Технологии баз данных

Лабораторная работа №1

Задача: Создать базу данных писателей, поэтов (или композиторов) с их произведениями. Задачу выполнить в виде программы в среде Lazarus, тип базы – dBase. В программу включить поиск произведения по фрагменту названия.

Для выполнения задачи используем следующие взаимосвязанные таблицы.

Таблица "Писатели (авторы)" – главная

Writers

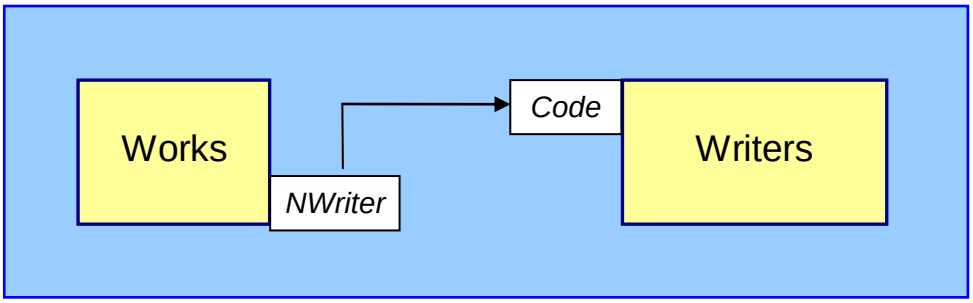
Описание поля	Наименование	Тип	Размер	Индекс
Код	Code	Integer		iCode
Полное имя	FullName	String	100	
Годы жизни (г.рожд.)	BYear	Integer		
(год смерти)	DYear	Integer		
Страна	State	String	40	

Таблица "Произведения"

Works

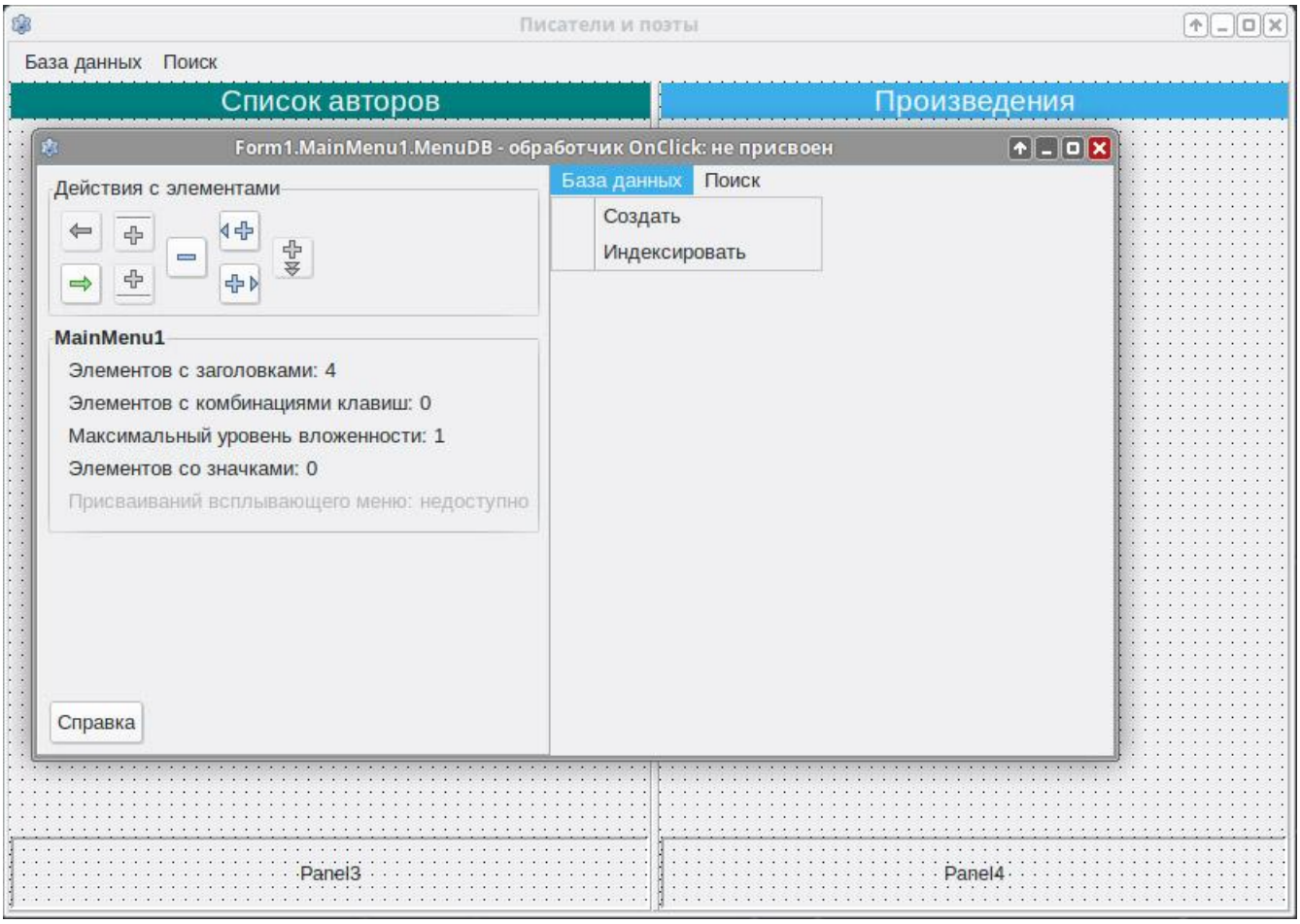
Описание поля	Наименование	Тип	Размер	Индекс
Код	NWork	Integer		iWork
Автор	NWriter	Integer		iWriter
Название	WorkName	String	120	
Год создания	CYear	Integer		

Связи между таблицами



Выполнение проекта

Интерфейс программы удобно выполнить в виде двух списков на форме: слева – таблица писателей, справа – произведения выбранного писателя. Для работы с этими списками воспользуемся меню – как показано на рисунке ниже.



