ''. Я здесь использую знак больше, чтобы отображать все похожие записи на введенный текст. Если установить знак равенства и пользователь введёт букву «с», то в таблице ничего отображаться не будет, потому что нет такой фамилии «с». А при знаке «*больше*» будут отображаться все фамилии начинающиеся на букву «с».

В конце фильтра я добавляю четыре одинарных кавычки. Почему четыре? Да потому что после значения нам надо добавить одну кавычку. Чтобы её добавить надо добавить строку, содержащую кавычку. Поэтому у меня стоит две одинарных кавычки чтобы открыть и закрыть эту строку. Внутри строки я ставлю эту кавычку, а как ты помнишь, для этого надо ставить две кавычки и в результате получится одна. Вот так и получается 4 одинарных кавычки.

Вот так мы получили простую возможность поиска. Более эффективные возможности можно получить используя язык SQL – это язык запросов к базам данных. Но это отдельная история и требует отдельного рассмотрения.

 На компакт диске, в директории \Примеры\Глава 14\Filter ты можешь увидеть пример этой программы.

## 14.9 Язык запросов SQL

**S**QL переводят на русский как Структурированный Язык Запросов. С помощью SQL-запросов можно создавать и работать с реляционными базами данных. Этот язык стал стандартом, поэтому если ты хочешь работать с базами данных, то ты должен знать этот язык как каждую дырку в своих зубах.

SQL определяется Американским Национальным Институтом Стандартов и Международной Организацией по стандартизации (ISO). Несмотря на это, некоторые производители баз данных вносят изменения и дополнения в этот язык. Эти изменения незначительны и основа остаётся совместимой со стандартом.

Что такое реляционная база данных? Это таблица, в которой в качестве столбцов выступают поля данных, а каждая строка хранит данные. В каждой таблице должно быть одно уникальное поле, которое однозначно будет идентифицировать строку. Это поле называется ключевым. Эти поля очень часто используются для связывания таблиц или для обеспечения уникальности каждой записи. Но даже если у тебя таблица не связана, ключевое поле всё равно обязательно. Представь, что ты пишешь телефонную базу данных. Сколько у тебя будет "Ивановых"? Как ты будешь отличать их? Вот тут тебе поможет ключ. В качестве ключа желательно использовать численный тип и если позволяет база данных, то будет лучше, если он будет типа "autoincrement" (автоматически увеличивающееся/уменьшающееся число).

Столбцы в базе данных, также должны быть уникальными, но в этом случае не обязательно числовыми. Их можно называть как угодно, лишь бы было уникально и тебе понятно, а остальное никого не касается.

SQL может быть двух типов: интерактивный и вложенный. Первый - это отдельный язык, он сам выполняет запросы и сразу показывает результат работы. Второй - это когда SQL язык вложен в другой, как например в C++ или Delphi.

Интерактивный SQL более близок к стандартному, а во вложенном очень часто встречаются отклонения и дополнения. Например, в стандартном SQL различаются только два типа данных: строки и числа, но некоторые производители добавляют свои

типы (Date, Time, Binary и т.д.). Числа в SQL делятся на два типа: целые (INTEGER или INT) и дробные (DECIMAL или DEC). Строки ограничены размером в 254 символа.

Более подробную информацию о SQL ты найдешь в документе «Язык запросов *SQL.doc*» на диске к книге в директории «Документация». В этом документе ты найдешь достаточно подробное описание языка SQL, поэтому прежде чем двигаться дальше советую прочитать этот документ. Особенно если ты хочешь в дальнейшем профессионально писать программы для работы с базами данных.

Давай посмотрим, как можно направить базе данных простейший SQL запрос. В качестве примера я реализую возможность поиска записей по номеру телефона в нашем телефонном справочнике.

Для отправки запросов базе данных используется компонент **TADOQuery** с закладки ADO палитры компонентов. Работа этого компонента схожа с таблицей **TADOTable** и у них даже много схожих полей. В **TADOQuery** ты также должен выбирать строку подключения (свойство *ConnectionString*) или связываться с компонентом **TADOConnection** через свойство *Connection*. У запросов даже есть возможность установки фильтра (свойства *Filtered* и *Filter*). Но мы его рассматривать не будем для экономии места, потому что работа с этим фильтром ничем не отличается от фильтра таблиц **TADOTable** который мы рассматривали в главе 14.8.

