

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

14.10 Связанные таблицы.....	359
14.11 Вычисляемые поля. ....	365
14.12 Цветные сетки DBGrid.....	368
14.13 Подключение к базе данных во время выполнения программы. ....	371

## 14.10 Связанные таблицы

Ты наверно можешь сказать, что у нас получился полноценный телефонный справочник. В него уже можно не только заносить данные, редактировать, удалять, но и искать по разным параметрам. Я дал уже достаточно информации, чтобы ты сам смог улучшить этот пример, но не всё так просто.

Представь себе ситуацию, когда у одного человека есть два телефона. Один из телефонов может быть простым, а другой может быть сотовым. Как внести такую информацию в нашу базу? Нужно внести две записи, в которых поля Фамилия, Имя, Город, e-mail и Дата рождения будут одинаковыми. Но это же неудобно и сразу заметно, что идёт лишний расход места на диске на хранение двух практически одинаковых строк.

Такая ситуация нарушает рациональность хранения информации. Я же говорил, что строки таблицы должны хранить как можно меньше одинаковой информации. Именно поэтому мы убрали поле Город в отдельный справочник, чтобы сэкономить место на диске и увеличить эффективность нашей базы данных. Но как поступить в данном случае, когда фамилию, имя и др. поля убирать в справочник нет смысла, да и телефоны держать в справочнике нерационально? Очень просто. В этом случае нам помогут связанные таблицы.

В одной таблице надо хранить следующие данные: Фамилия, Имя, Город, e-mail и Дата. В другой таблице будут: Телефон и Мобильник. Обе таблицы будут связаны между собой, как на рисунке 14.10.1.

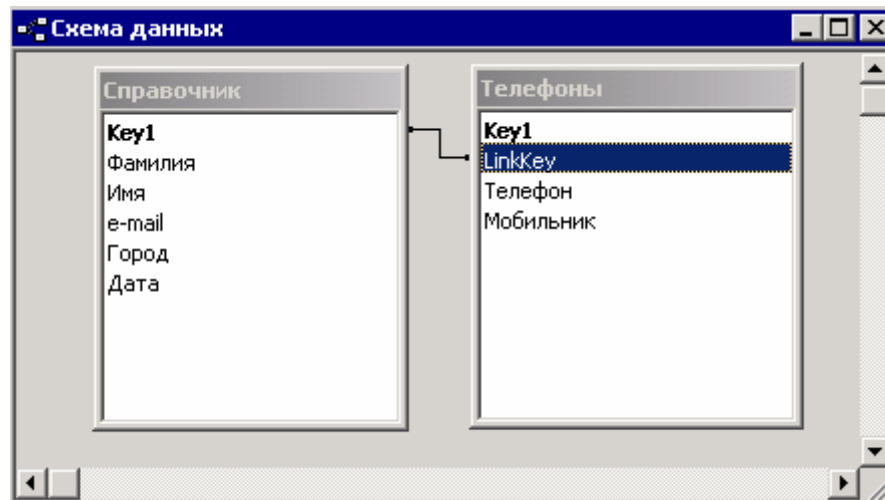


Рисунок 14.10.1 Схема данных

Здесь показаны две таблицы. В справочнике ты видишь все описанные мной поля общих сведений о владельце телефона. Во второй таблице «Телефоны» у нас четыре поля: счётчик, *LinkKey*, телефон и мобильник. Поля телефон (строковое поле, хранящее реальный номер телефона) и мобильник (логическое поле) выполняют ту же роль, что и раньше одноимённые поля в основной таблице «Справочник». В основной таблице теперь этих полей нет.

Что же такое *LinkKey* и почему между ним и полем «*Key1*» таблицы «Справочник» я провёл линию связи? *Key1* – уникальное поле, по которому мы точно можем его найти. *LinkKey1* – связующее поле и будет хранить ссылки на поле *Key1*. Например, взгляни на пример двух таблиц, показанный на рисунке 14.10.2.

Справочник			Телефоны			
Key1	Фамилия	Имя	Key1	Link	Телефон	Мобил
1	Иванов	Сергей	1	1	3351010	Нет
2	Петров	Иван	2	2	3461010	Да
			3	2	2125555	Нет
3	Иванов	Лёша	4	3	4547777	Да
			5	3	6562244	Нет

Рисунок 14.10.2 Схема данных

Слева на рисунке показана таблица «Справочник», а справа таблица «Телефоны». Если посмотреть на данные, то поле *Key1* постоянно увеличивается на единицу, потому что это счётчик. Теперь давай посмотрим на этом примере, как происходит связь через поля *Key1* таблицы «Справочник» и *LinkKey* таблицы «Телефоны».

В левой таблице у нас первая запись принадлежит Иванову Сергею. Во второй таблице ищем записи, у которых в поле *Link* стоит цифра 1. Такая запись только одна и она первая.

Вторая запись в левой таблице принадлежит Петрову Ивану. Смотрим в правой таблице записи, у которых в поле *Link* находится число 2. Таких записей аж две (2-я и 3-я), значит у Петрова есть два телефона.

Третья запись в левой таблице принадлежит Иванову Алексею. Ищем в правой таблице записи с цифрой 3 в поле *Link*. Таких записей опять 2, значит и у Алексея тоже есть два телефона.

Таким образом, мы связываем две таблицы с помощью ключей. Когда мы создавали поисковые поля, у нас получалась связь, чем-то похожая на эту, только тогда главная таблица содержала поле *Город* которое подвязывалось под главный ключ справочника городов.

На словах сказано, пора и сделать. Переходим к нашему примеру. Для начала нужно открыть базу данных в Access и отредактировать поля таблицы *Справочник*, а именно убрать поля *Телефон* и *Мобильник*.