При этом пунктам меню рекомендуется давать осмысленные названия, например, MenuDB, MenuDBCreate, MenuIndex, MenuSearch.

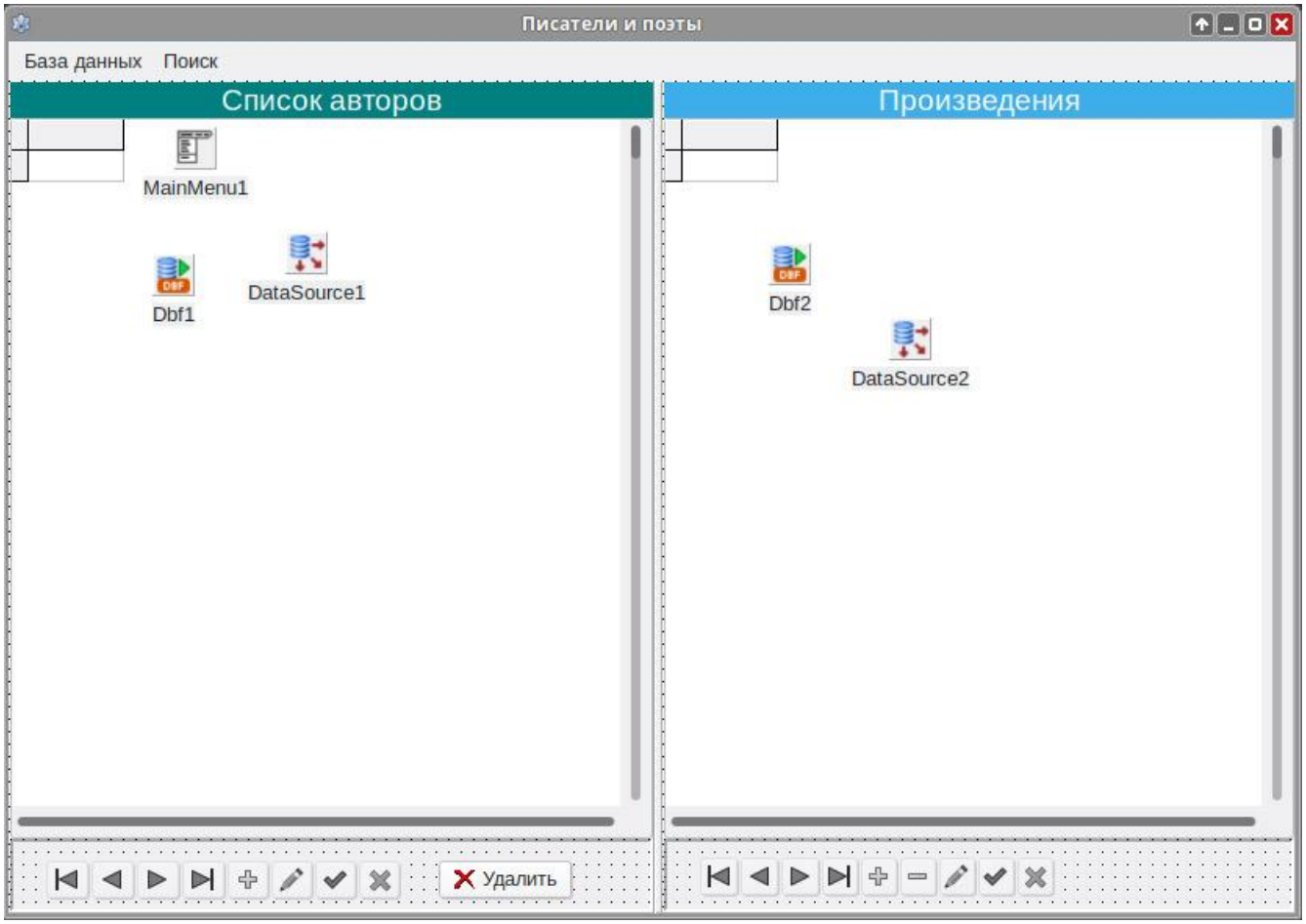
Чтобы удобно было работать с таблицами, поместим на форму сначала две панели, разделенные сплиттером. В левой панели зададим в *Инспекторе Объектов* свойство Align = alLeft, чтобы она всегда занимала всю левую часть. Затем помещаем Splitter, которым можно изменять ширину панелей. Затем – правая панель. Для неё укажем Align = alClient, чтобы она заняла всю оставшуюся часть формы.

Далее в верхней части левой панели размещаем Label со словами «Список писателей», а на правой панели – Label со словом «Произведения». У этих элементов свойство Align = alTop.

В нижнюю часть левой панели помещаем небольшую панель для кнопок, Align = alBottom. Аналогично поступаем и с правой панелью.

Остальную часть панели Panel1 заполняем компонентом DBGrid1, а Panel2 – DBGrid2. Здесь Align = alClient.