Давай откроем наш телефонный справочник и дополним его новыми возможностями.

Открой модуль данный *DataModule*, где у нас расположены все компоненты доступа к базе данных. Добавь сюда компонента **TADOQuery** (назовём его *FindQuery*) и компонент **TDataSource** (назовём его *FindSource*). Теперь надо связать эти компоненты, указав у компонента *FindSource* в свойстве *DataSet* компонент *FindQuery*.

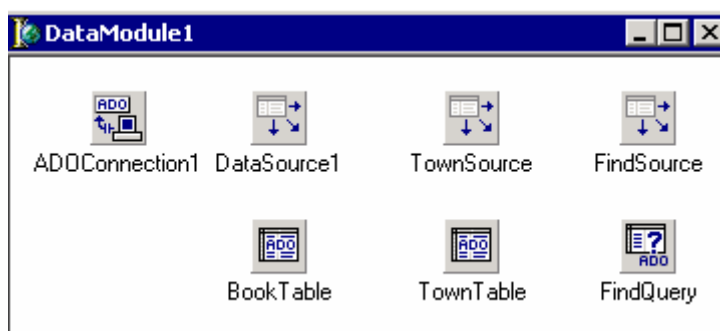


Рисунок 14.9.1 Модуль *DataModule*

Напоминаю, что **TDataSource** отвечает за отображение данных из таблиц. Компонент **TADOQuery** предназначен для отправки SQL запросов базе данных. Результат запросов возвращается в виде таблиц и для отображения результата нам будет необходим компонент **TDataSource**. Именно поэтому мы их установили на форму и связали между собой, чтобы компонент отображения видел данные, которые надо отображать.

Теперь выдели компонент *FindQuery*, здесь нам необходимо указать в свойстве *Connection* наш компонент подключения к базе данных *ADOConnection1*. Этим мы укажем компоненту, какой базе будут отправляться запросы.

Теперь напишем сам запрос. Для этого дважды щёлкни по свойству *SQL* и перед тобой откроется окно редактора запросов (рисунок 14.9.2).

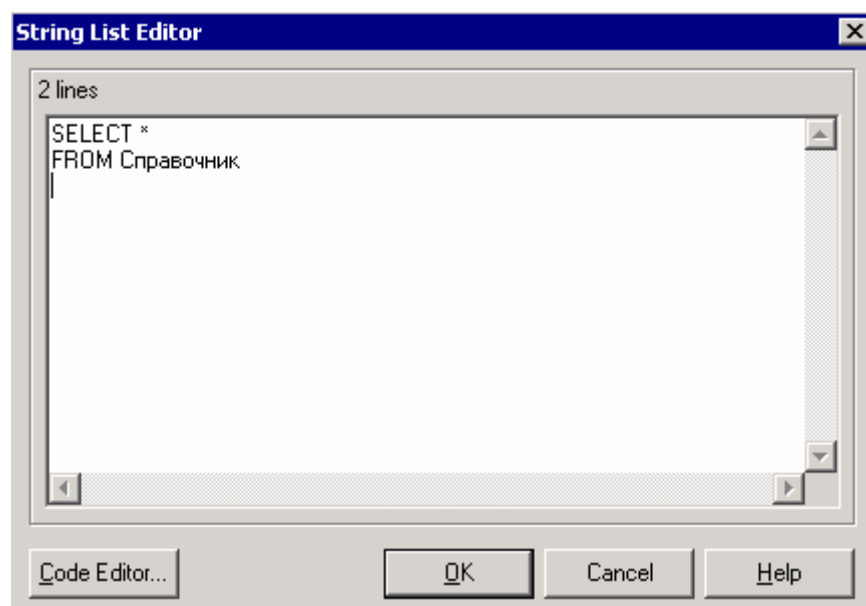


Рисунок 14.9.2 Редактор SQL запросов

В этом редакторе я написал простейший запрос – выбор всех строк и всех столбцов из таблицы «Справочник» базы данных, к которой мы подключились:

---

```
SELECT *  
FROM Справочник
```

---

Мы пока только написали запрос, но ещё не выполнили его. Для его выполнения нужно установить свойство *Active* в *true*. Сделай это.

Теперь перейдём в главный модуль, где у нас храниться основное окно. Выдели сетку *DBGrid1*, которая у нас отображает данные из таблицы *BookTable*. Перейди в объектный инспектор и измени свойство *DataSource* на *DataModule1.FindSource*, чтобы увидеть таблицу результата запроса.

