Лабораторная работа №5.
Проектирование системы управления базами данных в Delphi.

Время: 360 мин.
Что нужно освоить:
1) каким образом подключать базу данных к компонентам Delphi;
2) порядок создания приложения для управления простейшей локальной базой данных;
3) как управлять отображением таблиц баз данных;
4) как организовать сортировку и фильтрацию данных;
5) как использовать язык запросов SQL.

Создайте папку, в которую будете сохранять разрабатываемые приложения. Для каждого приложения в дальнейшем в главной папке следует создавать отдельный каталог.

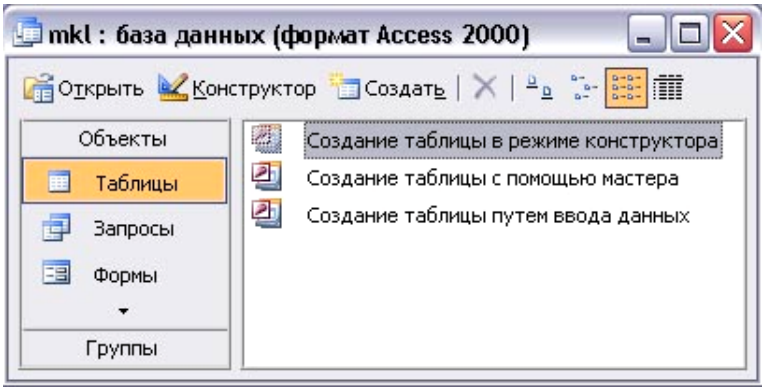
ШАГ 1. СОЗДАЕМ ПРОСТЕЙШУЮ БАЗУ ДАННЫХ

Прежде чем приступить к разработке системы управления базой данных (СУБД) следует эту базу создать. Воспользуемся для этой цели приложением Microsoft Office Access, так как офис установлен практически на любом компьютере и Access скорее всего был установлен по умолчанию.

В данном пособии будут рассмотрены возможности доступа к базам данных посредством только одной технологии – ADO (Active Data Objects), разработанной Microsoft. Это современная библиотека, прежде всего, позволяет работать с локальными базами MS Access и клиент-серверными MS SQL Server. Изучение этой библиотеки позволит вам в дальнейшем без затруднений перейти к базам данных, построенным на основе иных технологий.

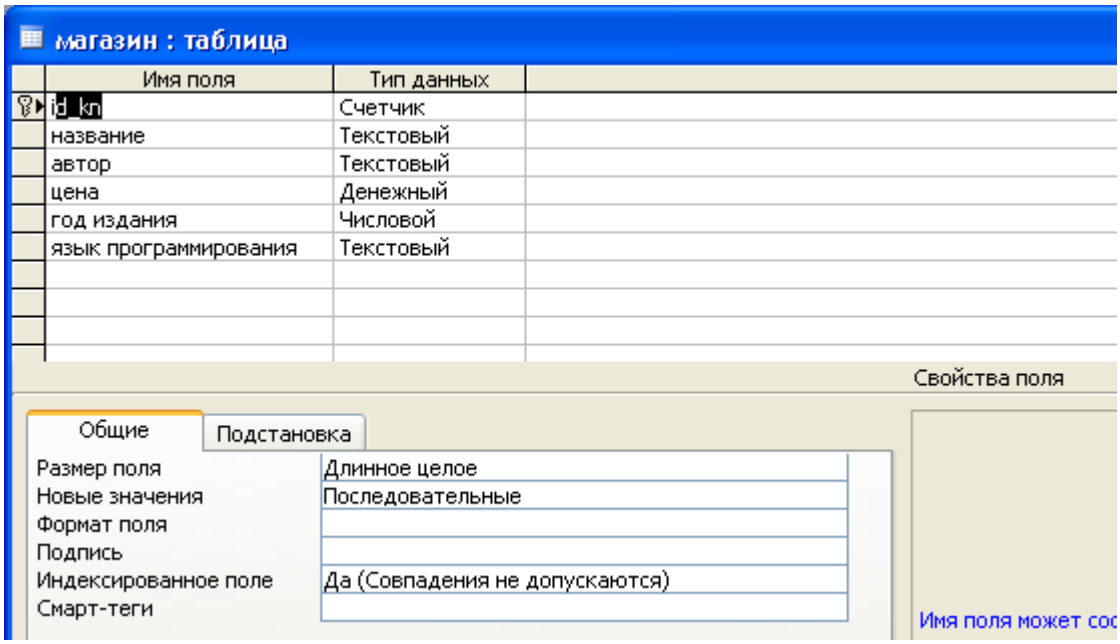
Создадим пока локальную базу данных магазина компьютерной литературы, состоящую из одной таблицы. Таблица как и двумерный массив состоит из столбцов и строк. Каждый столбец таблицы содержит представлен одним полем, например, названием книги или автором. Каждая строка таблицы содержит одну запись состоящую из нескольких полей, содержащих, например, название книги, автора, цену, год выпуска.

Запустите Microsoft Office Access. В меню нажмите `Файл/Создать` и далее в мастере выберите Новая база данных. Вам будет предложено выбрать место хранения базы и имя файла. Укажите путь к вашему первому будущему приложению (где в дальнейшем будете сохранять проект Delphi) и осмысленное имя для базы данных магазина компьютерной литературы, например, `mkl.mdb`. Откроется окно организации работы с базой данных (см. рис.).



Выберите двойным кликом мыши пункт «Создание таблицы в режиме конструктора» – откроется конструктор таблицы, в котором следует задать наименование полей таблицы и тип данных содержащихся в соответствующих полях.

Пример описания полей таблицы смотрите на рисунке ниже. Пусть в таблице будет шесть полей. В Access именам полей можно давать наименование, как на английском, так и на русском языках. Наименование полей №№2-5 очевидно, так же как и тип данных этих полей. Разберем поле №1. Наименование поля: `id_kn` – идентификатор книги. Это поле имеет для базы особое значение – это поле ключевое в таблице, оно несет неповторимый идентификатор записи. Установить опцию «Ключевое поле» можно через контекстное меню, возникающее при нажатии правой клавишей мыши на соответствующем поле в конструкторе таблицы. Сохраните таблицу, нажав на клавишу сохранения, вам будет предложено выбрать имя для таблицы – установите имя `магазин`.



Через меню `Вид` установите просмотр в `Режим таблицы`:

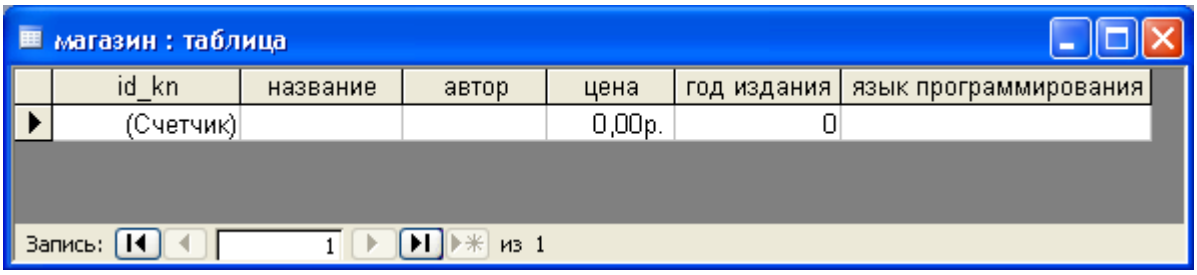


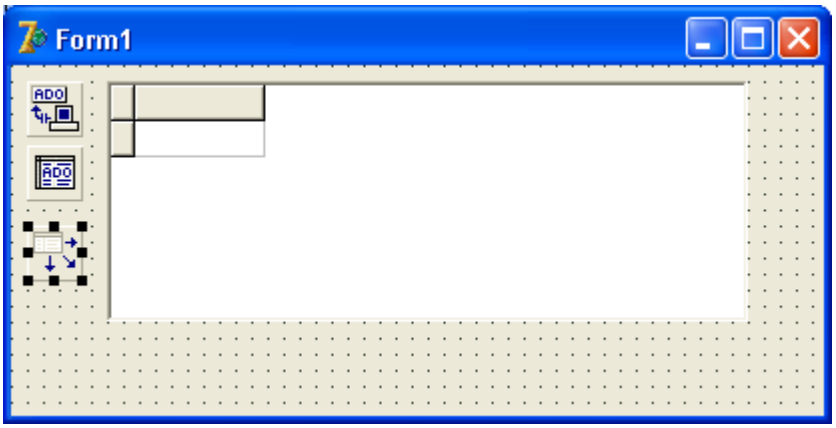
Таблица уже готова для заполнения, но мы сейчас не будем этим заниматься, так как основная наша цель состоит в изучении возможностей Delphi по управлению базами данных. Создадим приложение в Delphi и там уже и будем заниматься редактированием таблицы.

ШАГ 2. СОЗДАЕМ ПРОСТЕЙШЕЕ ПРИЛОЖЕНИЕ ДЛЯ УПРАВЛЕНИЯ БАЗОЙ ДАННЫХ

Простейшее приложение должно быть способно просмотреть содержимое базы данных (в нашем случае содержимое одной таблицы), кроме того должны быть функции исправления записей, их удаления и добавления. Аналогичную задачу можно, конечно, выполнить и без привлечения технологий обработки баз данных, но разработка такого приложения займет на два-три порядка больше времени.

Итак, запустите Delphi, создайте новое приложение и сохраните проект в папку, где находится файл базы данных. Пусть имя модуля будет magazin.pas, а имя проекта ProjectMagazin.dpr.

Теперь давайте определимся, какие компоненты с палитры необходимы для работы с базой данных. База данных состоит из таблиц, для просмотра которых необходим соответствующий визуальный компонент – DBGrid с вкладки Data Controls. Кроме того должны быть еще компоненты, которые обеспечивают связь приложения с местом расположения базы данных, распознают ее формат, делают выборку из определенной таблицы базы данных. Для этих целей используем следующие три компонента: ADOConnection и ADOTable с вкладки ADO и DataSource с вкладки Data Access.



Настроим свойства всех компонентов на форме.

1. ADOConnection1. Дважды кликните по компоненту (или в инспекторе объектов по строке свойства ConnectionString) – вам представится возможность ввести строку подключения (Use Connection String), запустите мастер нажатием клавиши Build и на вкладке «Поставщик данных» выберите драйвер подключения к базам данных Microsoft Jet OLE DB Provider. Нажмите «Далее» для перехода на вкладку «Подключение» и там, в строке «Выберите или введите имя базы данных», наберите имя файла – в нашем случае это mkl.mdb. Можно, конечно, нажать на клавишу рядом со строкой и непосредственно указать на файл, но, в этом случае, лучше сразу убрать путь к файлу, оставив только имя, чтобы при переносе приложения в другое место не возникло проблем с доступом к базе данных. Нажмите OK в мастере и OK на форме выбора строки подключения. Свойство LoginPromt переключите в False, чтобы каждый раз при подключении к базе данных к пользователю не было запроса о пароле.

2. ADOTable1. В свойстве Connection в выпадающем списке укажите на ADOConnection1, в свойстве TableName выберите таблицу (у нас она пока одна магазин). Свойство Active переведите в положение True (обратите внимание, что в дальнейшем при смене настроек вам часто придется это свойство возвращать в положение True). И, для удобства, переименуйте компонент в TableMagazin.

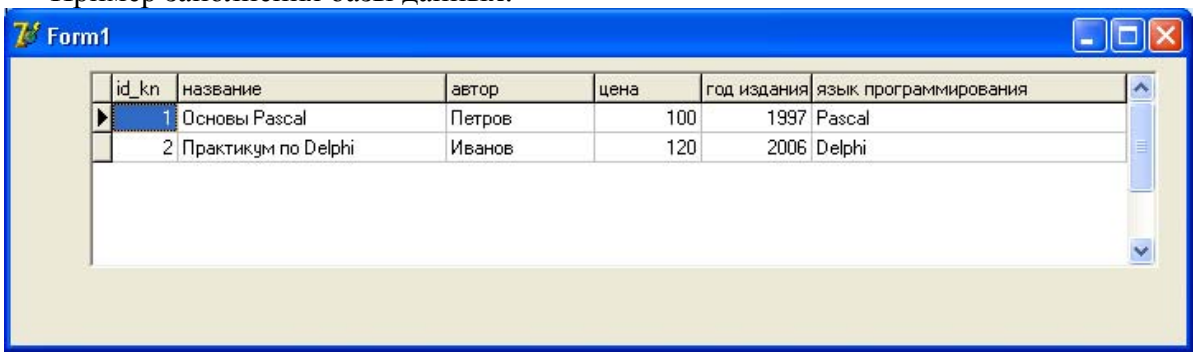
3. DataSource1. У этого компонента-посредника необходимо свойство DataSet установить в выпадающем списке на таблицу TableMagazin.

4. DBGrid1. Свяжем сетку с DBGrid1 с таблицей магазин из базы данных посредством DataSource1, установив в инспекторе объектов для свойства DataSource в выпадающем списке доступных компонентов DataSource1.

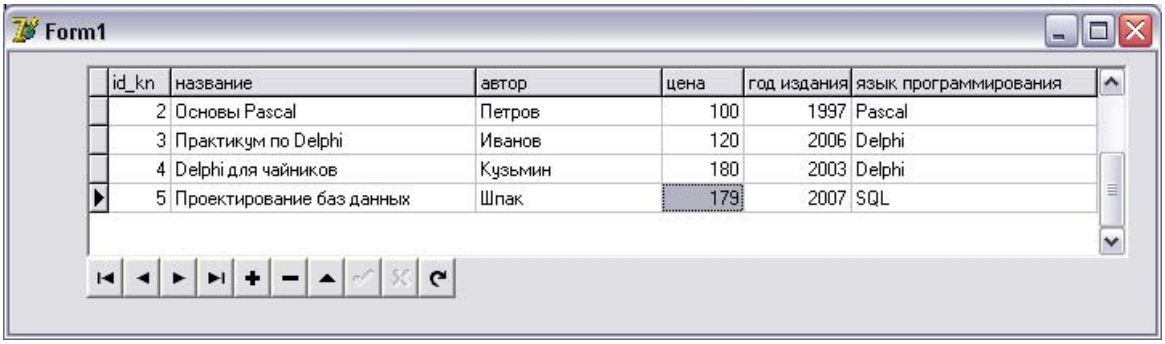
На этом создание простейшей базы данных закончено, а ведь мы не написали ни одной строчки кода. Если бы у вас уже был опыт работы с этими компонентами, то вы бы затратили менее минуты на разработку такой СУБД.

Запустите приложение клавишей F9 и поработайте над наполнением базы данных (клавиши управления: F2 – редактировать ячейку, Ins – добавить запись, Ctrl+Del – удалить запись). Закройте приложение и затем снова запустите – и вы убедитесь, что внесенные вами изменения сохранены.

Пример заполнения базы данных:

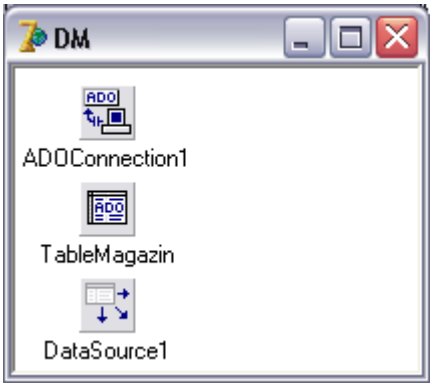


Не все предпочитают работать клавиатурой, если есть мышь. Поэтому иногда полезным может оказаться компонент `DBNavigator` с вкладки `Data Controls`. Разместите его для пробы (в дальнейшем его следует удалить) на форме по своему усмотрению и подсоедините используя посредник `DataSource1` – укажите на него в свойстве `DataSource`. По необходимости можно некоторые клавиши из панели управления базой данных отключить в свойстве `VisibleButtons`. И, хотя кнопки управления интуитивно понятны, имеется возможность снабдить их всплывающими подсказками, для чего установите свойство `ShowHint` в `True`, а текст подсказок можно установить/изменить в свойстве `Hints`. Возможный вид приложения после подключения компонента `DBNavigator` смотри на рисунке:



ОТСТУПЛЕНИЕ 1. МОДУЛЬ ДАННЫХ

Пока СУБД достаточно проста компоненты доступа к таблицам базы данных допустимо размещать непосредственно на главной форме. Но в дальнейшем вы будете усложнять приложение, будут появляться новые формы и новые компоненты доступа к базе данных, будет увеличиваться количество таблиц в базе данных, что будет усложнять работу. Разумно будет разместить все компоненты доступа к базе данных в одном месте – в `Delphi` для этих целей предназначена специальная форма: форма `Модуля Данных`. Эта форма не отображается во время работы приложения. Для создания такой формы следует пройти такой путь в меню: `Файл / Новый / Модуль данных` – у вас возникнет новая и пока пустая форма.



Заберите компоненты `ADOConnection1`, `TableMagazin` и `DataSource1` с главной формы приложения и вставьте их в форму модуля данных.

Форма модуля данных по умолчанию имеет не очень удобное (длинное) имя – `DataModule1`. В дальнейшем мы будем обращаться к нему, поэтому давайте сразу переименуем этот компонент в `DM` – лаконично, удобно помнить и набирать.

Модуль сохраните (`Файл / Сохранить как`) в

папке с базой данных (и с разрабатываемой программой) под именем UnitDM.pas.

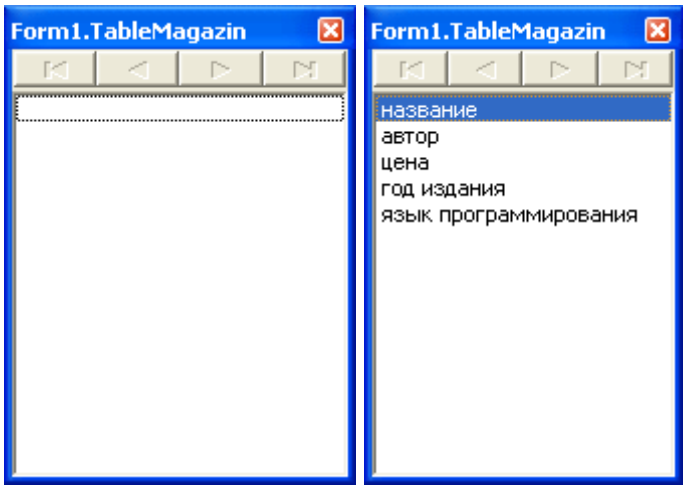
Теперь следует обновить связь компонента DBGrid1 с таблицей базы данных, так как компонент посредник DataSource1 перекочевал на другую форму. Первое что нужно сделать – указать главной форме (модулю magazin) на существование формы DM (модуля UnitDM). Для чего кликните главную форму, далее нажмите меню Файл / Использовать модуль, в открывшемся списке укажите на UnitDM (если вы не сохраняли модуль UnitDM.pas то он не будет отображаться в списке) и нажмите OK. Далее в инспекторе объектов для компонента DBGrid1 в свойстве DataSource укажите в выпадающем списке DM.DataSource1.

Впоследствии все компоненты доступа к базе данных следует размещать на форме DM, а к вновь создаваемым формам подключать модуль UnitDM.

Проверьте работоспособность приложения. И в следующий раз при проектировании СУБД в Delphi сразу создавайте Модуль данных.

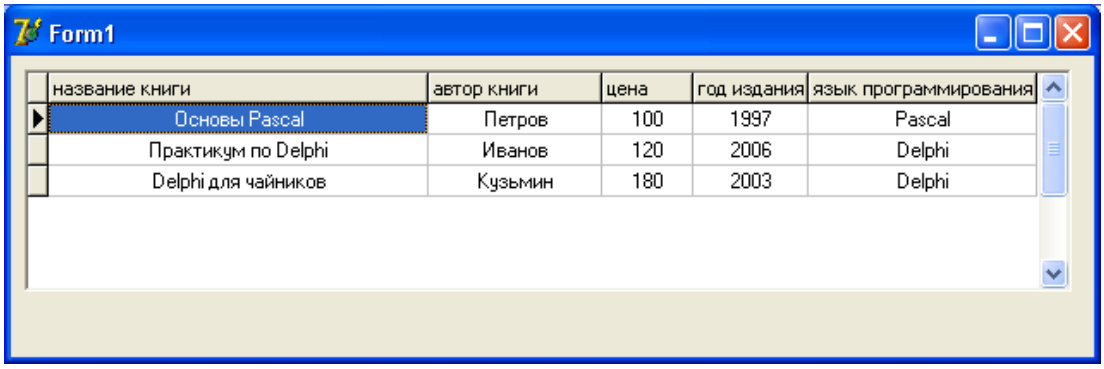
ШАГ 3. НАСТРОЙКА ВНЕШНЕГО ВИДА ОТОБРАЖЕНИЯ ТАБЛИЦЫ

Кликните дважды по компоненту TableMagazin – получите окно настройки полей таблицы Form1.TableMagazin. Кликните в нем правой клавишей мыши и в контекстном меню выберите опцию добавления всего содержимого таблицы – Add all fields.



Из полученного состояния можно удалить первое поле id_kn – идентификатор книги. Это поле является счетчиком, содержимое которого формируется автоматически и для пользователя не несет полезной информации. Оставшиеся поля можно настраивать по своему усмотрению.

Свойство Alignment определяет выравнивание содержимого полей, свойство DisplayWidth – отображаемую ширину столбцов, свойство DisplayLabel – надпись в титульной строке. Этого пока достаточно для первоначальной настройки.

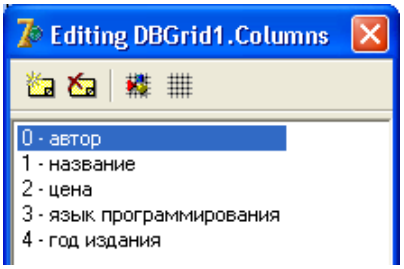


Дальнейшие настройки можно произвести в компоненте DBGrid1.

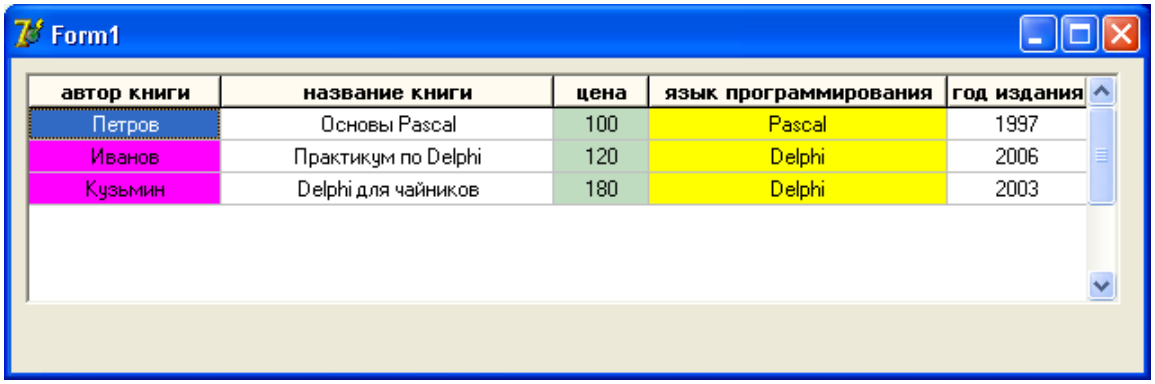
Для начала настройте через Инспектор Объектов по своему усмотрению шрифт титульной строки в свойстве TitleFont и установите приемлемый цвет в свойстве FixedColor. Далее отключите dgIndikator в свойстве Options.

Существует возможность настраивать вид столбцов по отдельности. Для этого кликните дважды по компоненту DBGrid1 и получите окно настройки полей таблицы. Клик-

ните правой клавишей мыши по окну настройке и возникшем контекстном меню выберите опцию добавления всех полей. Затем расставьте их по своему усмотрению, передвигая левой клавишей мыши вверх-вниз. Эта расстановка никак не скажется на исходной таблице в базе данных. Для каждого поля в инспекторе объектов можно установить свой цвет, выравнивание и другие параметры.



Пример оформления внешнего вида таблицы:



К текущему моменту еще не было написано ни строчки программного кода. Давайте все-таки уже попробуем управлять отображением таблицы динамически – через программный код. Предоставим пользователю возможность организовать сортировку таблицы по выбранному столбцу и выбрать направление сортировки.

Будем использовать достаточно простой метод, который имеется у компонента `TableMagazin`. Для организации сортировки достаточно параметру `Sort` присвоить имя столбца по которому проводится сортировка.

Разместите на форме компонент `RadioGroup1` и добавьте две строки в его свойство `Items`: автор и цена (по этим столбцам будем проводить сортировку). Дважды кликните по компоненту `RadioGroup1` и добавьте в обработчик события код:

```
procedure TForm1.RadioGroup1Click(Sender: TObject);
begin
  with RadioGroup1 do
    DM.TableMagazin.Sort:=Items[ItemIndex];
end;
```

Параметр `ItemIndex` содержит номер нажатой кнопки в `RadioGroup1` (счет ведется с нуля), а `Items` – содержит соответствующие строки (вы их сами набирали).

Теперь организуем возможность выбора направления сортировки. Для чего разместите на форме компонент `RadioGroup2` и добавьте две строки в его свойство `Items`: `по возрастанию` и `по убыванию`. В инспекторе объектов для компонента `RadioGroup2` в качестве обработчика события `OnClick` в выпадающем списке укажите обработчик `RadioGroup1Click`, то есть один обработчик будет обрабатывать события при выборе пользователем как направления сортировки так и столбца для сортировки. Это обстоятельство следует учесть в обработчике события:

```
procedure TForm1.RadioGroup1Click(Sender: TObject);
var ns: string;
begin
```

```
  case RadioGroup2.ItemIndex of
    0: ns:=' ASC';
    1: ns:=' DESC';
  end;
  with RadioGroup1 do
    DM.TableMagazin.Sort:=Items[ItemIndex]+ns;
end;
```

В этом коде уже используются элементы языка SQL (англ. Structured Query Language – «язык структурированных запросов») – универсальный компьютерный язык, применяемый для создания, модификации и управления информацией в базах данных. Данный язык не является целью при освоении материала описанного в этом пособии, но по мере необходимости он будет использоваться и будут приводиться соответствующие комментарии для лучшего понимания материала.

Тут следует упомянуть, что ASC (сокращение от ASCENDING – гл. подниматься) и DESC (от DESCENDING – гл. спускаться) инструкции, указывающие направление сортировки. Однако на практике слово ASC обычно не применяется, поскольку такой порядок используется по умолчанию (он предполагается, если не указано ни ASC, ни DESC).

Другой способ расширить функционал СУБД – обеспечить фильтрацию данных по выбору пользователя. Давайте предоставим пользователю возможность из таблицы отбирать книги по году издания. У компонента `TableMagazin` есть строковый параметр `Filter` – там укажем команду, определяющую условие фильтрации.

Разместите на форме компонент `GroupBox1` и добавьте в него компоненты `Label1`, `MaskEdit1` и `Button1` (см на рис. далее). `MaskEdit1` настройте так, чтобы пользователь не мог ввести буквы или символы, а только четырехзначное число (используя свойство EditMask = 9999 и Text = 2000 – это для образца).

В обработчик события нажатия кнопки добавьте код:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  with DM.TableMagazin do
  begin
    Filter:='[год издания]>='+MaskEdit1.Text;
    Filtered:=True;
  end;
end;
```

Пояснения к коду. Поле `год издания` следует брать в прямоугольные скобки так как оно состоит из двух слов (таковы правила `SQL`). Из компонента `MaskEdit` берется год

для проверки условия. Свойство Filtred (типа Boolean) переводится в состояние True, для того чтобы сработал механизм фильтрации.

Пример оформления формы для организации фильтрации по году издания:

ШАГ 4. ИЗБИРАТЕЛЬНОЕ ОТОБРАЖЕНИЕ ДАННЫХ

Для управления отображениям информации из таблиц базы данных удобнее использовать компонент ADOQuery с вкладки ADO. Его свойства аналогичны компоненту ADOTable, но дополнительно существует возможность использовать язык запросов SQL.

Создайте новую форму и сохраните новый модуль как Otbor.pas, разместите на новой форме DBGrid1. На форме Модуля Данных разместите компонент ADOQuery1 и переименуйте его в QueryMagazin. Также разместите компонент DataSource2 с вкладки Data Access. Компонент QueryMagazin подключите к ADOConnection1, а у компонента DataSource2 свойство DataSet установите в QueryMagazin. Для формы Form2 укажите использовать модуль UnitDM. Для сетки DBGrid1 в инспекторе объектов укажите компонент посредник DM.DataSource2. Аналогичные действия вы уже ранее. На этом приготовления закончены, осталось только указать параметры отбора информации из базы данных. Для этих целей используется свойство SQL компонента QueryMagazin.

Свойство SQL представляет собой список строк, куда можно помещать инструкции SQL. Давайте попробуем создать очень простую инструкцию для отображения всей информации из таблицы магазин.

Изначально список строк SQL пустой, добавьте туда строку:

```
SELECT * FROM магазин
```

На языке запросов это означает: выбрать все поля из таблицы магазин.

Чтобы такой отбор был реализован активизируйте QueryMagazin – переведите свойство Active из False в True. Посмотрите на результат – не запуская приложение уже можно видеть что отбор произошел.

Давайте немного усложним запрос:

```
SELECT * FROM магазин WHERE цена<=150
```

На языке запросов это означает: выбрать все поля из таблицы магазин, такие чтобы значение поля цена было не более 150. Чтобы такой отбор был реализован активизируйте QueryMagazin.

Учитывая, что к свойствам компонентов можно обращаться во время выполнения программы, то менять запрос можно динамически – по требованию пользователя. Для этого нужно организовать обращение к свойству SQL.

Давайте разберем один несложный пример.

Для начала сделаем так, чтобы форма Form2 была доступна из формы Form1. Кликните на первой форме и через меню укажите использовать модуль второй формы. Далее разместите на первой форме кнопку Button2 для вызова второй формы и в обработчик нажатия кнопки добавьте код: Form2.ShowModal. Запустите приложение и убедитесь, что на второй форме происходит отображение только тех книг, которые не дороже 150 руб.

На первой форме рядом с кнопкой Button2 разместите Label1 с текстом «книги не дороже:» и Edit1 с текстом «150» (рис. см. ниже). В обработчик нажатия кнопки добавьте код:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  with DM.QueryMagazin do
  begin
    Active:=False;
    SQL.Clear;
    SQL.Add('SELECT * FROM магазин WHERE цена<=150');
    Active:=True;
  end;
  Form2.ShowModal;
end;
```

Метод SQL.Clear очищает список строк, а процедура SQL.Add добавляет строку.

Апробируйте программу – эффект должен быть тот же: происходит отображение только тех книг, которые не дороже 150 руб.

Теперь попробуем менять условие отбора динамически, то есть во время выполнения программы. Сейчас цена установлена жестко – 150. А мы сделаем так, чтобы в запросе цена бралась из текстового поля Edit1. Измените код в обработчике нажатия кнопки:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  with DM.QueryMagazin do
  begin
    Active:=False;
    SQL.Clear;
    SQL.Add('SELECT * FROM магазин WHERE цена<='+Edit1.Text);
    Active:=True;
  end;
  Form2.ShowModal;
end;
```

Апробируйте работу программы с изменением значения Edit1.Text.

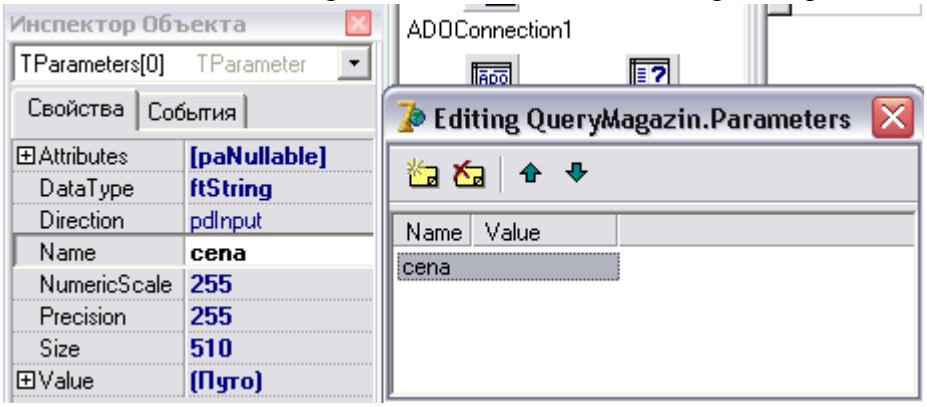
Полезным для вас будет узнать еще об одном удобном подходе к организации запросов. В дальнейшем запросы будут становиться все сложнее и объемнее, поэтому стирать полностью список строк SQL и затем его восстанавливать не самый оптимальный путь. Можно ведь не удалять, а исправлять строки:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  with DM.QueryMagazin do
  begin
    Active:=False;
```

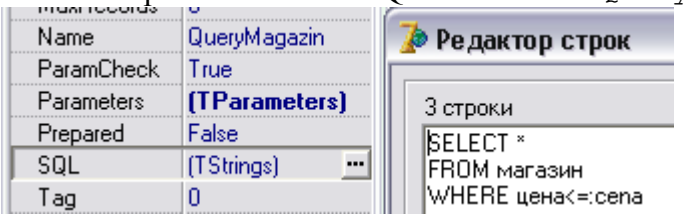
```
    SQL.Strings[0]:='SELECT * FROM магазин WHERE цена<='+Edit1.Text;
    Active:=True;
  end;
  Form2.ShowModal;
end;
```

И, наконец, рассмотрим третий способ. Будем исправлять не текст запроса, а используемые в нем переменные. Для чего их необходимо сначала назначить.

Кликните один раз на компоненте `QueryMagazin` и в Инспекторе Объектов найдите свойство `Parameters`. Кликните дважды на строке этого свойства для открытия редактора параметров. В редакторе добавьте один параметр и дайте ему в Инспекторе Объектов имя `cena`, а также укажите тип параметра – `ftString`. В поле Value можно назначить значение этого параметра по умолчанию.



Теперь изменим текст запроса в свойстве SQL компонента `QueryMagazin:`.



Как видите запрос можно писать в несколько строк, а переменные обозначаются символом «:» (двоеточие). Теперь формат запроса будет стабильным, но значение переменной в запросе будет меняться:

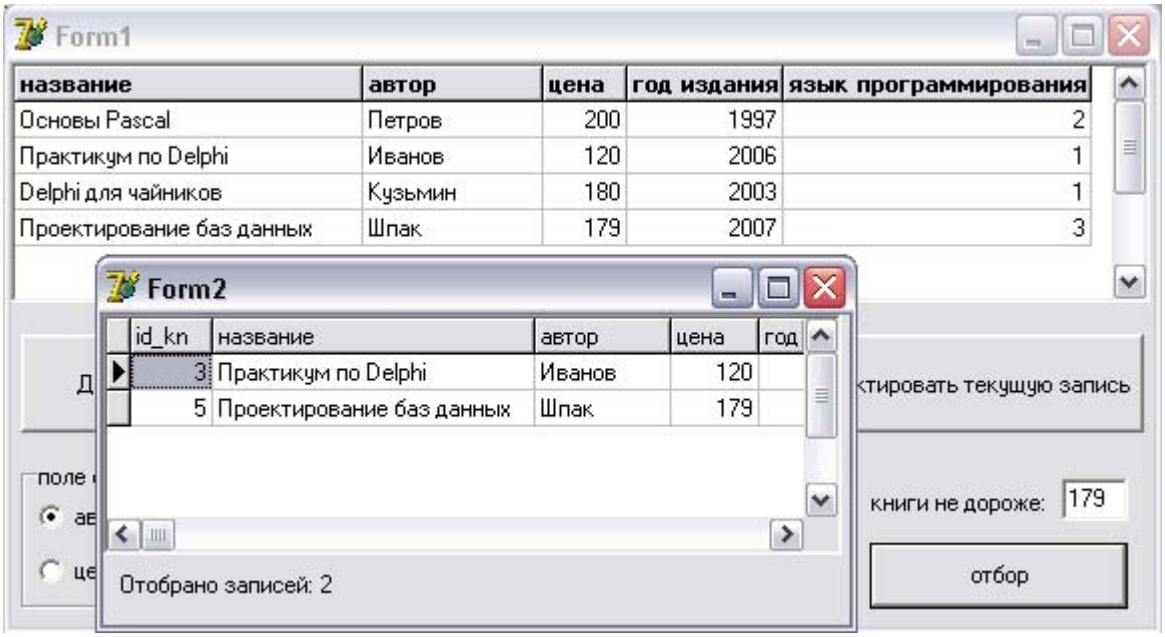
```
procedure TForm1.Button1Click(Sender: TObject);
begin
  with DM.QueryMagazin do
  begin
    Active:=False;
    Parameters.ParamByName('cena').Value:=Edit1.Text;
    Active:=True;
  end;
  Form2.ShowModal;
end;
```

Для удобства пользователя можно на форму `Form2` поместить одну метку `Label1` и в ней отображать общее количество отобранных по критерию (`WHERE цена<=:cena`) из основной таблицы записей (количество записей хранится в свойстве `RecordCount`):

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  with DM.QueryMagazin do
```

```
  begin
    Active:=False;
    Parameters.ParamByName('cena').Value:=Edit1.Text;
    Active:=True;
    Form2.Label1.Caption:='Отобрано записей: '+IntToStr(RecordCount);
  end;
  Form2.ShowModal;
end;
```

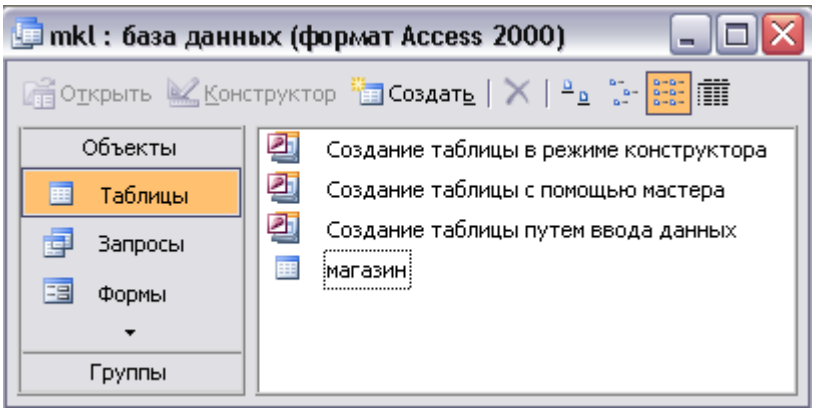
Пример выполнения запроса:



ОТСТУПЛЕНИЕ 2. СОКРАЩЕНИЕ ИЗБЫТОЧНОЙ ИНФОРМАЦИИ

Хранение одинаковой (повторяющейся) информации в разных строках таблицы неоправданно. Например, в нашем случае повторяется информация про языки программирования. Для устранения такой избыточности принято поступать так: создают отдельную таблицу для хранения повторяющейся информации (в нашем случае языки программирования), а в основной таблице вместо повторяющейся информации (названия языков программирования) размещаются только порядковые номера её расположения во вспомогательной таблице. Рассмотрим этот приём на примере.

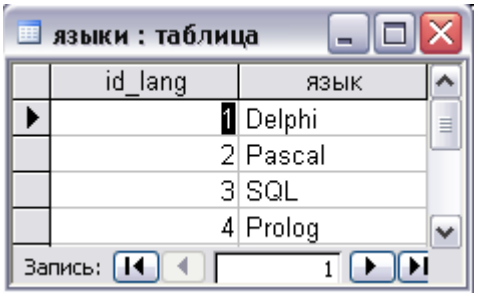
Откройте нашу базу данных `mkl.mdb` в `Microsoft Access`. В ней пока существует только одна таблица. В конструкторе базы данных нажмите `Создать` и укажите `Режим Конструктора`.



Далее определите два поля в таблице: `id_lang` – идентификатор языка и `язык` – название языка программирования.

Поле id_lang назначьте ключевым и сохраните таблицу, дав ей имя языки. В конструкторе базы данных станет доступной еще одна таблица – языки. Давайте её отредактируем, а точнее добавим названия языков программирования.

Кликните дважды в конструкторе базы данных по имени таблицы. У вас откроется таблица в режиме редактирования. Поле id_lang заполняется автоматически при добавлении новой записи так как мы установили его как Счетчик, а поле язык заполните сами по образцу



В таблице магазин тоже необходимо сделать изменения: удалите содержимое поля язык программирования во всех строках и через конструктор смените тип данных этого поля на числовой.

Теперь следует внести изменения в структуру приложения, так как формат полей таблицы изменился.

Кликните дважды на компонент TableMagazin и в окне настройки полей таблицы удалите поле язык программирования и добавьте его снова, затем активизируйте компонент TableMagazin. Всё – изменения вступили в силу, можно посмотреть на результат.

Пока поля для языка программирования заполним соответствующими цифрами, что несколько неудобно. Но в дальнейшем мы разработаем и оформим интерфейса пользователя, что избавит от необходимости подсматривать в таблицу языки.

Эффект сокращения избыточности информации пока не заметен и, возможно, малопонятен. Мы из одной таблицы сделали две. При увеличении числа книг будет расти только первая таблица, вторая будет оставаться без изменений. В первой таблице вместо названий языков будут стоять цифры (ссылки на строки таблицы языки). Но мы организуем так, чтобы пользователь мог видеть содержимое этих двух таблиц в удобной для себя форме.

Один из подходов основан на использовании языка SQL, который позволяет делать выборку из нескольких таблиц в одну. Рассмотрим пример.

Давайте для сравнения разместим такую объединенную таблицу прямо на главной форме – разместите на форме компонент DBGrid2, а в форме Модуля Данных компоненты ADOQuery1 (переименуйте в QueryOtbor) и DataSource3. QueryOtbor свяжите с ADOConnection1, DataSource3 с QueryOtbor, а DBGrid2 с DataSource3.

После того как все компоненты связаны можно определиться с запросом к базе данных, состоящей из двух таблиц. Для этого не обязательно знать язык SQL – достаточно грамотно воспользоваться возможностями Microsoft Access, в котором есть конструктор запросов.

Переходим в Microsoft Access, закрываем все открытые таблицы и в конструкторе базы данных нажимаем на кнопку Запросы, далее выбираем опцию Создание запроса в режиме конструктора – возникает окно добавления таблиц. Добавляем обе таблицы и закрываем это окно.

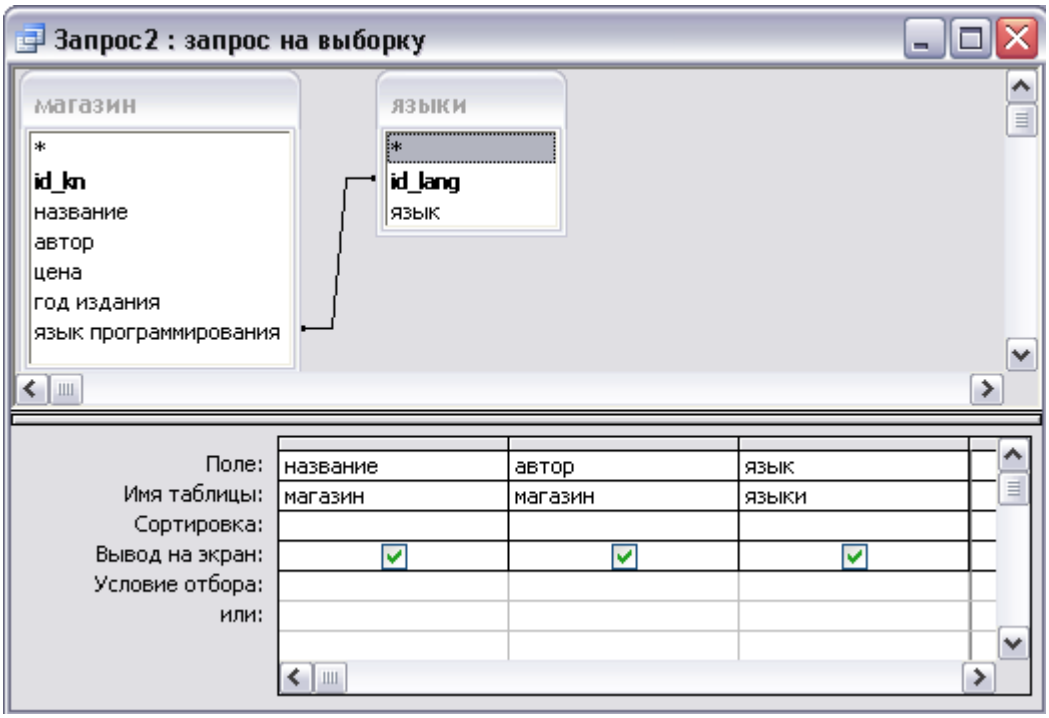
Установим связь между таблицами по полю язык программирования из таблицы магазин и по полю id_lang из таблицы языки. Для чего нажмем левой клавишей мыши на поле язык программирования в таблице магазин и удерживая его перенесем на поле id_lang из таблицы языки.

В конструкторе запроса (см. рис. ниже):

– в первом столбце выбираем Имя таблицы – магазин, а Поле – название;

– во втором столбце выбираем Имя таблицы – магазин, а Поле – автор;

– в третьем столбце выбираем Имя таблицы – языки, а Поле – язык.



На этом формирование запроса закончено. Можно его апробировать, выбрав в меню `Запрос / Запуск` – на что получите искомый результат. Но нас интересует не сколько результат, сколько текст запроса на языке `SQL`. Его можно просмотреть и скопировать выбрав в меню `Вид / Режим SQL`. Скопируйте весь текст запроса:

```
SELECT магазин.название, магазин.автор, языки.язык
FROM магазин INNER JOIN языки ON магазин.[язык программирования] = языки.id_lang
```

и вставьте его в свойство `SQL` компонента `QueryMagazin`. Активизируйте компонент (свойство `Active=True`). Результат можно увидеть в компоненте `DBGrid2`.

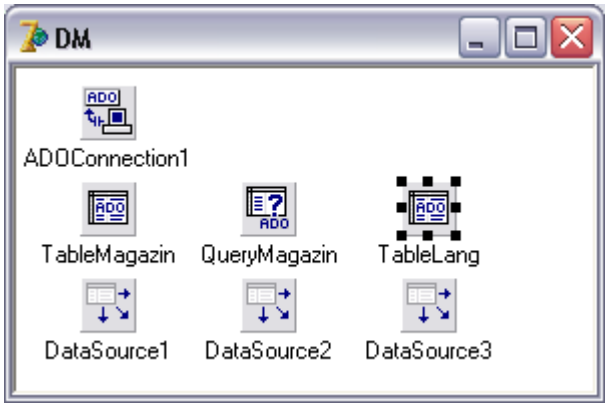
Рассмотрим кратко текст запроса. В первой строке после ключевого слова SELECT перечислены поля, которые будут отбираться из соответствующих таблиц. Во второй строке указано что данные берутся из (FROM) таблицы `магазин` и объединяются с таблицей `языки`, но выбираются только те строки, для которых истинно условие – магазин.[язык программирования] = языки.id_lang.

Итак, даже не зная языка `SQL`, используя конструктор запросов `Microsoft Access`, можно сформировать запрос различной сложности. Попробуйте сами добавить в текст запроса сортировку по полю `автор` по убыванию.

ШАГ 5. РАЗРАБОТКА ИНТЕРФЕЙСА ПОЛЬЗОВАТЕЛЯ

Хорошее приложение должно иметь удобный интерфейс пользователя.

Давайте рассмотрим один несложный пример организации работы пользователя с нашей базой данных.

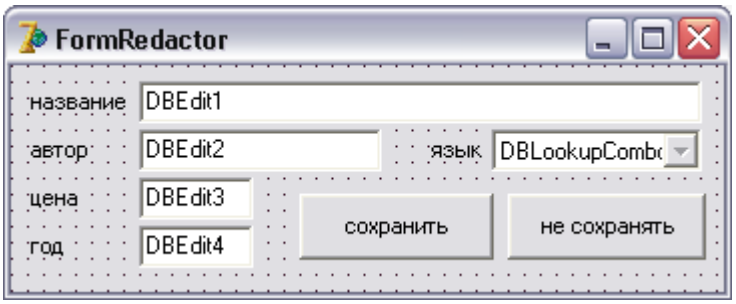


Чтобы не загромождать приложение удалите с главной формы компонент DBGrid2, а с формы Модуля данных компоненты QueryOtbor и DataSource3.

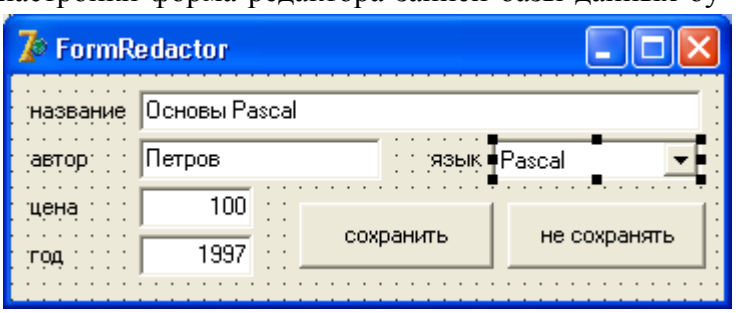
В Модуль Данных добавьте ADOTable1, переименуйте в TableLang и соедините с ADOConnection1; добавьте также и DataSource3 и соедините с TableLang.

У компонента DBGrid1 через инспектор объектов в множественном свойстве Options свойство dgEditing переведите в False, для того чтобы пользователь не мог непосредственно редактировать содержимое таблицы. Мы сделаем доступ к ячейкам таблицы через специальную форму.

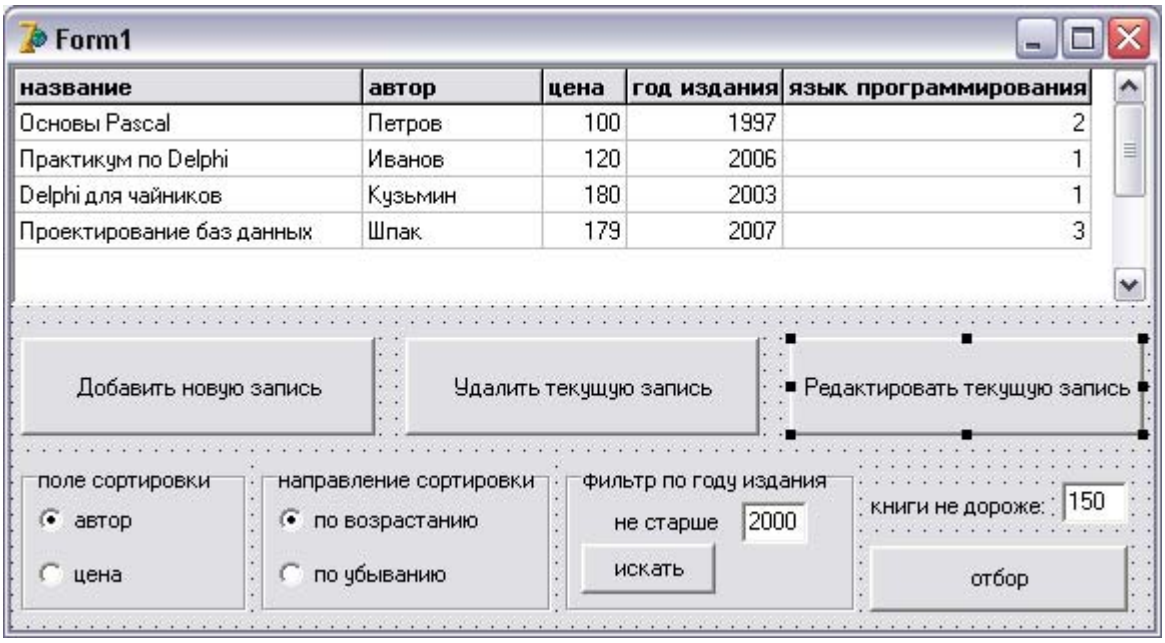
Создайте новую форму, дайте ей имя FormRedactor и сохраните модуль под именем Redactor.pas. Подключите к новой форме Модуль Данных UnitDM. Разместите на форме пять меток Label, четыре компонента DBEdit и один DBLookupComboBox1 с вкладки Data Controls, две кнопки («сохранить», «не сохранять»).



Подключим компоненты к базе данных. Для всех DBEdit в качестве компонента посредника укажите DM.DataSource1, а в свойстве DataField в ниспадающем списке укажите соответствующее поле для отображения. Несколько сложнее настроить компонент DBLookupComboBox1 – ведь он должен одновременно обращаться к двум таблицам: из таблицы магазин компонент будет выбирать номер языка программирования, а из таблицы языки его название. Для начала в свойстве DataSource укажите компонент посредник DM.DataSource1, связанный с основной таблицей магазин, которую и будем редактировать с помощью разрабатываемой формы. А в свойстве DataField укажите поле для редактирования – язык программирования. Далее надо сделать так, чтобы в ниспадающем списке компонента DBLookupComboBox1 отображались не номера языков программирования, а их названия. В свойстве ListSource укажите DM.DataSource3 – там будут выбираться названия языков, а в свойстве ListField укажите поле этой таблицы, из которого будут выбираться названия языков – язык. Но в основной таблице магазин следует сохранять не название, а номер языка программирования, поэтому в свойстве KeyField укажите id_lang. После настройки форма редактора записи базы данных будет иметь такой вид:



Давайте теперь подключим форму редактора к главной форме: переключитесь на главную форму, в меню Файл / Использовать модуль укажите Redactor. На главной форме разместите три кнопки: «Добавить новую запись», «Удалить текущую запись», «Редактировать текущую запись».



Начнем с кнопки «Редактировать текущую запись». В обработчик события OnClick этой кнопки добавьте одну строку кода – FormRedactor.ShowModal. Этим вы обеспечиваете запуск в модальном режиме формы редактора, которая будет отображать поля текущей выделенной записи и у пользователя не будет возможности перейти к другой записи базы данных пока он не закроет форму редактора. Апробируйте приложение – устанавливайте курсор на разные позиции в основной таблице и вызывайте редактор. Он должен отображать текущую запись в удобной для пользователя форме. Однако кнопки «сохранить» и «не сохранять» пока не действуют. Устраним эту недоработку.

Для кнопки сохранить можно использовать такой подход – если содержимое полей таблицы менялось, то сохрани их методом Post, после чего закрой форму редактора:

```
procedure TFormRedactor.Button1Click(Sender: TObject);
begin
  if DM.TableMagazin.Modified then DM.TableMagazin.Post;
  Close;
end;
```

Если пользователь выбрал кнопку «не сохранять», то можно использовать специальный метод выхода без сохранения – Cancel:

```
procedure TFormRedactor.Button2Click(Sender: TObject);
begin
  DM.TableMagazin.Cancel;
  Close;
end;
```

Апробируйте работу кнопок, внеся изменения в базу данных.

Осталось подключить кнопки добавления и удаления записи.

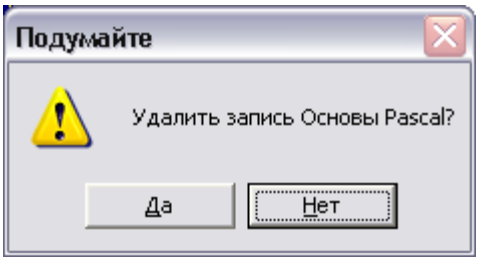
Что нужно для реализации действия добавления записи. В нашем случае две операции: вставить в таблицу пустую запись и открыть на ней форму редактора:

```
procedure TForm1.BitBtn1Click(Sender: TObject);
begin
  DM.TableMagazin.Insert;
  FormRedactor.ShowModal;
end;
```

Для удаления текущей записи из таблицы достаточно использовать специальный метод Delete, но для безопасности лучше предварить его вопросом к пользователю о подтверждении операции удаления. И было бы неплохо, чтобы программа показывала, что именно предстоит удалить, а не просто спрашивала о подтверждении операции удаления. Это можно сделать с помощью одного из видов модальных диалогов, например, MessageBox. Перенести информацию о текущей записи из главной формы можно с помощью обращения к модулю данных. В модуле данных дважды кликните по компоненту TableMagazin и в открывшемся окне настройки полей таблицы выделите поле название. В инспекторе объектов вы увидите имя, данное этому компоненту – TableMagazinDSDesigner. К нему как к строке мы и будем обращаться:

```
procedure TForm1.BitBtn2Click(Sender: TObject);
begin
  if MessageBox(0, PChar('Удалить запись '+DM.TableMagazinDSDesigner.AsString
+'?'), 'Подумайте', MB_YESNO or MB_ICONWARNING or MB_DEFBUTTON2)=id_yes
  then DM.TableMagazin.Delete;
end;
```

Примерно такой результат можно получить используя модальный диалог:



==========

На этом завершаем поверхностное знакомство с инструментами Delphi для работы с базами данных. Напомню, что мы рассмотрели только методику разработки СУБД с использованием компонентов ADO (базы данных Access и MS SQL Server). Кроме этого Delphi позволяет использовать иные технологии: BDE – для доступа к базам данных Paradox и dBase, DBExpress – для доступа к базам данных Oracle, DB2, MySQL). За рамками рассмотрения остались вопросы оформления отчетной документации по базам данных. Печать и сохранение данных можно оформить через компоненты QuickReports или Rave Reports, через вывод данных в Excel или Word. Хочу также обратить ваше внимание, что написать серьезную СУБД без понимания языка SQL вряд ли получится. Среда визуального проектирования и событийного программирования Delphi предоставляет широкие возможности для разработки приложений по управлению базами данных, но разработка СУБД требует познаний о разнообразных информационных технологиях и успеха можно достичь только самостоятельным анализом и освоением разрозненного материала.