Добавим компоненты, необходимые для работы с базой данных. На вкладке Data Access находим компоненты Dbf и DataSource. Помещаем на форму 2 комплекта (для двух связанных таблиц). На вкладке Data Controls находим DBNavigator и помещаем 2 компонента на нижние панели. В Инспекторе объектов настроим у них свойство VisibleButtons: уберём кнопку nbRefresh, а в левом компоненте ещё и nbDelete – удаление записей из левой таблицы сделаем через отдельную кнопку – можно взять для этого BitBtn, тогда можно её снабдить значком удаления.



Кроме того, настроим связи между компонентами в Инспекторе объектов:

```
DBGrid1.DataSource = DataSource1
DBGrid2.DataSource = DataSource2
DataSource1.DataSet = Dbf1
DataSource2.DataSet = Dbf2
DBNavigator1.DataSource = DataSource1
DBNavigator2.DataSource = DataSource2
```

Начальный вариант интерфейса готов. Можно заняться кодом программы.

Активируем событие Form1.OnCreate и напишем следующий его обработчик:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  BaseDir:= ExtractFilePath(Application.ExeName) + 'Base';
  FWriters:= BaseDir + d + 'Writers.dbf';
  FWorks:= BaseDir + d + 'Works.dbf';
  if not DirectoryExists(BaseDir) then CreateDir(BaseDir)
  else
    if FileExists(FWriters) and FileExists(FWorks) then begin
      Dbf1.TableName:= FWriters;
      Dbf2.TableName:= FWorks;
       // проверка и замена код. страницы, если не соответствует Linux (88)
      SetLangCode(FWriters, 88);
      SetLangCode(FWorks, 88);

      Dbf2.MasterSource:= nil; // оключаем связь между таблицами
      Dbf2.IndexName:= 'iWork'; // чтобы перейти на главный индекс
      Dbf2.MasterFields:= '';
      Dbf1.Open;
      Dbf2.Open;
      Dbf1.Last;
      if Dbf1.RecordCount > 0 then
        IDWriter:= Dbf1.Fields[0].AsInteger;
      Dbf2.Last;
      if Dbf2.RecordCount > 0 then
        IDWork:= Dbf2.Fields[0].AsInteger;
      Dbf1.Close;
      Dbf2.Close;
      Dbf2.MasterSource:= DataSource1; // возвращаем связь между таблицами
      Dbf2.IndexName:= 'iWriter';
      Dbf2.MasterFields:= 'CODE';
      Dbf1.Open;
      Dbf2.Open;
    end
end;
```

Здесь используется константа d, её объявляем в разделе Interface:

```
const
{$IFDEF linux}
   d = '/';
{$ENDIF}
{$IFDEF windows}
   d = '\';
{$ENDIF}
```

Это разделитель для каталогов, который пишется по-разному для разных операционных систем.

Объявим глобальные переменные (в разделе interface):

```
var
  Form1: TForm1;
  BaseDir, FWriters, FWorks: string;
  IDWriter: integer = 0;
  IDWork: integer = 0;
```

И запишем в начале раздела implementation свою специальную функцию SetLangCode, с помощью которой будем контролировать кодовую страницу в файле типа *.dbf. Дело в том, что Windows ставит файлу при создании код 38 для русского алфавита, а Linux такой файл считает ReadOnly, т.е. читает, но корректировать не даёт. Linux файлу при создании ставит код 88.

```
implementation

{$R *.lfm}

uses unit_search;

// изменить кодовую страницу в *.dbf
function SetLangCode(FileName:string; Code:byte):boolean;
var
  FS: TFileStream;
  b: byte;
begin
  Result:=true;
  b:=0;
  try
    FS:= TFileStream.Create(FileName, fmOpenReadWrite);
    FS.Seek(29, soFromBeginning);
    FS.ReadBuffer(b,1);
    if b <> Code then begin
      b:= Code;
      FS.Seek(-1, soFromCurrent);
      FS.WriteBuffer(b,1);
    end;
    FS.Free;
  except
    Result:= false;
  end;
end;
```

Строку uses unit_search пока закомментируйте. IDWriter и IDWork – это значения ключевых полей (идентификаторы) соответствующих таблиц (т.е. Code и NWork). Мы организуем автоматическое заполнение этих полей, поэтому их последние значения будем запоминать в данных переменных. Подсчёт их значений будет выполняться в момент сохранения новой записи. Для этого задействуем событие BeforePost в Dbf

```
procedure TForm1.Dbf1BeforePost(DataSet: TDataSet);
begin
  if DataSet.Fields[0].IsNull then begin
    inc(IDWriter);
    DataSet.Fields[0].AsInteger:= IDWriter;
  end;
end;
```

```
procedure TForm1.Dbf2BeforePost(DataSet: TDataSet);
begin
  if DataSet.Fields[0].IsNull then begin
    inc(IDWork);
    DataSet.Fields[0].AsInteger:= IDWork;
  end;
  if DataSet.FieldByName('NWriter').IsNull then
     DataSet.FieldByName('NWriter').Value:= Dbf1.Fields[0].Value
end;
```

В таблице Works ссылку на автора (NWriter) также заполняем автоматически. Поэтому соответствующие столбцы в DBGrid (Code, NWork, NWriter) удобно сделать ReadOnly, чтобы пользователь не мог ввести вручную эти числа. По окончании отладки проекта эти столбцы можно даже скрыть – указать свойство Visible=False.

Процедура создания таблиц БД (через соответствующий пункт меню):