Теперь создавай новую таблицу в режиме конструктора со следующими полями:

---

*Key1* – счётчик, ключевое поле.

*LinkKey* – числовое поле, в свойстве (*Индексированное Поле*) укажи (*Да*) (*Допускаются совпадения*).

*Телефон* – текстовое, размер 10.

*Мобильник* – логическое.

---

Сохрани таблицу под именем *Телефоны*.

Теперь запускай Delphi. Открой модуль с компонентами для доступа к данным и дважды щёлкни по компоненту *BookTable*. В редакторе полей выдели поле *Телефон* и удали его кнопкой Del. Потом удали поле *Мобильник*, потому что мы удалили эти поля из базы. Удали эти же поля из компонента *FindQuery* плюс ещё здесь надо удалить поле *Key1*

(это потому что запрос на поиск будет проходить сразу на две таблицы и в обоих есть поле с именем Key1). Потом ты увидишь, почему мы удалили ключевое поле. А пока помни, что поиск у нас не работает, потому что SQL запрос сейчас неправильный. Чуть позже я исправлю его.

Теперь добавь сюда компоненты *DataSource* (назови его *TelephonSource*) и *ADOTable* (назови его *TelephonTable*) для доступа к таблице «Телефоны». Наведи свойство *DataSource* компонента *TelephonSource* на *TelephonTable*.

Теперь установи следующие свойства компонента *TelephonTable*:

*Connection* – укажи здесь наш компонент присоединения к базе данных *ADOConnection1*.

*TableName* – здесь укажи имя таблицы *Телефоны*.

*Active* – установи в *true*, чтобы открыть таблицу.

*MasterSource* – выбери здесь из выпадающего списка *BookSource*. Этим ты указываешь главную таблицу для таблицы *Телефонов*.

*MasterFields* – здесь мы должны указать связующие поля. Щёлкни по этому полю дважды и перед тобой откроется окно, как на рисунке 14.10.3. В списке *Detail Fields* (поля подчинённой базы) выбери поле *LinkKey*, а в списке *Master Fields* (поля главной базы) выбери поле *Key1*. Нажми кнопку *Add* и в списке *Joined Fields* (связанные поля) появится строка отображающая выделенную связь. Закрой окно, кнопкой *OK* чтобы сохранить указанную связь.

Посмотри теперь в объектном инспекторе на свойство *IndexFieldNames*. Как видишь, там появилось имя поля *LinkKey* – поля через которое происходит связь. Так что в связанных таблицах нельзя использовать это поле для обеспечения сортировки, иначе нарушится связь. ПОМНИ ЭТО!!!

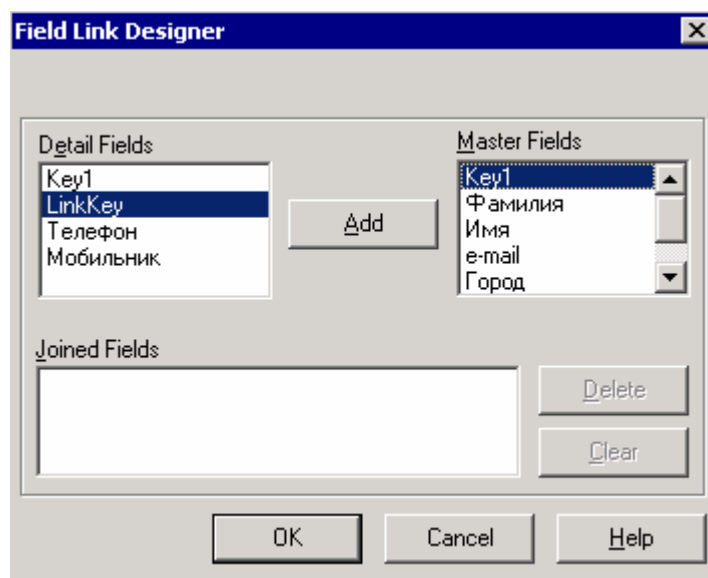


Рисунок 14.10.3 Окно создания связей между главной и подчинённой таблицей

Теперь дважды щёлкни по компоненту *TelephonTable*, чтобы увидеть окно редактирования свойств полей. Здесь тебе надо добавить поля, чтобы ты мог обращаться к ним потом по именам и спрятать первые два поля (ключевые поля, которые не несут пользователю полезной информации).

Теперь можно переходить к программированию или почти к программированию. Открой главную форму и добавь сюда ещё одну сетку *DBGrid*. Я расположил её по правому краю окна, как показано на рисунке 14.10.4. У сетки нужно изменить только свойство *DataSource* указав там нашу таблицу телефонов *DataModule1.TelephonSource*.

Может ты уже заметил, но я люблю ещё устанавливать свойство *BorderStyle* в *bsNone*, так красивее смотрится.

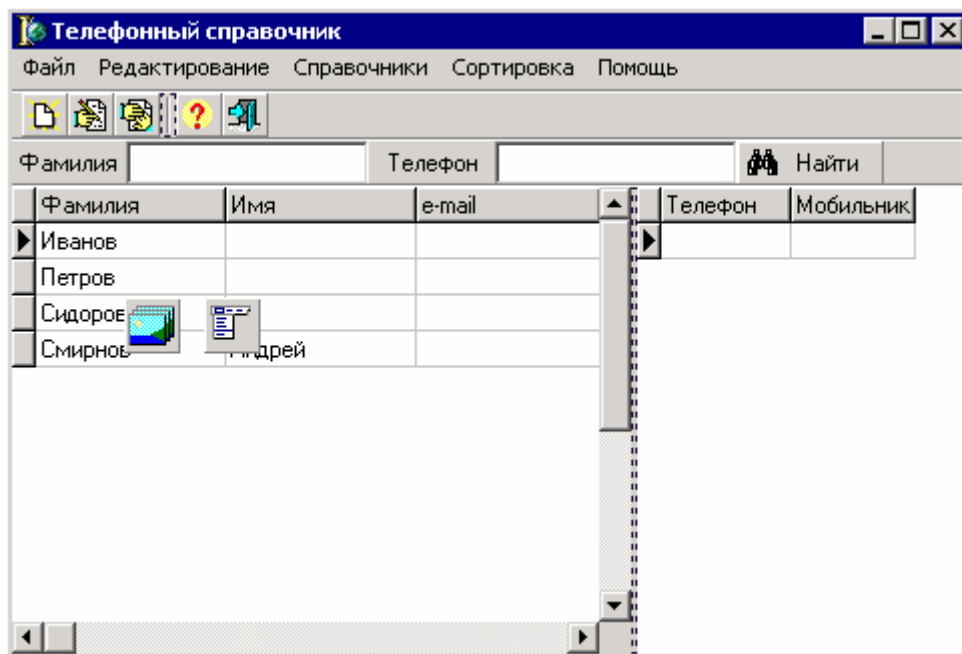


Рисунок 14.10.4 Улучшенная главная форма программы

В принципе, пример готов к запуску. Ты можешь выбирать в левой сетке любого человека, а в правой сетке можно вносить любое количество телефонов для выбранной записи. Для вставки новой записи можно использовать кнопку *Ins*, для удаления *Ctrl+Del*, чтобы сохранить изменения, нужно курсором перейти на другую запись, отменить редактирование записи (если изменения ещё не приняты) – кнопка *Esc*.

Вообще, на счёт сохранения изменений могу сказать следующее. Когда ты пишешь программы не для собственных нужд, то пользователи очень часто забывают или даже не знают о том, что для запоминания введённых изменений надо перейти на другую запись. Именно поэтому я люблю по событию *OnClose* для главной формы писать следующий код для всех таблиц моей программы:

---

```
if Table1.Modified then  
    Table1.Post;
```

---

Здесь я проверяю, если во время закрытия программы таблица *Table1* изменена, то запоминаю изменения.

Ещё один способ следить за изменениями – устанавливать в сетках *DBGrid* в свойстве *Options* параметр *dgCancelOnExit* в *false*. А лучше использовать сразу оба этих способа.

Теперь давай подправим наш запрос на поиск по телефону. Открой модуль данных и введи в компонент *FindQuery* следующий запрос:

---

```
SELECT *  
FROM Справочник, Телефоны  
WHERE Телефон LIKE :Telephone  
AND Справочник.Key1=Телефоны.LinkKey
```

---

Здесь я уже выбираю все поля из двух таблиц, где есть записи с указанным значением телефона. К тому же здесь указана связь между таблицами. Если ты прочитал мой мануал по SQL, который идёт на диске, то с пониманием этого запроса проблем не будет.

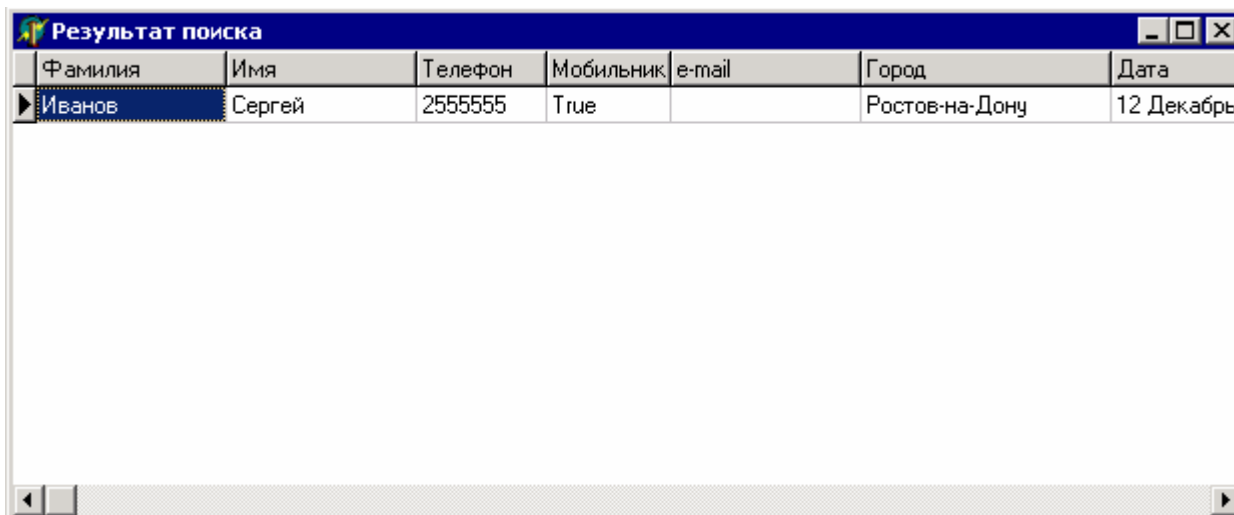
Закрой редактор SQL запроса и дважды щёлкни по компоненту *FindQuery*. В редакторе свойств полей щёлкни правой кнопкой мыши и выбери пункт *Add All Fields*. В редактор будут добавлены все новые поля, которых в нём не было (это поля из таблицы *Телефоны* и ключевое поле таблицы *Справочник*) Обрати внимание, что теперь нет поля *Key1*. У нас две таблицы и в обеих есть поле *Key1*, поэтому для их различия названия полей изменены на следующий вид:

---

#### Имя Таблицы . Имя поля

---

Спрячь ключевые поля, потому что пользователь не должен их видеть и отсортируй всё так, чтобы удобно было работать. Попробуй запустить программу и посмотреть на результат (рисунок 14.10.5). Несмотря на то, что данные о телефонах хранятся в трёх таблицах, мы их получаем от SQL запроса в виде одной.



Фамилия	Имя	Телефон	Мобильник	e-mail	Город	Дата
Иванов	Сергей	2555555	True		Ростов-на-Дону	12 Декабрь

Рисунок 14.10.5 Улучшенная главная форма программы

Обрати внимание, что в окне результата поиска можно редактировать данные и они будут правильно отображены в своих таблицах. Попробуй изменить какое-нибудь поле. Когда ты закроешь окно результата поиска ты пока ничего не увидишь, потому что в сетке главной формы данные нужно обновить методом *Refresh* таблиц. Так что чтобы увидеть изменения, нужно закрыть программу и открыть её снова.

Чтобы не встречаться с такой проблемой, нужно подкорректировать обработчик события на нажатие кнопки «Найти». Допиши в конце (после показа окна результата) следующие строки:

---

```
DataModule1.BookTable.Refresh;  
DataModule1.TelephonTable.Refresh;
```

---

Кстати, в нашей программе можно смело удалять сортировку, потому что использовать её в том виде, в котором мы её написали нельзя. Мы сортировали с помощью индекса, а в связанных таблицах индекс выполняет роль связей. Если мы изменим значение индексного поля, то нарушим связь между таблицами.

Но мы ещё можем воспользоваться свойством *Sort* таблицы. Единственное – мы не сможем сортировать по телефону, потому что он находится в другой таблице, но можем упорядочить записи по любому полю главной таблицы. По нажатию пункта меню «По фамилии» из меню «Сортировка» напиши следующий код:

---

```
DataModule1.BookTable.Sort:='Фамилия ASC';
```

---

Пункт меню «По телефону» можно убирать, а вместо него давай сделаем сортировку по городу. По его нажатию пиши следующий код:

---

```
DataModule1.BookTable.Sort:='Город ASC';
```

---

Программу можно считать законченной, если бы не одно но – нужно подкорректировать окно редактирования данных. У нас изменилась главная таблица, значит и это окно должно измениться. Я не буду тратить место в книге на объяснения, как изменить окно, попробуй сделать это сам, потому что все необходимые знания у тебя уже есть. На рисунке 14.10.6 ты можешь видеть моё окно, постарайся привести его к такому виду. В принципе, тут добавлена только одна сетка для телефонов и три кнопки для добавления, редактирования и удаления записей.

Телефон	Мобильник

Рисунок 14.10.6 Новая форма редактирования данных о пользователе.

Для добавления и редактирования записей нам нужно ещё одно новое окно. Создай новую форму и назови её *PhoneEditForm*. Её внешний вид ты можешь увидеть на рисунке 14.10.7.

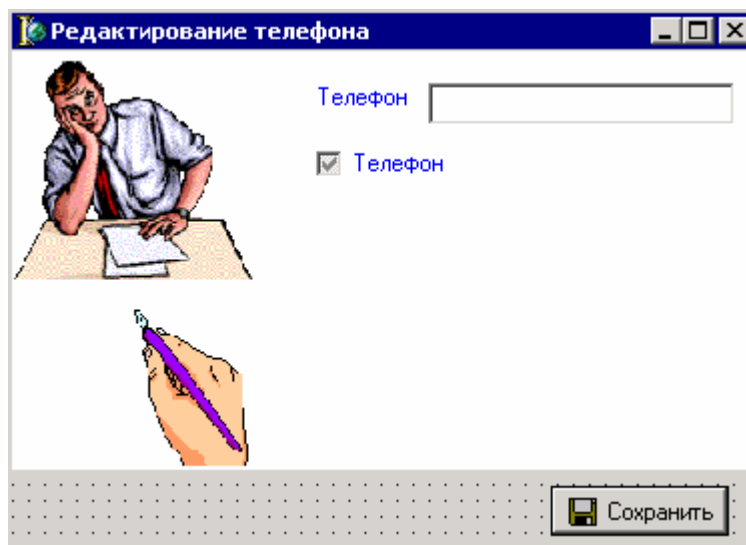



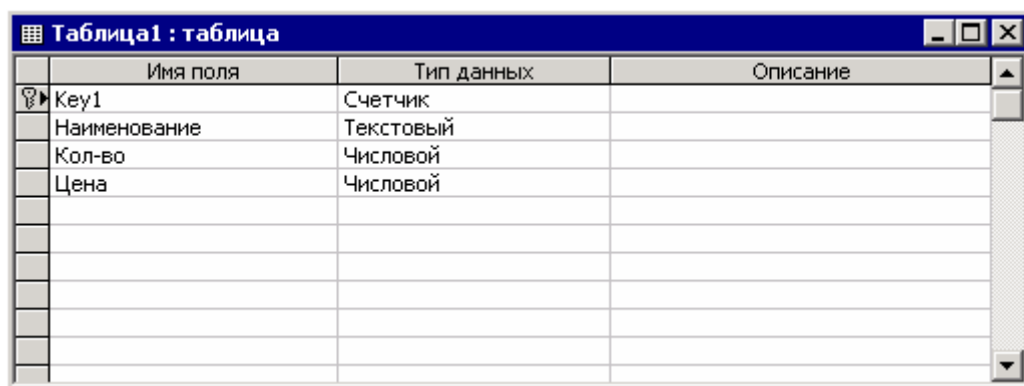
Рисунок 14.10.7 Окно для редактирования данных о телефоне

Код для кнопок добавления, редактирования и удаления записей попробуй написать сам. Он похож на тот, что мы уже писали для окна редактирования данных о пользователе. Если что-нибудь будет непонятно, ты всегда сможешь обратиться к исходникам на диске.

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

## 14.11 Вычисляемые поля.

Допустим, что у тебя есть база данных со следующими полями: Наименование, кол-во, цена. А почему «допустим», давай создадим новую базу данных в которой будет таблица с такими полями (рисунок 14.11.1). Сохрани эту таблицу под именем «Товары». В ней мы будем хранить наименование покупки, кол-во и цену.



Имя поля	Тип данных	Описание
Key1	Счетчик	
Наименование	Текстовый	
Кол-во	Числовой	
Цена	Числовой	

Рисунок 14.11.1 Таблица «Товары» новой базы данных



Цена в таблицы будет указываться за 1 товара, т.е. за штуку, кг или метры. Чтобы узнать общую цену товара, надо цену за единицу умножить на кол-во товара. При этом, итог должен моментально реагировать на любые изменения колонок кол-ва и цены. На первый взгляд задача достаточно сложная, но в программировании она проста, как никогда.

Создай новый проект в Delphi, и сразу добавь в него модуль *DataModule*. В этот модуль кинь компоненты *ADOConnection* (для соединения с базой данных), *DataSource* для возможности отображения данных из таблицы и *ADOTable* для соединения с таблицей (на рисунке 14.11.2 показано моё окно *DataModule*). Подключись к новой базе данных с помощью компонента *ADOConnection1*.

У *DataSource1* в свойстве *DataSet* укажи таблицу *ADOTable1*. В таблице *ADOTable1* в свойстве *Connection* укажи компонент *ADOConnection1*, в свойстве *TableName* нужно указать нашу таблицу «Товары». После этого можно делать таблицу активной (в свойстве *Active* нужно указать *true*).

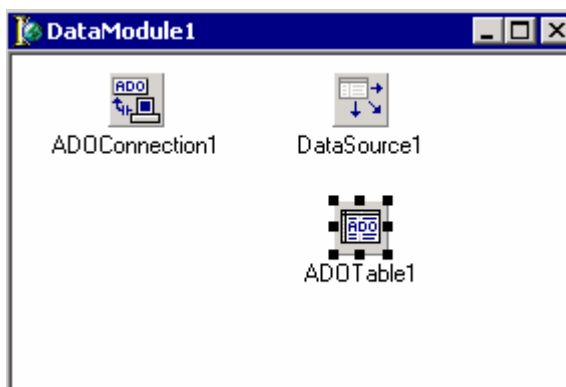


Рисунок 14.11.2 Окно *DataModule*.

Теперь щёлкай дважды по компоненту *ADOTable1* и в появившемся окне редактора свойств добавляй все поля таблицы. Для начала здесь надо сделать невидимым ключевое поле. Потом нужно установить значения по умолчанию для полей *Кол-во* и *Цена*. Эти поля будут участвовать в математических расчётах, поэтому в них обязательно должны быть какие-нибудь значения. Если в одном из полей не будет данных, то программа во время расчётов выдаст ошибку. Для поля *Кол-во* я указал в свойстве *DefaultExpression* (значение по умолчанию) единицу, а для поля *Цена* в том же свойстве поставил ноль.

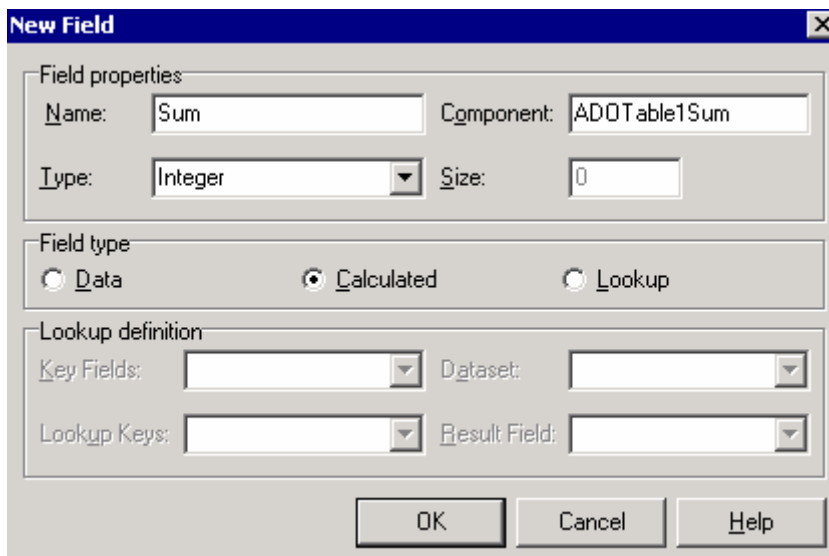


Рисунок 14.11.3 Окно создания нового поля.

Теперь создадим новое поле, которое будет хранить итог расчётов. Но прежде чем это делать, нужно сделать таблицу неактивной. Как ты помнишь, при создании поисковых полей я предупреждал, что новое поле можно создавать только при неактивной таблице. Щёлкой правой кнопкой в окне редактора свойств и выбирай пункт *New Field*. В окне свойств нового поля заполни следующие поля:

*Name* (имя нового поля) – назовём поле *Sum*.

*Type* (тип поля) – у нас будет числовая сумма, поэтому выбирай тип *Integer*.

*Field Type* (тип поля) – выбирай *Calculated*, чтобы создать вычисляемое поле.

На рисунке 14.11.3 ты можешь увидеть заполненное мной окно свойств созданного нового поля. Как только поле создано, таблицу снова можно делать активной.

Теперь выдели компонент *ADOTable1* и создай обработчик события *OnCalcFields*. Это событие вызывается каждый раз, когда надо пересчитать вычисляемые поля. Оно будет вызываться для всех видимых пользователю записей. В этом обработчике напиши следующее:

---

```
procedure TDataModule1.ADOTable1CalcFields(DataSet: TDataSet);
begin
  ADOTable1Sum.Value:=ADOTable1DSDesigner2.AsInteger*
    ADOTable1DSDesigner3.AsInteger;
end;
```

---

Прежде чем разбираться с этим кодом, открой окно редактора свойств полей таблицы *ADOTable1* и посмотри имена (свойство *Name*) полей *Кол-во*, *Цена*, и *Sum*. У меня это *ADOTable1DSDesigner2*, *ADOTable1DSDesigner3* и *ADOTable1Sum* соответственно. С помощью этих имён мы можем обращаться к значениям, находящимся в полях. Надо только написать имя поля и вызвать один из его методов, для преобразования значения в нужный формат. Тебе доступны следующие методы полей:

*AsInteger* – получить значение, хранящееся в данном поле в виде числа.

*AsDateTime* – в виде объекта *TDateTime*.

*AsBoolean* – в виде булева значения.

*AsCurrency* – в виде цены.

*AsFloat* – в виде вещественного значения.

*AsString* – в виде строки.

*AsVariant* – в виде типа *Variant*. Это универсальный тип, который может принимать любые значения, хоть число, хоть строку, в общем любые доступные типы.

Теперь взгляни на код и сразу же встанет всё понятно. Здесь мы записываем в свойство *Value* поля *ADOTable1Sum* результат перемножения значений полей цены и количества. Значения полей я получаю как целые числа – *AsInteger*.


Теперь переходим в главное окно нашей программы. Подключи к нему модуль *DataModule* (File->Use Unit) и брось на форму одну сетку *DBGrid*. Теперь в свойстве *DataSource* сетки выбери нашу таблицу *DataModule1.DataSource1*. Всё!!! Программа готова. Запускай, и попробуй ввести в базу несколько полей. На рисунке 14.11.4 ты можешь увидеть окно результата работы моего примера, но я тебе советую самому попробовать поиграть с ним.

Кстати, поле итога не должно изменяться пользователем вручную, потому что оно вычисляемое. Именно поэтому ты даже не сможешь туда ввести никакого значения. Delphi просто блокирует любые такие попытки, хотя поле и не имеет признака «Только для чтения» (*ReadOnly*).



Наименование	Кол-во	Цена	Sum
Хлеб	2	5	10
Колбаса	1	75	75
Чипсы	5	10	50
Пиво	10	12	120

Рисунок 14.11.4 Результат работы программы.

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

## 14.12 Цветные сетки DBGrid.

Я уже много раз встречался с проблемой, когда пользователи просят, чтобы у них в программе была возможность выделять некоторые записи каким-нибудь цветом. Это действительно удобно, да и в кодинге не сильно сложно. Сейчас я тебе покажу, как это делается.

Открой базу данных из предыдущей главы и добавь туда поле *Color*. Это поле должно иметь текстовый тип, а размер поля достаточно установить в 15 символов.

Теперь открывай Delphi. Загружай тот же пример и в модуле данных дважды щёлкай по компоненту *ADOTable1*. Выбери уже надоевший пункт «Add All Fields». В редактор будут загружены новые поля, точнее сказать одно поле – *Color*. Сделай его сразу же невидимым, потому что пользователю не надо видеть его текст, его интересует цвет строки.

Теперь перейди в главное окно программы. Добавь сюда компонент *PopUpMenu* (всплывающее меню). Щёлкни по нём дважды, чтобы открыть редактор меню. Создай в нём следующие пункты:

1. Чёрный.
2. Красный.
3. Зелёный.
4. Жёлтый.
5. Синий.
6. Пурпурный.

Имена пунктов меню должны идти именно в таком порядке, а в свойстве *Tag* всех этих пунктов должен находиться порядковый номер пункта. Нумеровать цвета надо с нуля и до 5. Это значит, что у пункта меню *Жёлтый* в свойстве *Tag* будет находиться число 3, а у пункта *Пурпурный* значение 5.

На рисунке 14.12.1 ты можешь увидеть созданное мной меню.

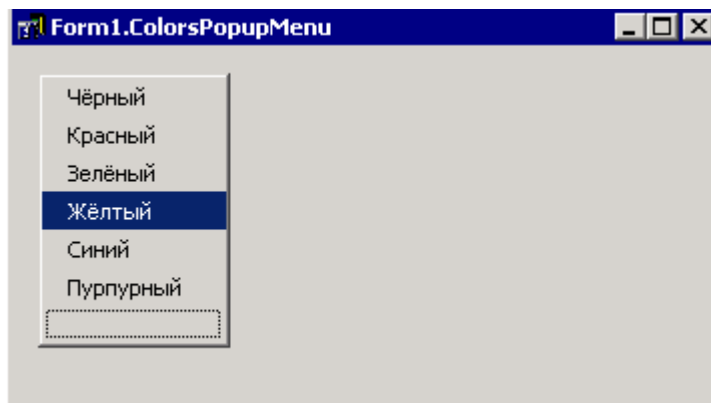


Рисунок 14.12.1 Всплывающее меню выбора цвета.

Теперь выдели все эти пункты меню (щёлкни на первом пункте и удерживая Shift на последнем). Перейди на закладку *Events* объектного инспектора и создай обработчик события *OnClick*. Будет создана процедура-обработчик события, которая будет назначена всем выделенным пунктам меню, т.е. одна процедура отвечает за нажатие любого из пунктов. В этом обработчике напиши следующий код:

---

```
procedure TForm1.N13Click(Sender: TObject);
const
  MenuColors: array[0..5] of TColor =(clBlack, clRed,
    clGreen, clYellow, clBlue, clPurple);
begin
  //Перейти в режим редактирования поля
  DataModule1.ADOTable1.Edit;

  //Занову в поле Color выбранный цвет
  DataModule1.ADOTable1.Color.AsString:=
    ColorToString(MenuColors[TMenuItem(Sender).Tag]);

  //Запомнить изменения
  DataModule1.ADOTable1.Post;
end;
```

---

В разделе **Const** я объявил одну константу – массив из 6 элементов имеющих тип *TColor*. Так как это массив-константа и в процессе программирования не может менять свои значения, то эти значения нужно обязательно описать здесь. А именно, сразу же после объявления массива ставиться знак равенства и в скобках перечисляются значения элементов массива. Так как массив из 6 элементов, то и в скобках должно быть именно 6 элементов, ни больше, ни меньше. Я указал цвета, которые у меня указаны в пунктах всплывающего меню. Их имена перечислены в том же порядке, что и в меню.

В первой строке кода я перевожу таблицу в режим редактирования с помощью вызова метода *Edit*. Если этого не сделать, то любые попытки изменить данные в полях текущей записи встретят меня шквалом ошибок.

В следующей строке я присваиваю полю *Color* значение выбранного цвета. Выбранный цвет я получаю следующим образом: *MenuColors[TMenuItem(Sender).Tag]*.

*Sender* – эта переменная передаётся нам в качестве параметра в обработчик события и указывает на объект, который породил данное событие.

*TMenuItem(Sender)* – переменная *Sender* универсальна, поэтому имеет тип **TObject** – родитель всех компонентов. Но в таком виде мы не можем обратиться к свойствам, которых нет у **TObject**, но есть у пунктов меню (нас интересует свойство *Tag*). Поэтому мы показываем компилятору, что данное событие *Sender* имеет тип **TMenuItem**.

*TMenuItem(Sender).Tag* – получаем значение, указанное в свойстве *Tag* пункта меню, породившего это событие.

*MenuColors[TMenuItem(Sender).Tag]* – получаем из массива *MenuColors* значение цвета соответствующее значению, указанному в свойстве *Tag*.

Значение полученного цвета переводиться в строку с помощью функции *ColorToString* и записывается в поле *Color* в виде строки.

Код написан, теперь надо выделить сетку *DBGrid* и в свойстве *PopupMenu* указать созданное нами меню.

Теперь мы можем запускать приложение, изменять свойство *Color* в базе данных и осталось только научить программу читать это свойство и выводить текст в зависимости от указанного там значения цвета. Для этого создай обработчик события *OnDrawDataCell* для нашей сетки. Это событие вызывается, когда нужно перерисовать данные какой-нибудь ячейки сетки. В обработчике напиши следующий код:

---

```
procedure TForm1.DBGrid1DrawDataCell(Sender: TObject; const Rect: TRect;
  Field: TField; State: TGridDrawState);
begin
  try
    DBGrid1.Canvas.Font.Style:=[];
    if (gdSelected in State) or (gdFocused in State)then
      begin
        DBGrid1.Canvas.Brush.Color:=clHighLight;
        DBGrid1.Canvas.Font.Color:=clWhite;
      end
    else
      begin
        DBGrid1.Canvas.Brush.Color:=clWhite;
        DBGrid1.Canvas.Font.Color:=clBlack;

        //Если поле цвета не пустое то использовать цвет из поля
        if DataModule1.ADOTable1Color.AsString<>" then
          DBGrid1.Canvas.Font.Color:=
            StringToColor(DataModule1.ADOTable1Color.AsString);
        end;
        //Очищаю ячейку
        DBGrid1.Canvas.FillRect(Rect);
        //Вывожу текст ячейки
        DBGrid1.Canvas.TextOut(Rect.Left, Rect.Top, Field.AsString);
      except
        DBGrid1.Canvas.TextOut(Rect.Left, Rect.Top, Field.AsString);
      end;
    end;
```

---

Прежде чем описывать этот код хочу тебя предупредить, что пока что у тебя ничего не откомпилируется. Delphi будет ругаться на тип **TField**. Этот тип описан в модуле *db*, поэтому добавь его в раздел **uses**. После этого при компиляции ошибок не должно быть.

Теперь рассмотрим параметры, которые мы получили в обработчик. У нашего обработчика события есть следующие параметры:

*Sender* – объект, который сгенерировал это событие. У нас это будет сетка.

*Rect* – здесь хранятся границы области ячейки, которую надо перерисовать. Границы передаются в виде структуры *TRect*.

*Field* – этот параметр имеет тип *TField* и указывает на поле, которое надо перерисовать.

*State* – здесь находятся параметры, указывающие на текущее состояние ячейки. Возможны следующие параметры:

**gdSelected** – ячейка выделена.

**gdFocused** – ячейка имеет фокус ввода.

**gdFixed** – ячейка является фиксированной. Такие ячейки используются для названий колонок (сверху сетки) и для индикации текущей строки (слева сетки).

Ну а теперь давай знакомиться с кодом самой процедуры. В первой строке я устанавливаю стиль шрифта у холста сетки. Точнее сказать, я очищаю все настройки шрифта, потому что стилю присваивается пустой набор [].

В следующей строке я проверяю, если текущая ячейка выделена или имеет фокус ввода, то цвет кисти изменяю на *clHighLight* (это константа, хранящая цвет, используемый для подсветки). Цвет шрифта я устанавливаю в белый.

Если ячейка не выделена, то цвет фона (цвет кисти) делаю белым, а цвет шрифта чёрным. Далее идёт проверка, если поле цвета текущей строки не пустое, то цвет шрифта меняю на тот цвет, который указан в поле цвета. Единственное, что надо учитывать – цвет храниться в виде строки, поэтому я преобразовываю его в цвет с помощью функции *StringToColor*.

Все приготовления закончены, можно заняться рисованием. Для начала я очищаю старое значение ячейки с помощью вызова метода *FillRect* холста сетки, который закрашивает указанную область цветом кисти (фона). В качестве единственного параметра я указываю область ячейки в виде структуры *TRect*, которую получаю через параметры обработчика.

После закраски я рисую текст ячейки. Если произошла какая-то ошибка во время подготовки к рисованию (она может возникнуть только при преобразовании *StringToColor*, если в поле находится неправильное значение), то я пытаюсь снова вывести текст в блоке *except...end*.

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

#### 14.13 Подключение к базе данных во время выполнения программы.

Использовать заготовленную базу данных очень удобно, но вдруг пользователь захочет выбирать, с какой базой данных ему работать? Зачем это нужно? Допустим, что у тебя программа каждый месяц копирует базу данных в отдельное место и потом очищается, чтобы не содержать устаревших данных. А что если пользователь захотел посмотреть эти старые данные? Ему надо писать отдельную программу для просмотра или придётся пользоваться неудобной программой Access. Но есть выход проще – подключить программу к старой архивной базе данных.

Возьмём пример написанный в прошлой главе. Добавь на форму одну кнопку и по её нажатию напиши следующий код:

---

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  DataModule1.ADOConnection1.Close;
```

```
if EditConnectionString(DataModule1.ADOConnection1) then
begin
  DataModule1.ADOConnection1.Connected:=true;
  DataModule1.ADOTable1.Active:=true;
end;
end;
```

---

Чтобы этот код откомпилировался, нужно в раздел **uses** добавить модуль *ADOConEd*. Вот теперь программа откомпилируется и без проблем запустится.

Теперь посмотрим, что здесь происходит. В первой строке я выполняю метод *Close* компонента *ADOConnection*, чтобы закрыть соединение с базой данных. После этого я вызываю функцию *EditConnectionString*, которая отображает окно подключения к базе данных, которое ты уже видел и можешь ещё раз увидеть на рисунке 14.13.1. В качестве параметра в эту функцию нужно передать компонент *ADOConnection*, параметры которого будут изменяться.

Если пользователь выбрал новое имя файла, то программа вернёт значение *true* и мы откроем соединение с базой и сделаем единственную таблицу активной. Помни, что после закрытия и открытия базы данных, все таблицы становятся неактивными, поэтому тебе придётся программно восстанавливать их активность.

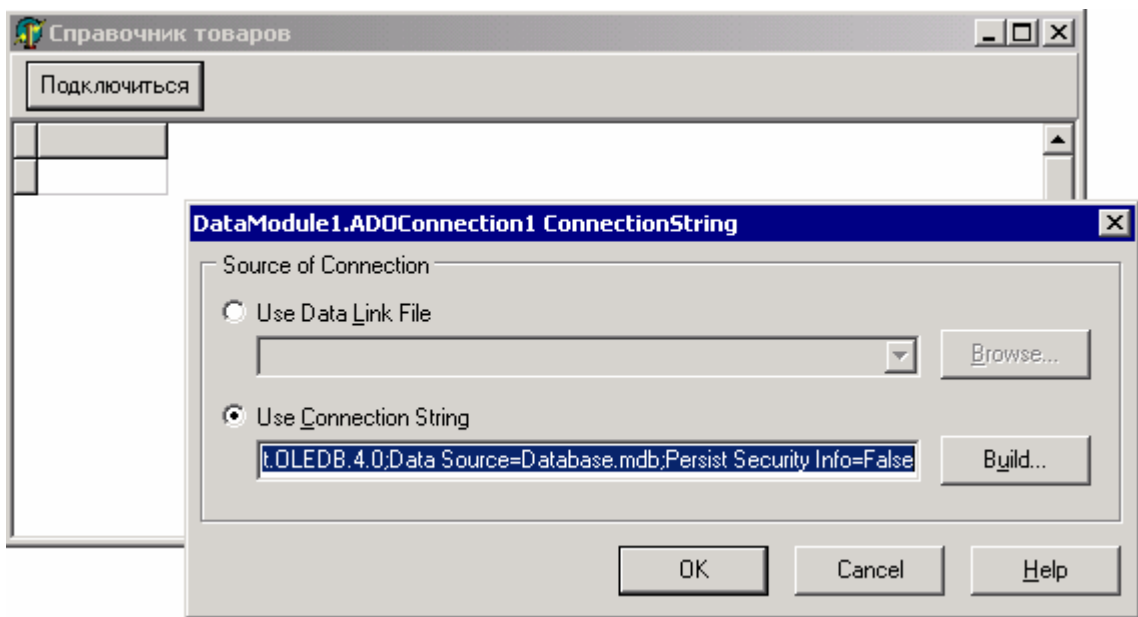



Рисунок 14.13.1 Результат работы программы.

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