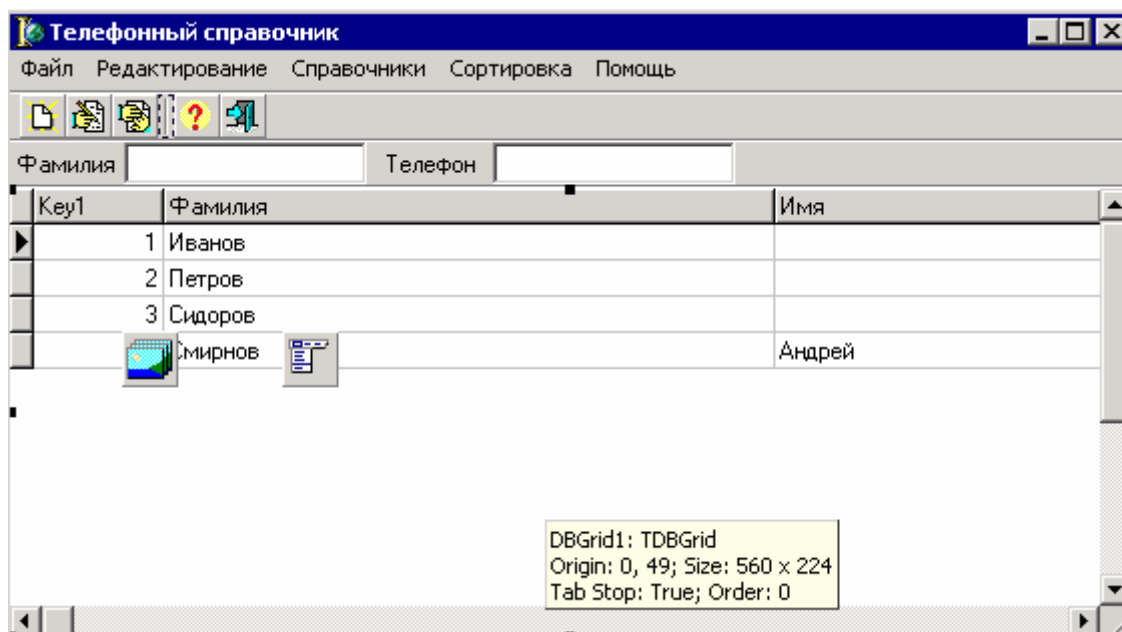


Рисунок 14.9.3 Результат запроса

Посмотри на результирующую таблицу, она похожа на то, что мы видели при работе с компонентом *TADOTable*, только пока что ты видишь все поля (даже те, которые мы уже скрыли из виду от пользователя в компоненте *BookTable*) и отображение полей не настроено. Но всё это можно исправить, если дважды щёлкнуть по компоненту *FindQuery*. Перед тобой откроется то же самое окно настройки свойств полей. Сейчас мы сделаем это, но сначала снова выдели сетку *DBGrid1*, в главном окне. Перейди в объектный инспектор и измени свойство *DataSource* на *DataModule1.DataSource1*, чтобы вернуть всё на родину.

А вот теперь переходи в модуль данных *DataModule* и дважды щёлкни по компоненту *FindQuery*. В появившемся окне щёлкни правой кнопкой мыши и выбери пункт меню «*Add all fields*». В редакторе должны отобразиться все поля нашей таблицы. Теперь дважды щёлкните по компоненту *BookTable*, чтобы отобразить окно свойств нашей уже настроенной таблицы. Расположи оба окна рядом с друг другом, чтобы ты мог всегда их видеть. Теперь выделяй первое свойство в редакторе свойств таблицы *BookTable*, запоминай их, переходи в редактор свойств полей запроса и делай там те же настройки.

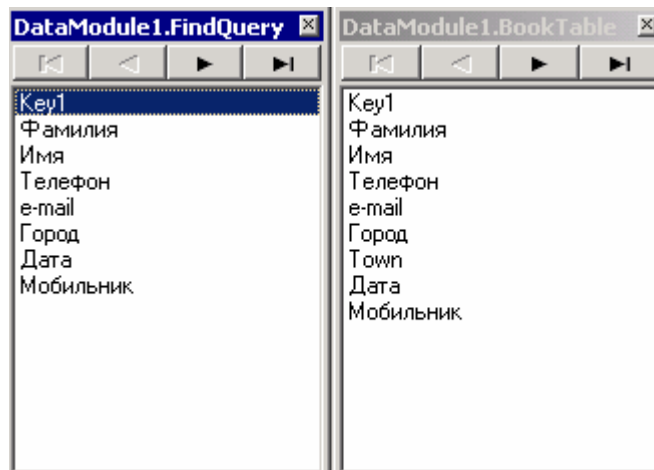


Рисунок 14.9.4 Слева свойства полей запроса, справа свойства полей таблицы

Когда перенесёшь все свойства (не забудь создать поисковое поле для поля «Город»), можешь снова посмотреть на результат используя сетку главного окна. Только опять верни всё на родину. Теперь результат вообще не должен отличаться.

Теперь реализуем непосредственно поиск с помощью нашего SQL запроса. Для этого создадим новую форму, в которой будет отображаться результат и назовём её *FindResultForm*. Теперь измени следующие свойства:

*Caption* – измени на «Результат поиска»;

*Position* – установи в *poMainFormCenter*, чтобы наше окно отображалось по центру главного окна.

Чтобы окно видело таблицы, к нему надо подключить наш модуль данных – *DataModuleUnit*. Для этого используй уже знакомое меню *File->Use Unit*. Теперь брось на форму сетку *DBGrid* с закладки *Data Controls* палитры компонентов и растяни её по всей форме. Перейди в объектный инспектор и измени свойство *DataSource* на *DataModule1.FindSource*, чтобы увидеть в сетке результат запроса.

Всё, можно сохранять форму под именем *FindResultUnit*. Внешний вид моего окна ты можешь увидеть на рисунке 14.9.5.

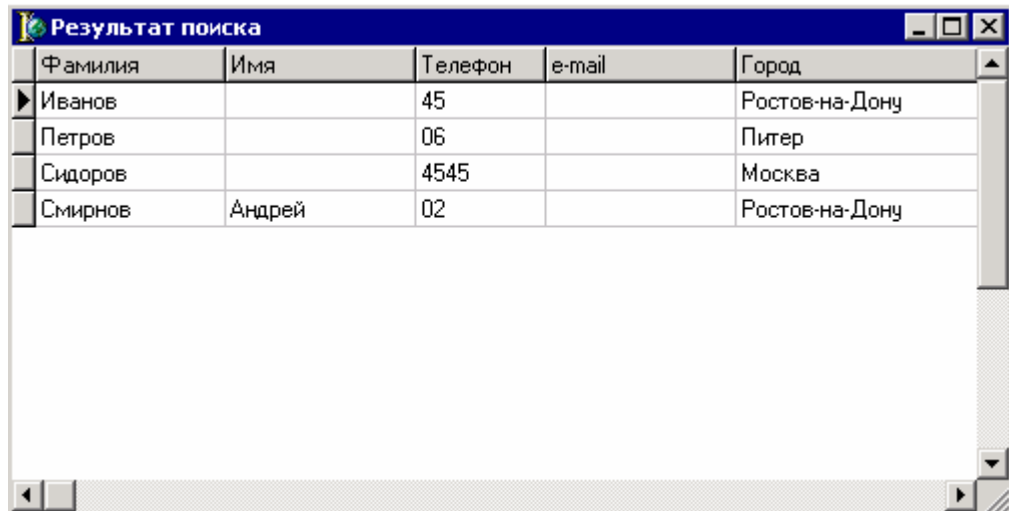


Рисунок 14.9.5 Окно результата поиска.

Теперь перейди на главную форму и на панели поиска добавь надпись и строку ввода для поиска по телефону. Самой последней добавь кнопку, по нажатию которой будет запускаться поиск. На рисунке 14.9.6 ты можешь видеть улучшенное главное окно нашей программы.

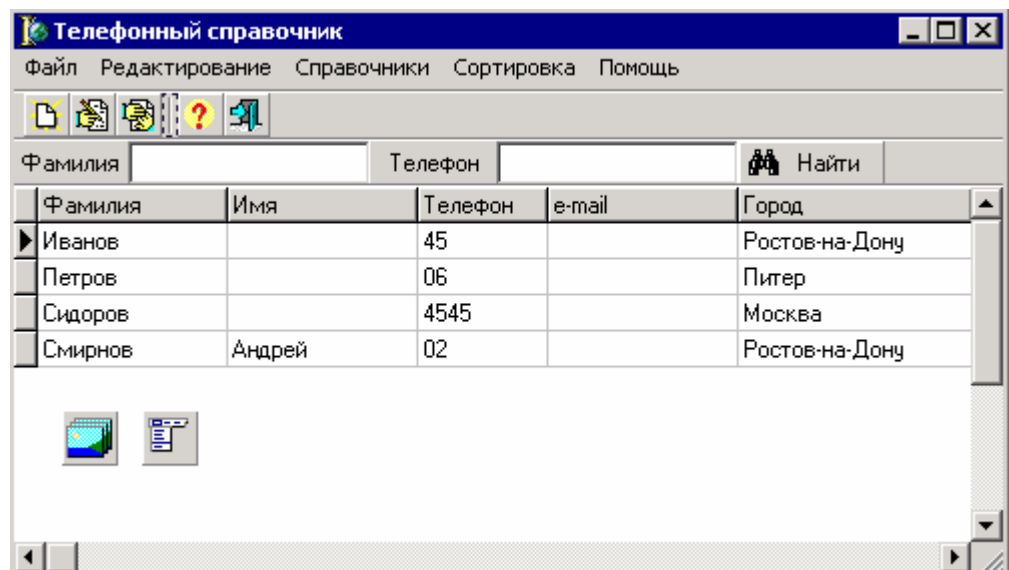


Рисунок 14.9.6 Улучшенная панель поиска.

По нажатию кнопки «Найти» пишем следующий код:

```
procedure TMainForm.FindButtonClick(Sender: TObject);
begin
  DataModule1.FindQuery.Active:=false;
  DataModule1.FindQuery.SQL.Clear;
  DataModule1.FindQuery.SQL.Add('SELECT *');
  DataModule1.FindQuery.SQL.Add('FROM Справочник');
  DataModule1.FindQuery.SQL.Add('WHERE Телефон LIKE '"+FindTelephoneEdit.Text+"'");
  DataModule1.FindQuery.Active:=true;

  FindResultForm.ShowModal;
end;
```

В первой строке кода я делаю компонент запроса неактивным. После этого мне надо заполнить свойство *SQL* запросом на поиск данных. Это свойство имеет уже знакомый тип *TStrings*, который мы использовали при работе с элементами списка *TListBox*, *TComboBox* и др. Но прежде чем заполнять новыми значениями, нужно очистить свойство от старого запроса, который мог остаться после последнего вызова (если пользователь уже нажимал кнопку «Найти», то там будет старый запрос, который при следующем нажатии надо очистить). Для очистки вызываем метод *Clear*.

Далее идёт заполнение свойства *SQL* текстом запроса. Нам нужно внести следующий запрос:

---

```
SELECT *  
FROM Справочник  
WHERE Телефон LIKE ''' + FindTelephoneEdit.Text + '''
```

---

Здесь написано примерно следующее: выбрать все поля из таблицы «Справочник», где поле «Телефон» равно указанному в компонент *FindTelephoneEdit* тексту. Обрати внимание, что параметр, с которым я сравниваю (текст из *FindTelephoneEdit*) должен быть заключён в кавычки. Поэтому как и при работе с фильтром, мне приходится использовать множество одинарных кавычек.

Как только текст запроса занесён в свойство *SQL*, его можно выполнять. Для этого я делаю компонент активным. Ну и чтобы отобразить результат, я показываю окно *FindResultForm*.

---



Для выполнения запроса чаще всего достаточно сделать компонент *ADOQuery* активным. Это прекрасно работает, если ты запрашиваешь данные из таблицы базы данных. Но если в запросе ты удаляешь строки или изменяешь структуру таблицы (в запросе есть такие операторы как *INSERT*, *UPDATE*, *DELETE*, или *CREATE TABLE*), то необходимо вызывать метод *ExecSQL* компонента *ADOQuery*.

---

 На компакт диске, в директории [\Примеры\Глава 14\SQL1](#) ты можешь увидеть пример этой программы.

---

В принципе пример готов, и на этом можно было бы остановиться, но я хочу показать тебе ещё один вариант использования динамических запросов. Что я понимаю под словом «динамический»? Если мы просто вписали запрос в свойство *SQL* компонента *ADOQuery* и в течении всей программы его не изменяем, то такой запрос можно назвать статическим. Но если в течении выполнения программы нам надо изменять текст запроса, то его можно назвать динамическим.

При поиске записей нам приходится изменять запрос, потому что каждый раз используется разный параметр, который необходимо найти. Но зачем же каждый раз изменять весь запрос, когда он не меняется? Не легче ли внести в запрос переменную и изменять только её? Легче, поэтому давай подкорректируем наш пример.

Выдели компонент *FindQuery* и дважды щёлкни по свойству *SQL*. В редакторе запроса введи следующий запрос:

```
SELECT *  
FROM Справочник  
WHERE Телефон LIKE :Telephone
```

---

В принципе, здесь написан практически тот же запрос, что мы уже использовали. Единственная разница – вместо параметра *FindTelephoneEdit.Text* стоит *:Telephone*. Что это такое? Это переменная, о чём говорит двоеточие вначале имени. Переменная в *SQL* запросе оформляется как:

---

**:ИмяПеременной**

---

Закрой окно редактора запроса и дважды щёлкни по свойству *Parameters*. Перед тобой откроется окно редактора параметров (рисунок 14.9.7). Как видишь, описанный в запросе параметр автоматически попадает сюда. Выдели параметр *Telephone* и посмотри на его свойства в объектном инспекторе.

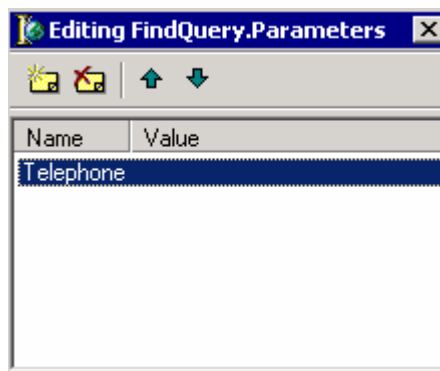


Рисунок 14.9.7 Редактор параметров.

В свойстве *DataType* ты должен указать тип переменной. В нашем случае будет строка с номером телефона, поэтому выбери из списка тип *ftString*. В свойстве *Value* ты можешь указать значение по умолчанию. Попробуй указать там любой номер телефона из твоей базы и сделать компонент *FindQuery* активным. Теперь открой окно, которое должно отображать результат поиска и посмотри на сетку, в которой уже отображаться результат поиска.

Ну и наконец подправим обработчик события кнопки «Найти». Там мы полностью формировали запрос, но теперь этого не надо делать. Достаточно только изменить значение параметра. Для этого удали старый код обработчика и напиши следующий:

---

```
DataModule1.FindQuery.Active:=false;  
DataModule1.FindQuery.Parameters.ParamByName('Telephone').Value:=  
    FindTelephoneEdit.Text;  
DataModule1.FindQuery.Active:=true;  
  
FindResultForm.ShowModal;
```

---



В самом начале я так же делаю компонент запроса неактивным. После этого надо изменить значение параметра. Для этого я использую конструкцию *DataModule1.FindQuery.Parameters.ParamByName('Telephone').Value*. Сложно? Зато удобно.

Все параметры хранятся в свойстве *Parameters* компонента запроса. В этом свойстве, чтобы найти нужный параметр я использую метод *ParamByName*. В качестве единственного параметра этому методу нужно передать имя нашего параметра. Ну и наконец в свойстве *Value* мы записываем значение для найденного параметра.

После этого, запрос можно делать активным, чтобы он выполнялся и мы увидели результат. Как видишь, такая работа с динамическими запросами на много легче, особенно, если запросы большие, а изменяется только какое-то значение, которое можно заменить на переменную.

 На компакт диске, в директории \Примеры\Глава 14\SQL1 ты можешь увидеть пример этой программы.

Я думаю, что на этом можно закончить разговор про запросы, осталось лишь добавить пару слов. Как я уже сказал, компонент *ADOQuery* очень похож на *ADOTable*. В них очень много общего и часто используют в качестве основного доступа к данным именно *ADOQuery*. В нашем случае можно было поступить так же, потому что основное предназначение телефонного справочника – поиск необходимой информации. А вот дополнительные справочники (как, например, справочник городов) можно реализовывать в виде таблицы *ADOTable*.

Компонент *ADOQuery* имеет все необходимые методы необходимые для полноценной работы с базой данных, такие как *Insert*, *Delete*, *Edit*, *Post*, *First*, *Next*, *Prev*, *Last* и т.д., которые мы рассматривали для компонента *ADOTable*.