```
procedure TForm1.MenuDBCreateClick(Sender: TObject);
begin
  Dbf2.Close;
  With Dbf1 do begin
    Close;
    with FieldDefs do begin
      Clear;
      Add('Code',ftInteger);
      Add('FullName',ftString,100);
      Add('BYear',ftInteger);
      Add('DYear',ftInteger);
      Add('State',ftString,40);
    end;
    TableName:= FWriters;
    CreateTable;
    Exclusive:= true;
    Open;
    AddIndex('iCode', 'Code', []);
  end;
  With Dbf2 do begin
    with FieldDefs do begin
      Clear;
      Add('NWork',ftInteger);
      Add('NWriter',ftInteger);
      Add('WName',ftString, 120);
      Add('CYear',ftInteger);
    end;
    TableName:= FWorks;
    CreateTable;
    Exclusive:= true;
    Open;
```

```
    AddIndex('iWork', 'NWork', []);
    AddIndex('iWriter', 'NWriter', []);
    Close;
  end;
     // после создания индексов требуется настройка связи
  Dbf2.MasterFields:= 'CODE';
  Dbf2.IndexName:= 'iWriter';
  Dbf2.Open;
  IDWriter:= 0;
  IDWork:= 0;
  ShowMessage('База данных создана успешно!');
end;
```

Процедура переиндексации таблиц БД (через пункт меню):

```
procedure TForm1.MenuIndexClick(Sender: TObject);
begin
  Dbf2.Close;
  With Dbf1 do begin
    Close;
    TableName:= FWriters;
    if FileExists(BaseDir + d + 'Writers.mdx') then
       DeleteFile(BaseDir + d + 'Writers.mdx');
    Exclusive:= true;
    Open;
    AddIndex('iCode', 'Code', []);
  end;
  With Dbf2 do begin
    TableName:= FWorks;
    if FileExists(BaseDir + d + 'Works.mdx') then
       DeleteFile(BaseDir + d + 'Works.mdx');
    Exclusive:= true;
    Open;
    AddIndex('iWork', 'NWork', []);
    AddIndex('iWriter', 'NWriter', []);
    Close;
  end;
  Dbf2.MasterFields:= 'CODE';
  Dbf2.IndexName:= 'iWriter';
  Dbf2.Open;
  ShowMessage('База данных переиндексирована!');
end;
```

Для удаления записи автора (в левой таблице) используем соответствующую процедуру для кнопки «Удалить». Чтобы использовать константы MB_YESNO, MB_ICONQUESTION и mrNO, следует подключить к модулю библиотеку LCLType (через **uses**).

```
    { --------------- Удаление автора ------------------ }
procedure TForm1.BitBtn1Click(Sender: TObject);
begin
  // uses LCLType - for MB_xxx
  if Application.MessageBox('Удалить запись?', PChar(Caption),
    MB_ICONQUESTION + MB_YESNO) = mrNO then exit;
  Dbf2.First;
  while not Dbf2.eof do Dbf2.Delete;
  Dbf1.Delete;
end;
```

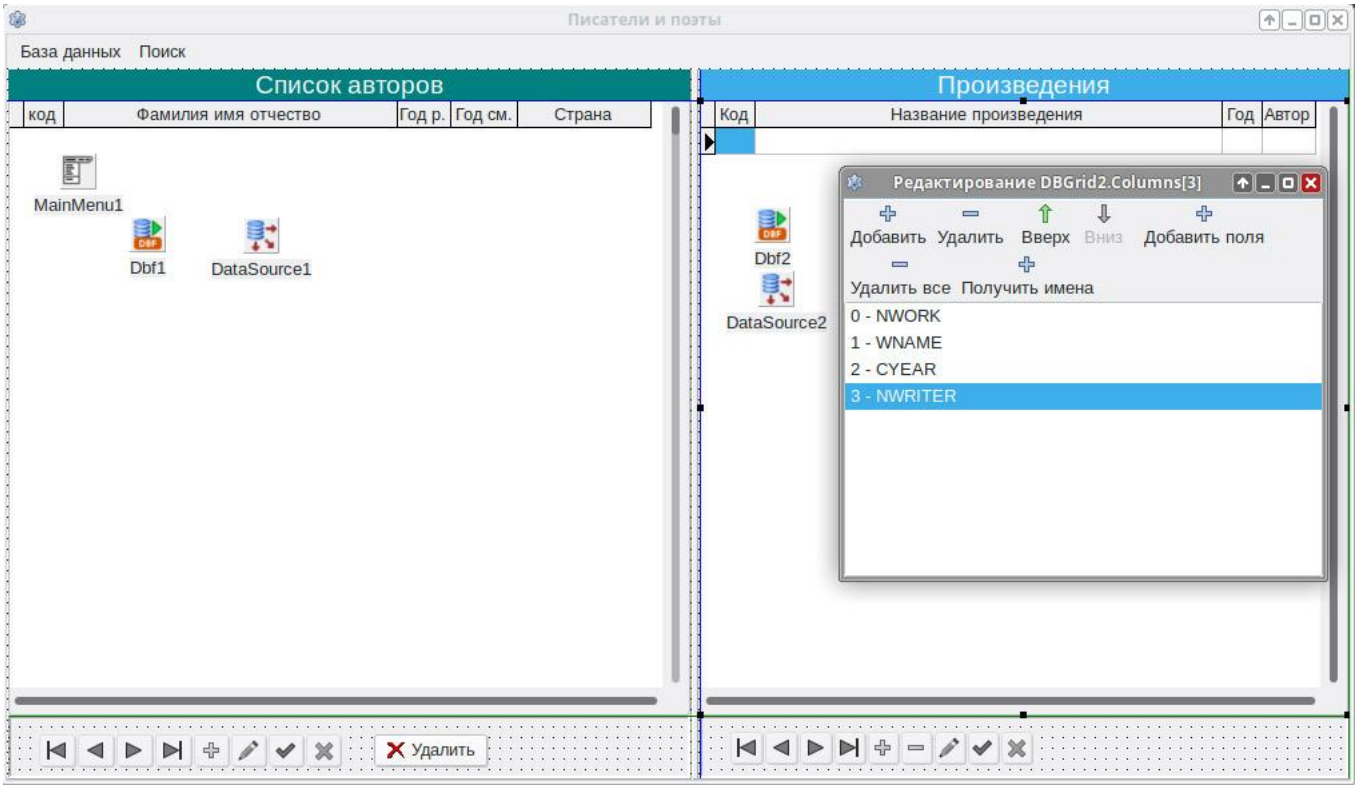
Здесь сначала удаляются подчинённые записи (произведения), затем – главная.

Теперь можно запустить программу и создать таблицы.

После создания таблиц настроим связь между ними. Сделаем это в Инспектореобъектов. Сначала укажем для компонентов Dbf1 и Dbf2 в свойстве TableName названия файлов Writers.dbf и Works.dbf. FilePath=Base/. Возможно, потребуется указать FilePathFull (полный путь к файлам). Для создания связи между таблицами используем свойства компонента Dbf2:

```
MasterSource = DataSource1
IndexName = iWriter
MasterFields = CODE
```

Теперь удобно будет настраивать столбцы компонента DBGrid. Правой кнопкой мышки вызываем редактор столбцов и в Инспекторе объектов заполняем свойства FieldName, Width,Title.Caption, Title.Alignment = taCenter. В этот момент следует установить для Dbf свойство Active=True.

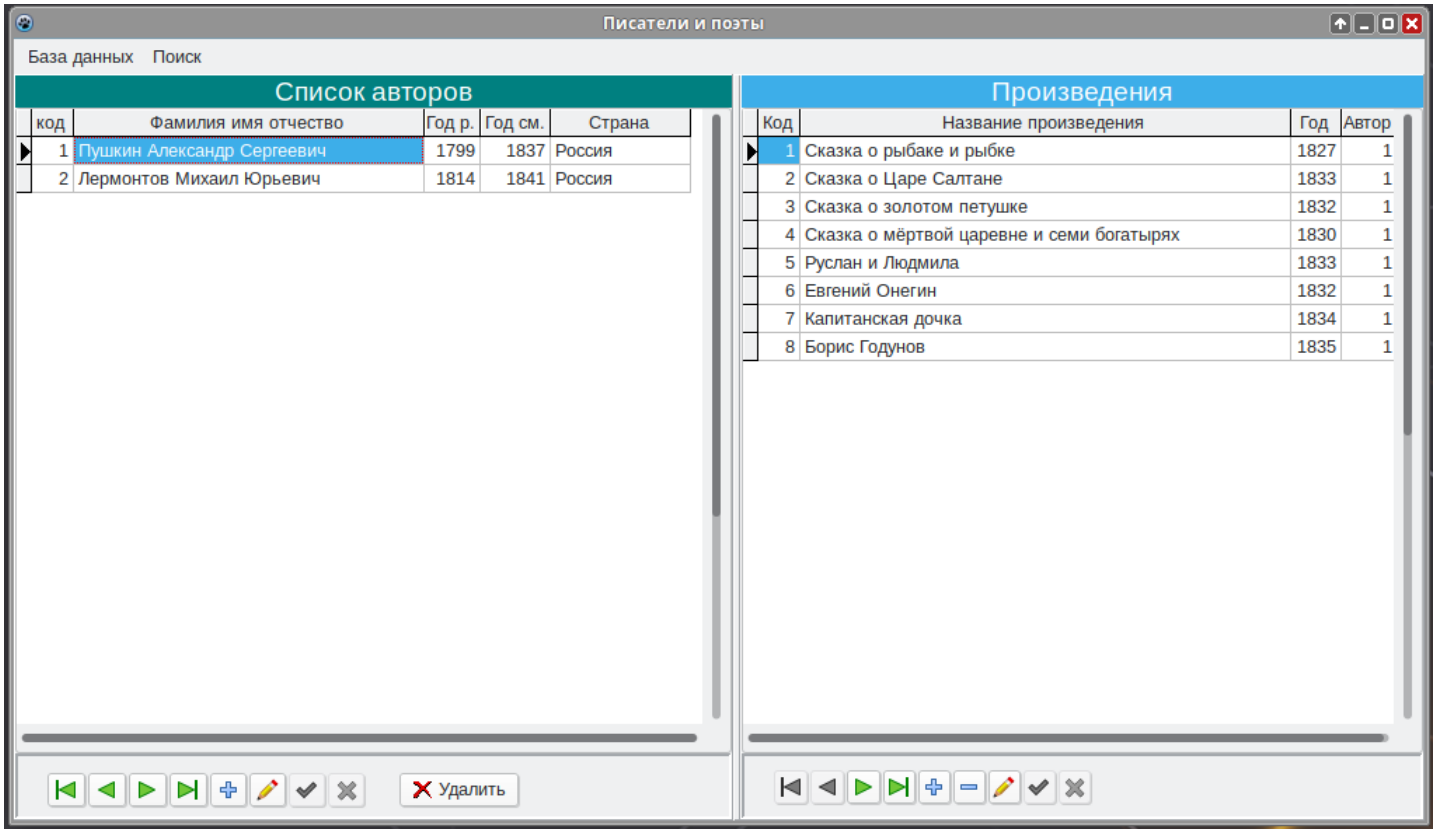


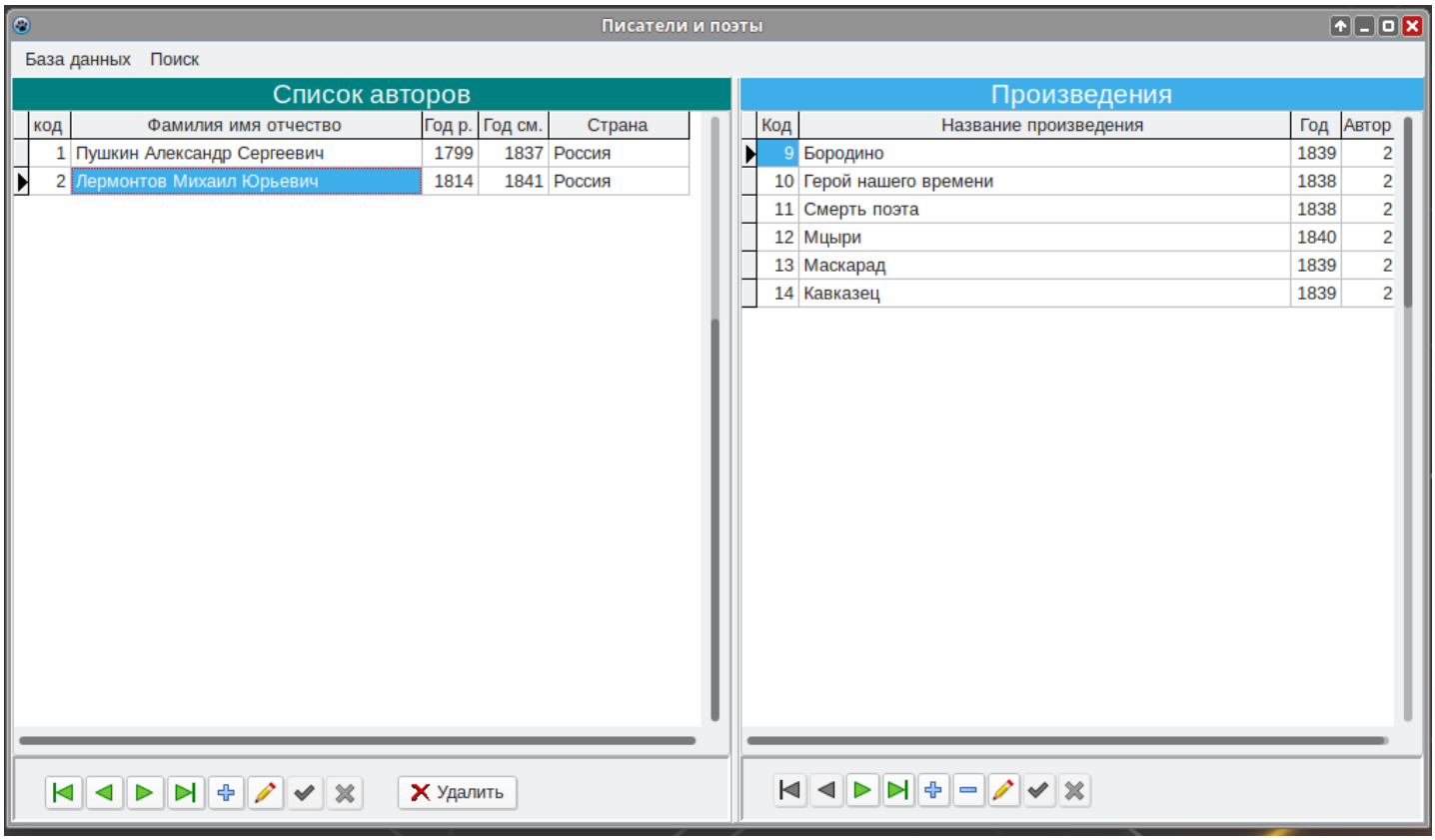
Чтобы запись сохранялась автоматически при покидании DBGrid, задействуем событие DBGrid.OnExit:

```
procedure TForm1.DBGrid1Exit(Sender: TObject);
begin
  if Dbf1.Modified then Dbf1.Post;
end;

procedure TForm1.DBGrid2Exit(Sender: TObject);
begin
  if Dbf2.Modified then Dbf2.Post;
end;
```

После этого можно проверить работу программы на заполнение данных в таблицах:



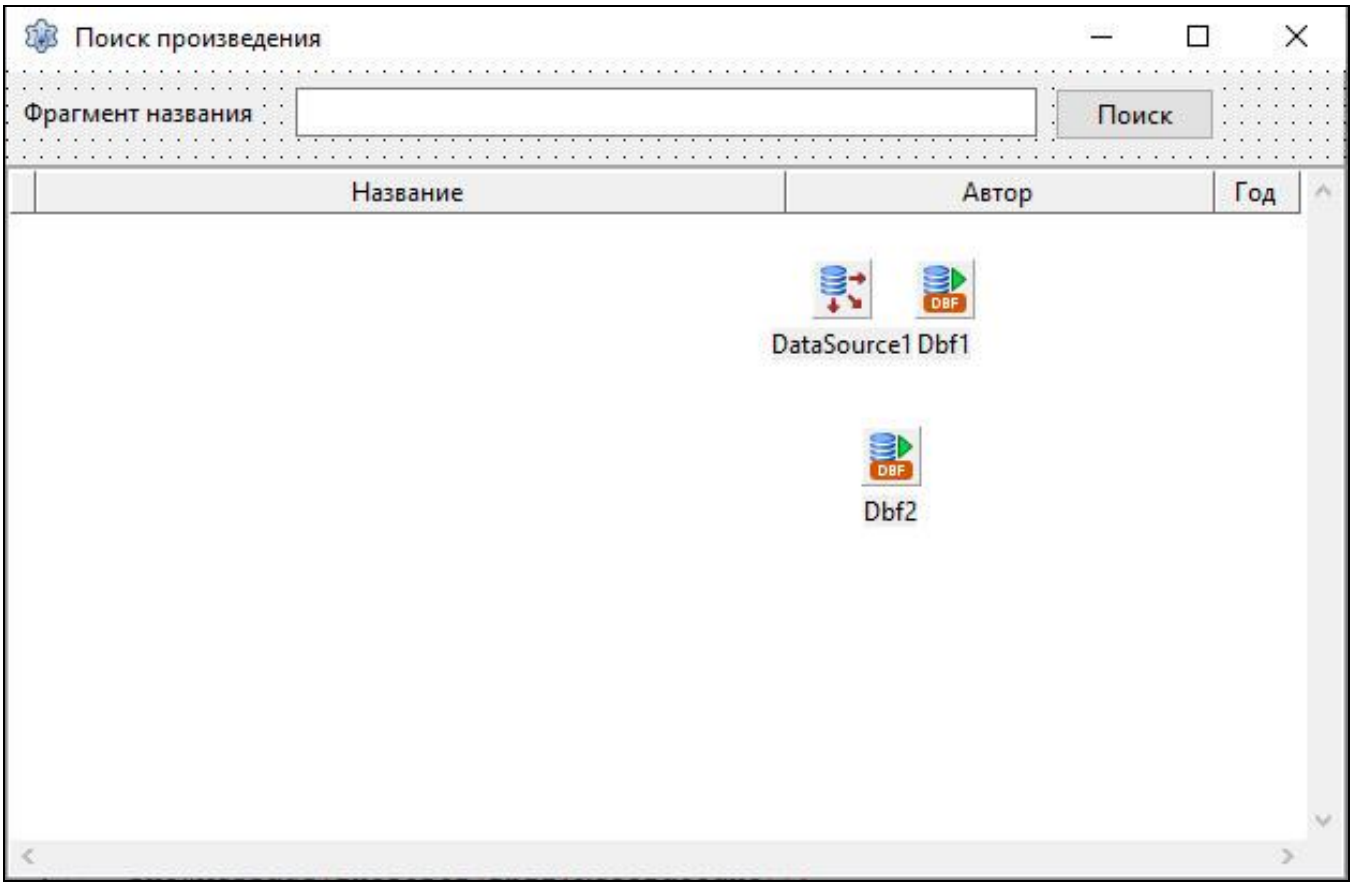


Обработчик события Form1.OnClose:

```
procedure TForm1.FormClose(Sender: TObject;
     var CloseAction: TCloseAction);
begin
  Dbf2.Close;
  Dbf1.Close;
  CloseAction:= caFree;
end;
```

Поиск произведения

Для этого создаём вторую форму



Чтобы удобнее было работать с DBGrid, следует в Инспекторе объектов выполнить соответствующие настройки DB-компонентов. Здесь у нас основная таблица Works, поэтому пусть для неё будет компонент Dbf1.

```
Dbf1.TableName = Works.dbf
Dbf1.FilePath = Base/
```

При этом ещё следует указать FilePathFull – полный путь к файлам БД.

Можно воспользоваться кнопкой загрузки в строке FilePath и задать путь к папке Base, которую перед этим создайте вручную. Тогда строка из FilePath сама скопируется в FilePathFull.

Аналогично и для Dbf2.

Таблица на форме у нас состоит из записей двух таблиц, так как имя автора мы будем брать из Writers, пользуясь ссылкой NWriter. В таком случае удобно создать дополнительное поле для основной таблицы – вычисляемое поле-объект.

Щёлкаем ПКМ по Dbf1. Выбираем «Редактировать поля». Если всё правильно настроено, то можно увидеть все поля таблицы. Все их добавляем в поля-объекты. Затем там же добавляем новое (вычисляемое) поле. Назовём его AuthorName, укажем тип String, размер – 100. Теперь столбец в DBGrid можно связать с этим полем.

Для этого вычисляемого поля задействуем событие Dbf1.OnCalcFields:

```
procedure TForm2.Dbf1CalcFields(DataSet: TDataSet);
var
   k: integer;
begin
  k:= Dbf1NWriter.Value;
  if Dbf2.SearchKey(k,stEqual) then
    Dbf1AuthorName.Value:= Dbf2.FieldByName('FullName').AsString;
end;
```

В Lazarus поиск по ключевому полю выполняется через функцию `SearchKey`, в которой параметром указываем константу `stEqual` (точное совпадение). Он находится в модуле `dbf_common`, поэтому его следует добавить в раздел **uses**.

Здесь используется поиск записи по индексу, поэтому в Инспекторе объектов в настройках Dbf2 следует указать `IndexName = ICODE`.

Для поиска произведений (выбора подходящих записей) будем применять фильтр методом OnFilterRecord:

```
procedure TForm2.Dbf1FilterRecord(DataSet: TDataSet; var Accept: Boolean);
var s,wn:string;
begin
  s:= AnsiUpperCase(Edit1.Text); // чтобы не зависеть от регистра
  wn:= AnsiUpperCase(DataSet.FieldByName('WNAME').AsString);
  if pos(s,wn) > 0 then Accept:= True
  else Accept:= False
end;
```

Чтобы это сработало, в Инспекторе объектов следует указать

`Dbf1.Filtered=True`.

Запускается процесс кнопкой «Поиск»:

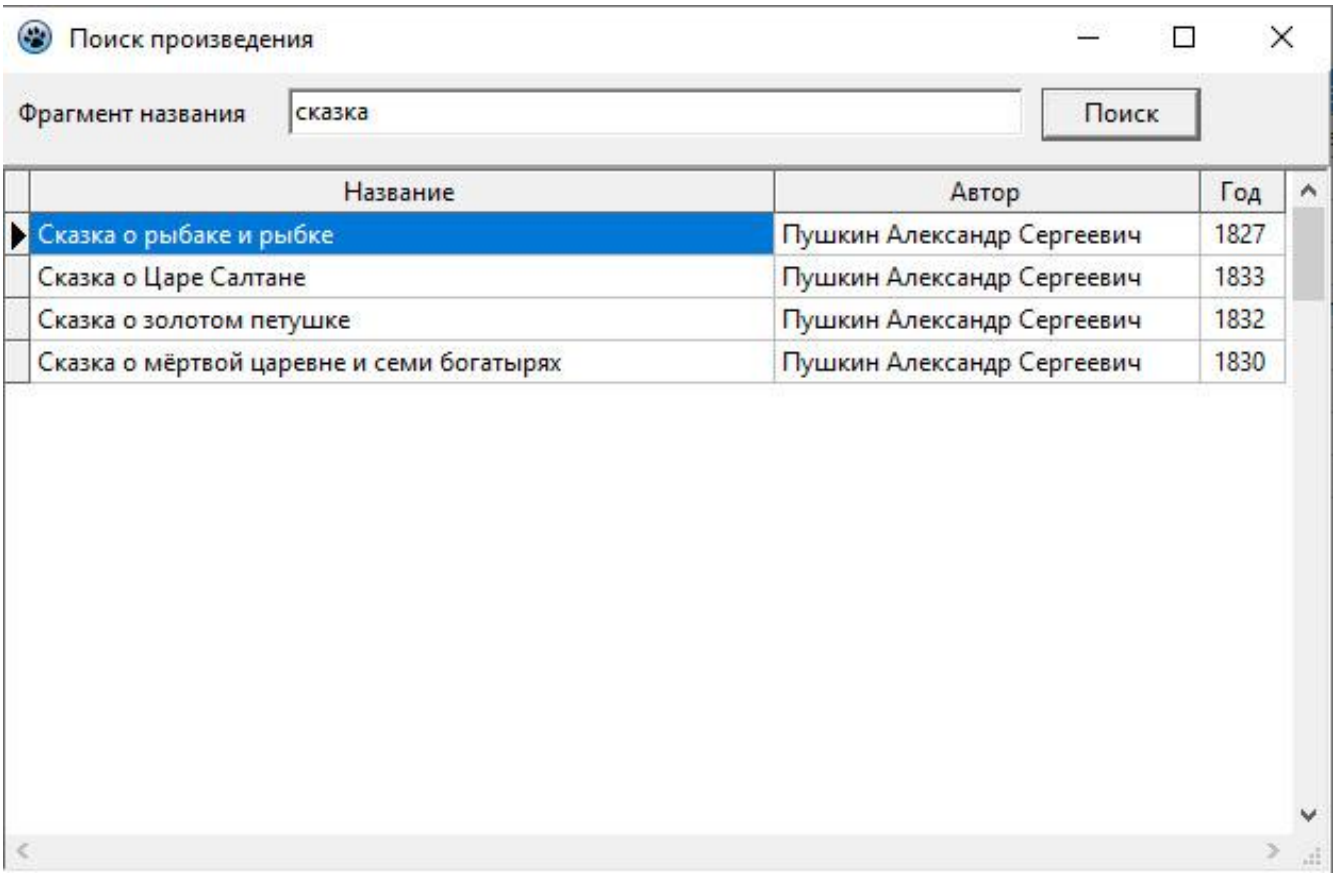
```
//  Dbf1 - for Works, Dbf2 - for Writers

procedure TForm2.Button1Click(Sender: TObject);
// var s:string;
begin
 {   так получается только с английским текстом...
  s:= Edit1.Text;
  Dbf1.Filter:='WNAME = "*' + s + '*"';
  Dbf1.Filtered:= true;
  }
  Dbf1.Close;
  Dbf2.Close;
  Dbf2.Open;
  Dbf1.Open;

end;
```

Здесь только закрывается старый набор данных и открывается новый.

Обработчик закрытия формы:

```
procedure TForm2.FormClose(Sender: TObject;
    var CloseAction: TCloseAction);
begin
  Dbf1.Close;
  Dbf2.Close;
  CloseAction:= caHide;
end;
```

Результат работы поиска: