МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНА МЕТАЛУРГІЙНА АКАДЕМІЯ УКРАЇНИ

Г. Г. Швачич, О. В. Овсянніков, В. В. Кузьменко, Н. І. Нєчаєва

СИСТЕМИ УПРАВЛІННЯ БАЗАМИ ДАНИХ

Частина 2

Затверджено на засіданні Вченої ради академії
як конспект лекцій

Дніпропетровськ НМетАУ 2007

УДК 004 (075.8)
Швачич Г.Г., Овсянніков О.В., Кузьменко В.В., Нєчаєва Н.І. Системи управління базами даних: Конспект лекцій. Частина 2. – Дніпропетровськ: НМетАУ, 2007. – 35 с.

Викладені основи конструювання систем управління базами даних у середовищі розробки прикладного програмного забезпечення Delphi.

Призначений для студентів спеціальності 6.020100 – документознавство та інформаційна діяльність.

Іл. 11. Бібліогр.: 5 найм.

Відповідальний за випуск Г.Г. Швачич, канд. техн. наук, проф.

Рецензенти: Б. І. Мороз, д-р техн. наук, проф. (Академія таможенної Служби України)
Т. І. Пашова, канд. техн. наук, доц. (Дніпропетровський державний аграрний університет)

ТЕМА 1 КЛАСС TDATASET – ОСНОВНОЙ КЛАСС ДОСТУПА К ДАННЫМ

Прежде чем приступить к разработке приложений баз данных необходимо познакомиться с механизмами доступа к данным и изучить свойства и методы базовых компонентов.

1.1 Класс TDataSet

Основным классом, определяющим доступ к данным, является класс **TDataSet**, который содержит абстрактные методы непосредственного управления данными. Данный класс является предком всех компонентов доступа к данным. При разработке приложений баз данных пользователь никогда не будет создавать объект типа **TDataSet**. Вместо этого пользователь будет использовать объекты **TTable**, **TQuery** или другие потомки класса **TDataSet.** Объект **TDataSet** представляет собой набор записей, содержащий определенное количество полей и указатель на текущую запись (рис. 1).

В большинстве случаев **TDataSet** имеет прямое соответствие (один к одному) с физической таблицей, существующей на диске. Тем не менее, **TDataSet** обладает методами, позволяющими получить любое подмножество записей одной таблицы или объединения нескольких таблиц. Однако, во многих случаях можно исполнять запрос или выполнять другое действие, возвращающие **DataSet**, содержащий любое подмножество записей одной таблицы, либо объединение (**join**) между несколькими таблицами. Далее в тексте будут иногда использоваться термины **DataSet** и **TTable** как синонимы.

Схема доступа к данным

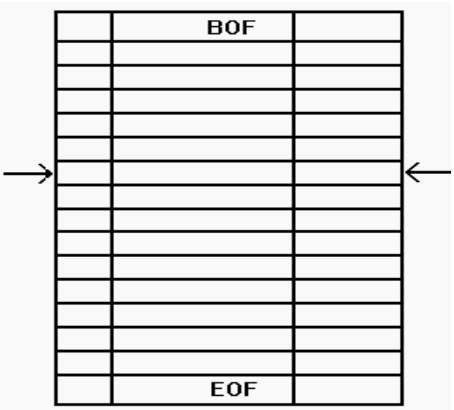

Рис.1

1.2 Открытие и закрытие DataSet

При использовании объекта **TTable** для доступа к таблице необходимо определить некоторые его свойства. Для изучения объекта необходимо поместить его во время дизайна в форму, и указать с какой таблицей будет установлена связь. Связь с таблицей устанавливается с помощью свойств **DatabaseName** и **TableName**. Для свойства **DatabaseName** достаточно указать директорий, в которой расположены таблицы в форматах **dBase** или **Paradox** например: **C:\DELPHI\DEMOS\DATA**. Также можно выбрать из списка псевдоним базы данных **DBDEMOS** или любой другой псевдоним. Псевдоним базы данных (**Alias**) определяется в утилите **Database Engine Configuration**. В поле **TableName** необходимо указать имя таблицы. Если свойство **Active** установлено в состояние **True**, то при запуске приложения таблица будет открываться автоматически.

Существует два способа открытия таблицы во время выполнения программы. Первый способ соответствует записи: **Table.Open;**

а второй способ заключается в установке свойства **Active** таблицы в состояние **True**: **Table.Active := True;**

Большого отличия между первым и вторым способом нет, так как метод **Open** заканчивается установкой свойства **Active** в состояние **True**. Поэтому второй способ несколько более эффективен.

Также имеется и два способа закрыть таблицу. Первый способ определяется вызовом метода **Close**: **Table.Close;**

Второй способ осуществляется установкой свойства **Active** в состояние **False**: **Table.Active := False;**

1.3 Навигация

Для перемещения по записям внутри таблицы объект **TDataSet** обладает следующими методами и свойствами:

- **procedure** First.
- **procedure** Last.
- **procedure** Next.
- **procedure** Prior.
- **property** BOF: Boolean **read** FBOF.

- **property** EOF: Boolean **read** FEOF.
- **procedure** MoveBy(Distance: Integer).

Метод **Table.First** перемещает указатель к первой записи в таблице. Метод **Table.Last** перемещает указатель к последней записи в таблице. Методы **Table.Next** и **Table.Prior** перемещают указатели на одну запись вперед и назад соответственно. Свойства **BOF** и **EOF** указывают на начало и конец таблицы. Метод **MoveBy** перемещает указатель на определенное число записей к началу или концу таблицы. Метод **Table.MoveBy(1)** аналогичен методу **Table.Next**, а метод **Table.MoveBy(-1)** аналогичен методу **Table.Prior.**

Рассмотрим на примере описанные методы и свойства объекта **TDataSet**. Необходимо поместить в форму следующие компоненты: **DataSource**, **Table**, **DBGrid**, **DBNavigator**, 4 компонента **Button** и компонент **SpinEdit**. Кнопкам **Button** необходимо присвоить следующие имена: **NextBtn**, **PriorBtn**, **FirstBtn**, **LastBtn**, **ReturnBtn**, **GoEndBtn** и **MoveBtn**. Посредством свойства **DataSource** требуется установить связь компонентов **DBGrid1** и **DBNavigator1** с компонентом **DataSource1**. Используя свойство **DataSet** компонента **DataSource1**, необходимо установить связь с компонентом **Table1**. Достаточно выбирать из списков **DataBaseName** псевдоним **DBDEMOS** и **TableName** имя таблицы **CUSTOMER** и установить свойство **Active** таблицы в состояние **True**. После выполнения данной операции в компоненте **DBGrid1** появятся данные таблицы **CUSTOMER** (рис.2).

Доступ к данным в период дизайна

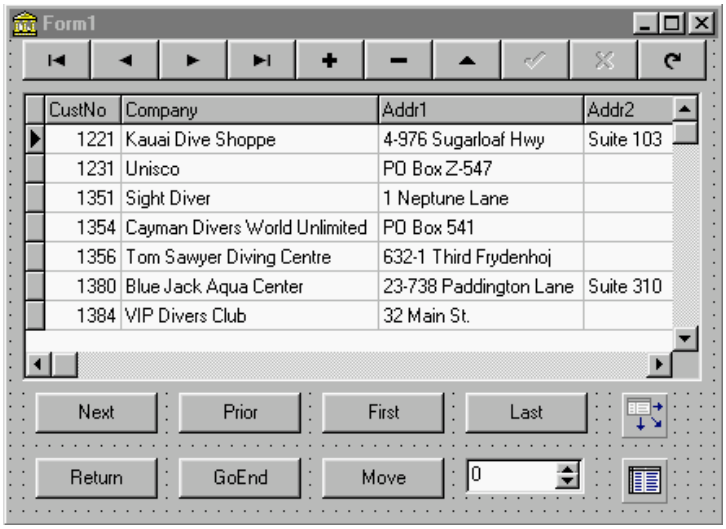

Рис.2

Теперь можно запустить приложение на выполнение и проследить перемещение по полям таблицы посредством нажатия соответствующих кнопок компонента **DBNavigator1** и полос прокруток компонента **DBGrid1**.

Компонент **DBNavigator** обладает практически всеми необходимыми свойствами и методами управления таблицей. Однако нашей задачей является рассмотрение вопроса программного управления объектом **TDataSet** без использования универсального элемента управления **DBNavigator**. Поэтому, для обработчиков событий **OnClick** кнопок необходимо написать строки, приведенные в примере 1.

Пример 1

```
procedure TForm1.NextBtnClick(Sender: TObject);
begin
  Table1.Next;
end;
procedure TForm1.PriorBtnClick(Sender: TObject);
begin
 Table1.Prior;
end;
procedure TForm1.FirstBtnClick(Sender: TObject);
begin
 Table1.First;
end;
procedure TForm1.LastBtnClick(Sender: TObject);
begin
  Table1.Last;
end;
```

Теперь можно запустить приложение и проверить функционирование кнопок.

При работе с базами данных часто возникает необходимость изменять значения полей для всей таблицы. С этой целью удобно использовать свойства **TDataSet.BOF** и **TDataSet.EOF**.

TDataSet.BOF – read-only Boolean свойство, используется для проверки, находитесь ли Вы в начале таблицы. Свойства **BOF** возвращает **true** в трех случаях:

- После того, когда пользователь открыл файл.
- После того, когда пользователь вызывал **TDataSet.First**.
- После того, когда вызов **TDataSet.Prior** не выполняется.

Первые два пункта – очевидны. Когда пользователь открывает таблицу, выполняется помещение на первую запись. Когда вызывается метод **First**, также происходит перемещение указателя в начало таблицы. Метод **Prior** требует небольшого пояснения. После того, когда пользователь многократно вызывает метод **Prior**, возникает ситуация достижения начала таблицы, и следующий вызов **Prior** будет неудачным. При этом **BOF** возвратит значение **True**. Используя данное свойство, можно определить достижение начала таблицы, как показано в примере 2.

Пример 2

```
procedure TForm1.ReturnBtnClick(Sender: TObject);
begin
  Table1.Last;
  while not Table1.Bof do
  begin
  { В данном месте должен находиться текст модификации
    полей  таблицы.}
  Table1.Prior;
  end;
end;
```

Аналогично свойству **BOF** применяется свойство **EOF** для проверки достижения конца таблицы. **EOF** возвращает **True** в следующих трех случаях:

- После того, когда пользователь открыл пустой файл.
- После того, когда пользователь вызывал **TDataSet.Last**.
- После того, когда вызов **TDataSet.Next** не выполняется.

Пример 3 демонстрирует простой способ перемещения по всем записям таблицы, начиная с первой записи.

Пример 3

```
procedure TForm1.GoEndBtnClick(Sender: TObject);
begin
Table1.First;
  while not Table1.EOF do
  begin
{В данном месте должен находится текст модификации полей таблицы.}
 Table1.Next;
  end;
end;
```

Для перемещения на заданное число записей в любом напрвлении используется метод **MoveBy**. Пример 4 демонстрирует указанный метод.

Пример 4

```
procedure TForm1.MoveBtnClick(Sender: TObject);
begin
   Table1.MoveBy(SpinEdit1.Value);
end;
```

1.4 Поля таблицы

Для получения доступа к полям таблицы объект **TDataSet** обладает рядом методов и свойств, основными из которых являются:

- **property** Fields[Index: Integer];
- **function** FieldByName(**const** FieldName: string): TField;
- **property** FieldCount.

Свойство **FieldCount** возвращает число полей в текущей структуре записи. Если необходимо программным путем прочитать имена полей, то для доступа к ним следует применить свойство **Fields** (пример 5).

Пример 5

```
var
S: String;
begin
S := Table.Fields[0].FieldName;
end;
```

В приведенном выше примере 5 переменной S присваивается имя первого поля таблицы, индекс которого соответствует нулю. Для доступа к имени последнего поля следует указать индекс равный **Table.FieldCount – 1 (**Пример 6).

Пример 6

```
var
S: String;
begin
 S := Table.Fields[Table.FieldCount - 1].FieldName;
end;
```

Если необходимо определить текущее содержание выбранного поля конкретной записи, то рекомендуется использовать свойство **Fields** или метод **FieldsByName**. Для этого достаточно найти значение i-го поля записи как i - й элемент массива **Fields** (пример 7).

Пример 7

```
var
S: String;
i: Integer;
begin
i := 3;
S := Table.Fields[i].AsString;
end;
```

Предположим, что указанное поле в записи содержит номер записи, тогда код, приведенный выше, возвратил бы строку типа «10», «1012» или «1024». Если требуется получить доступ к этой переменный, как к числовой величине, тогда необходимо использовать тип **AsInteger** вместо типа **AsString**. Аналогично, указываются и другие типы данных: **AsBoolean**, **AsFloat** и **AsDate**.

В примере 8 показано, что для доступа к данным можно применять функцию **FieldsByName** вместо свойства **Fields**.

Пример 8

```
var
S: String;
```

begin
S := Table.FieldsByName(‘Nuncode’).AsString;
end;

В приведенных примерах 5 – 8 показано, что метод **FieldsByName**, и свойство **Fields** возвращают те же самые данные. Два различных синтаксиса используются для того, чтобы обеспечить программистов гибким и удобным набором инструментов для доступа к содержимому **DataSet**.

Рассмотрим пример, как можно использовать доступ к полям таблицы во время выполнения программы. Для этого необходимо поместить в форму объект **Table**, два объекта **ListBox** и две кнопки **Button**, и присвоить кнопкам имена **Fields** и **Values** (рис. 3.). Далее требуется установить связь объекта **Table1** с базой данных **DBDEMOS** посредством свойства **DatabaseName** и выбрать таблицу **CUSTOMER** в списке свойства **TableName**. Для доступа к данным в период дизайна требуется установить свойство **Active** таблицы в состояние **True**.

Расположение элементов управления в форме

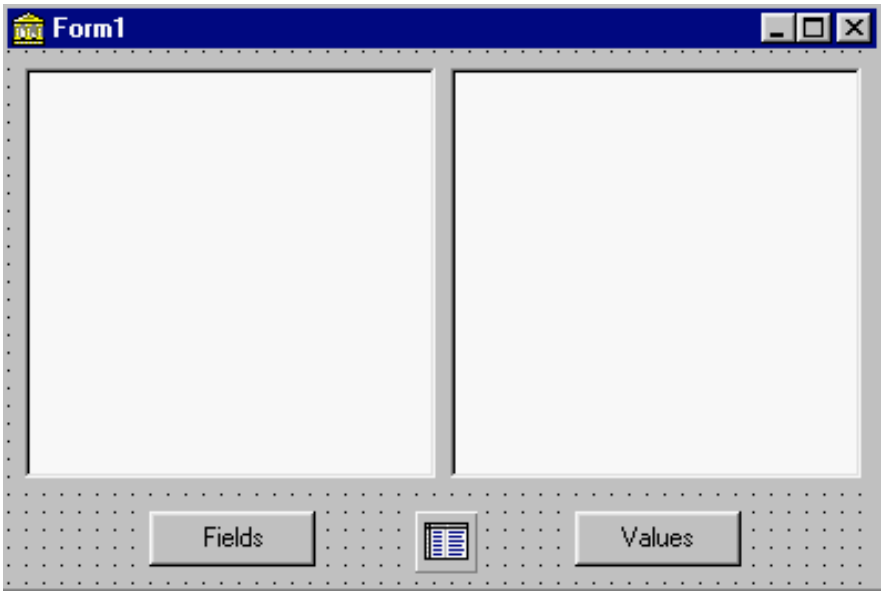

Рис.3

Далее необходимо написать обработчик события для нажатой кнопки **Fields** (пример 9).

Пример 9

```
procedure TForm1.FieldsClick(Sender: TObject);
var
i: Integer;
begin
ListBox1.Clear;
for i := 0 to Table1.FieldCount - 1 do
ListBox1.Items.Add(Table1.Fields[i].FieldName);
end;
```

Приведенная процедура (пример 9) функционирует следующим образом. Первая программная строка очищает компонент **ListBox1**. Затем в цикле последовательно добавляются имена полей таблицы. Обратите внимание на то, что счет в цикле начинается с 0, и заканчивается **Table1.FieldCount – 1**. Если не вычесть единицу из значения **FieldCount**, то произойдет исключительная ситуация «**List Index Out of Bounds**», обусловленная попыткой прочесть имя поля которое не существует.

Для получения доступа к содержимому полей обработчик события для кнопки **Values** будет иметь вид, приведенный в примере 10.

Пример 10

```
procedure TForm1.ValuesClick(Sender: TObject);
var
i: Integer;
begin
ListBox2.Clear;
for i := 0 to Table1.FieldCount - 1 do
ListBox2.Items.Add(Table1.Fields[i].AsString);
end;
```

На рисунке 4 приведено функционирующее приложение, демонстрирующее выполнение процедур приведенных в примерах 9 – 10.

Реализация процедур

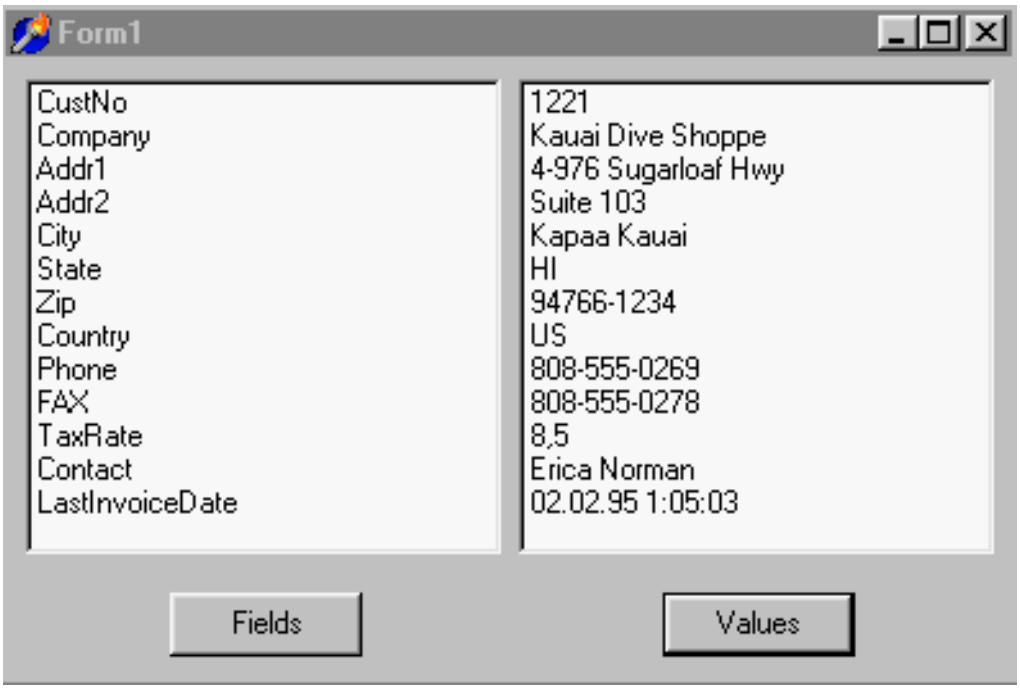

Рис.4

Напомним, что класс **TField** обладает следующими свойствами:

- **property** AsBoolean;
- **property** AsFloat;
- **property** AsInteger;
- **property** AsString;
- **property** AsDateTime.

Если возникает необходимость, то можно преобразовывать тип поля **Boolean** к **Integer** или к **Float**, тип поля **Integer** к **String** или тип поля **Float** к **Integer**. Однако преобразовывать тип поля **String** к **Integer** невозможно. Если необходимо работать с полями **Date** или **DateTime**, то можно использовать преобразования **AsString** и **AsFloat** для доступа к ним.

1.5 Работа с данными

Для работы с данными объект TTable имеет следующие методы:

- **procedure** Append;
- **procedure** Insert;
- **procedure** Cancel;

- **procedure** Delete;
- **procedure** Edit;
- **procedure** Post;

Все указанные методы являются методами **TDataSet**. Они унаследованы и используются объектами **TTable** и **TQuery**. Всякий раз, когда пользователь желает изменить данные, он должен сначала перевести **DataSet** в моду редактирования. Большинство элементов управления базами данных переводят **DataSet** в моду редактирования автоматически. Однако, если необходимо изменить **TTable** программно, пользователю придется использовать вышеуказанные функции.

Для изменения содержимого поля в текущей записи необходимо выполнить последовательность программных строк (пример 11).

Пример 11

```
Table1.Edit;
Table1.FieldByName('CustName').AsString := 'Client 1024';
Table1.Post;
```

Первая строка программы переводит **DataSet** в режим редактирования. Вторая строка присваивает значение полю '**CustName**'. Третья строка обеспечивает запись данных на диск, путем вызова метода **Post**.

Другим способом сохранения записи на диске является перемещение к следующей записи после ее редактирования (пример 12).

Пример 12

```
Table1.Edit;
Table1.FieldByName('CustNo').AsInteger := 1024;
Table1.Next;
```

На основании вышеизложенного, следует вывод, что всякий раз, когда выполняется перемещение с текущей записи на другую в режиме редактирования выполняется запись на диск. Это означает, что вызовы методов **First**, **Next**, **Prior** и **Last** завершаются методом **Post**, при условии, что объект **DataSet** нахотится в состоянии редактирования.

Для добавления и вставки новых записей в таблицу используются методы **Append** и **Insert** соответственно. Рассмотрим данные методы на следующем примере. Для демонстрации указанных методов необходимо

разместить в форме компоненты **Table**, **DataSource**, **DBGrid** и три кнопки **Button**. Присвоим кнопкам соответствующие имена: **InsertBtn**, **AppendBtn** и **DeleteBtn**. Далее необходимо установить соединения между объектами **DataSource1**, **Table1**, **DBGrid1,** и связать объект **Table1** с таблицей **COUNTRY** базы данных **DBDEMOS** (рис. 5). Для доступа к данным в период дизайна требуется установить свойство **Active** объекта **Table1** в состояние **true**.

Расположение элементов управления в форме

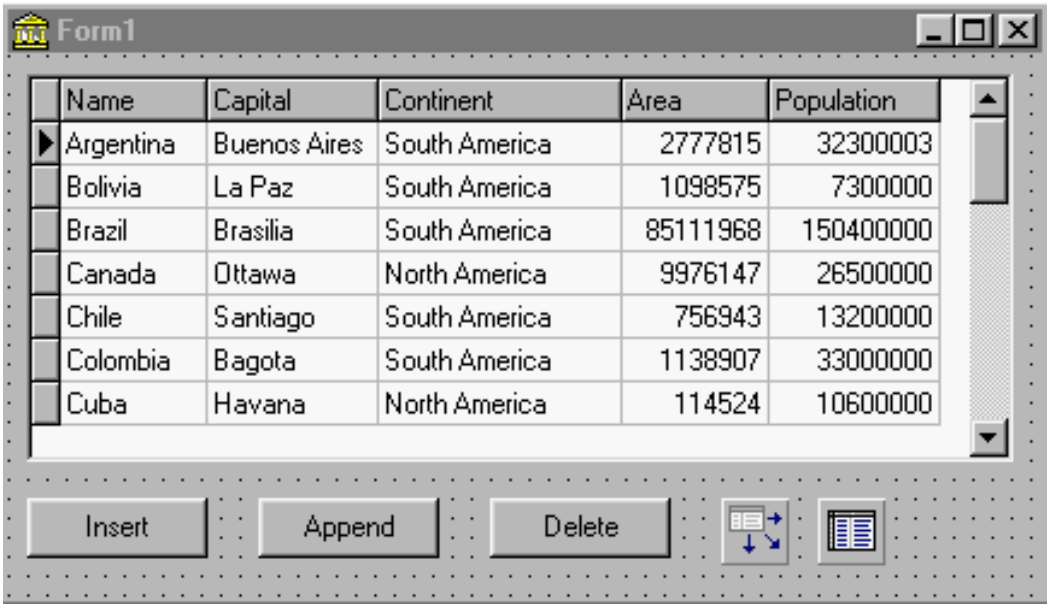

Рис.5

Процедуры обработчиков событий **OnClick** кнопок, в нашем случае, будут иметь вид приведенный в примере 13.

Пример 13

```
procedure TForm1.InsertBtnClick(Sender: TObject);
begin
  Table1.Insert;
  Table1.FieldByName('Name').AsString := 'Ukraine';
  Table1.FieldByName('Capital').AsString := 'Kiev';
  Table1.Post;
end;
procedure TForm1.AppendBtnClick(Sender: TObject);
```

```
begin
  Table1.Append;
  Table1.FieldByName('Name').AsString := 'Ukraine';
  Table1.FieldByName('Capital').AsString := 'Kiev';
  Table1.Post;
end;
procedure TForm1.DeleteBtnClick(Sender: TObject);
begin
   Table1.Delete;
end;
```

Процедура **TForm1.InsertBtnClick(Sender: TObject)** переводит таблицу в режим вставки, то есть, создается новая запись с пустыми полями, которая вставляется в текущую позицию **dataset**. После вставки пустой записи, выполняется присвоение значений одному или нескольким полям, а затем вызывается метод **Post**. Аналогично функционирует обработчик события **AppendBtnClick**.

Существует несколько различных способов присвоения значений полям таблицы. В рассматриваемой программе пользователь мог бы просто ввести информацию в новую запись, используя возможности **DBGrid**. Также пользователь мог разместить в форме стандартное элемент ввода **TEdit** и присвоить каждому полю значение, которое он ввел в элемент ввода:

Table1.FieldByName(‘Name’).AsString := Edit1.Text;

Интересным моментом в рассматриваемом примере является то, что нажатие кнопки **Insert** дважды подряд автоматически вызывает исключительную ситуацию ‘**Key Violation**’, так как первое поле таблицы **COUNTRY** опредедено как певичный ключ. Таким образом, **DataSet** защищает базу данных от повторного присвоения одинакового имени.

Если после вызова методов **Insert** или **Append**, необходимо отказаться от вставки или добавления новой записи, то надо вызвать метод **Cancel** до метода **Post**.

1.6 Использование SetKey для поиска записей в таблице

С целью нахождения необходимых значений полей таблицы, используются две процедуры **SetKey** и **GotoKey**. Обе эти процедуры предполагают, что поле, по которому производится поиск индексировано.

Для того, чтобы создать программу поиска данных, требуется поместить в форму компоненты **Table**, **DataSource**, **DBGrid**, **Button**, **Label** и **Edit**, и расположить их как показано на (рис. 5). Далее необходимо изменить имя кнопки **Button** на имя **SearhBtn**, и затем соединить компоненты управления базой данных так, чтобы пользователь открыл в **DBGrid1** таблицу **Customer**.

Расположение компонентов в форме

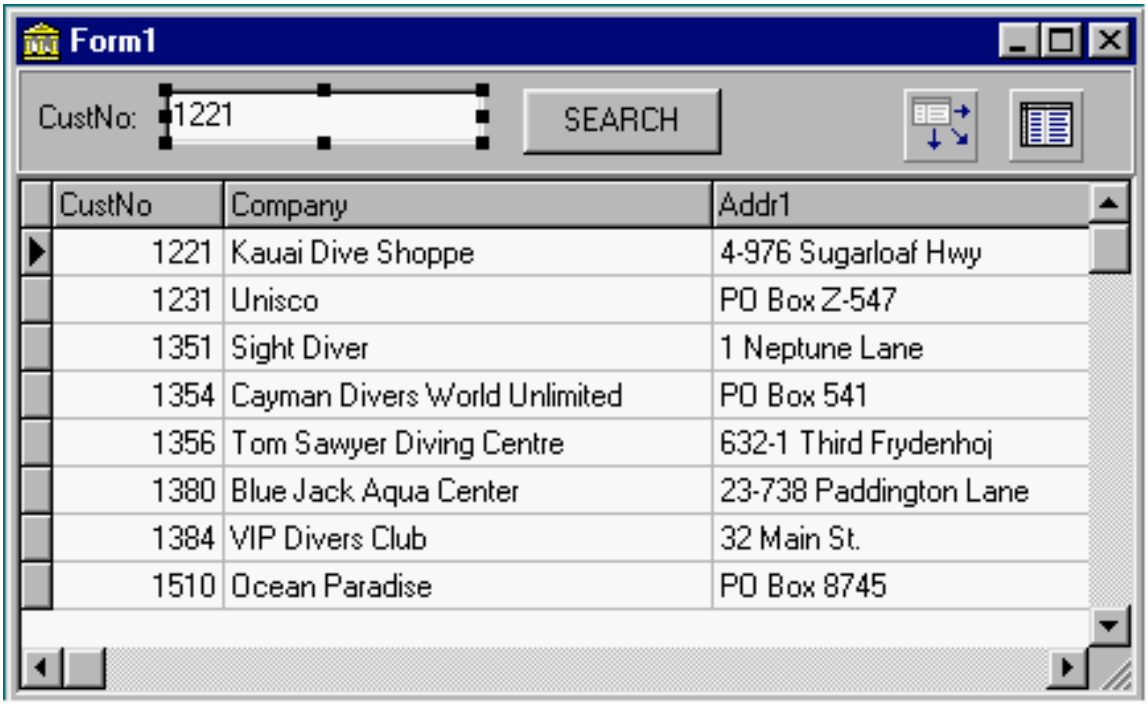

Рис.6

Вся функциональность данной программы скрыта в единственном методе, который является обработчиком события **OnClick** кнопки **SearchBtn**. Данный обработчик считывает строку, введенную в окно редактора, и ищет ее значение в поле **CustNo**, и помещает фокус на найденной записи. В простейшем варианте, код обработчика события выглядит так (пример 14):

Пример 14

```
procedure TForm1.SearhBtnClick(Sender: TObject);
begin
Table1.SetKey;
Table1.FieldByName('CustNo').AsString := Edit1.Text;
Table1.GotoKey;
end;
```

Первый вызов в этой процедуре установит **Table1** в режим поиска. Далее, свойству **Fields** присваивается значение, которое пользователь ввел в элемент ввода. Для фактического выполнения поиска нужно вызывать метод **Table1.GotoKey**.

Если необходимо выполнить поиск не по первичному индексу файла, тогда пользователю потребуется определить имя индекса, который будет использоваться свойством IndexName. Например, если таблица Customer имеет вторичный индекс по полю City, тогда пользователь должен установить свойство IndexName равным имени индекса. Реализация описанного метода приведена в примере 15.

Пример 15

```
procedure TForm1.SearhBtnClick(Sender: TObject);
begin
Table1.IndexName := 'CityIndex';
Table1.Active := True;   Table1.SetKey;
Table1.FieldByName('City').AsString := Edit1.Text;  Table1.GotoKey;
end;
```

Обращаем внимание на тот факт, что поиск не будет выполняться, если пользователь не назначит правильно индекс (свойство **IndexName**). Также, пользователь должен помнить, что **IndexName** - это свойство объекта **Ttable**, и не присутствует в других прямых потомках **TDataSet** или **TDBDataSet**.

Когда пользователь производит поиск некоторого значения, всегда существует вероятность того, что поиск окажется неудачным. В таком случае будет автоматически возникать исключительная ситуация (**exception**). Если возникает необходимость обработать ошибку самостоя-

тельно, то пользователь мог бы написать примерно такую процедуру (пример 16).

Пример 16

```
procedure TForm1.SearhBtnClick(Sender: TObject);
begin
Cust.SetKey;
Cust.FieldByName('CustNo').AsString:= CustNoEdit.Text;
if not Cust.GotoKey then
raise Exception.CreateFmt('Cannot find CustNo %g', [CustNo]);
end;
```

В приведенном примере 16, неверное присвоение номера, или неудача поиска автоматически приведут к сообщению об ошибке:

‘Cannot find CustNo %g’.

Иногда требуется найти не точно совпадающее значение поля, а близкое к его значению. Для этого следует вместо **GotoKey** пользоваться методом **GotoNearest**.

1.7 Использование фильтров для ограничения числа записей

Процедура **ApplyRange** позволяет установить фильтр, который ограничивает диапазон просматриваемых записей. Например, в таблице **Customers**, поле **CustNo** имеет диапазон от 1,000 до 10,000. Если пользователь желает получить доступ к идентификаторам в диапазоне 2000 – 3000, то он должен использовать метод **ApplyRange**, и еще два связанных с ним метода. Данные методы работают только с индексированным полем.

Приведенные ниже процедуры наиболее часто используются для установки фильтров:

- **procedure** SetRangeStart;
- **procedure** SetRangeEnd;
- **procedure** ApplyRange;
- **procedure** CancelRange.

Кроме того, объект **TTable** содержит дополнительные методы для управления фильтрами:

- **procedure** EditRangeStart;

- **procedure** EditRangeEnd;
- **procedure** SetRange.

Для создания фильтра с использованием указанных процедур необходимо:

- Вызвать метод **SetRangeStart** и посредством свойства **Fields** определит начало диапазона.
- Вызвать метод **SetRangeEnd** и посредством свойства **Fields** указать конец диапазона.
- Для активизации фильтра достаточно вызвать метод **ApplyRange**.
- Чтобы отказаться от действия фильтра необходимо вызвать метод **CancelRange**.

Рассмотрим пример использования приведенных методов для создания фильтра. Для этого необходимо поместить компоненты **Table**, **DataSource** и **DbGrid** в форму и соединить их поля таким образом, чтобы обеспечить доступ к таблице **CUSTOMERS** базы данных **DEMOS**. Затем поместить в форму два объекта **Label** и назовить их **Start Range** и **End Range**. Затем включить в форму два объекта **Edit** и две кнопки **ApplyRange** и **CancelRange** (рис.7).

Расположение компонентов в форме

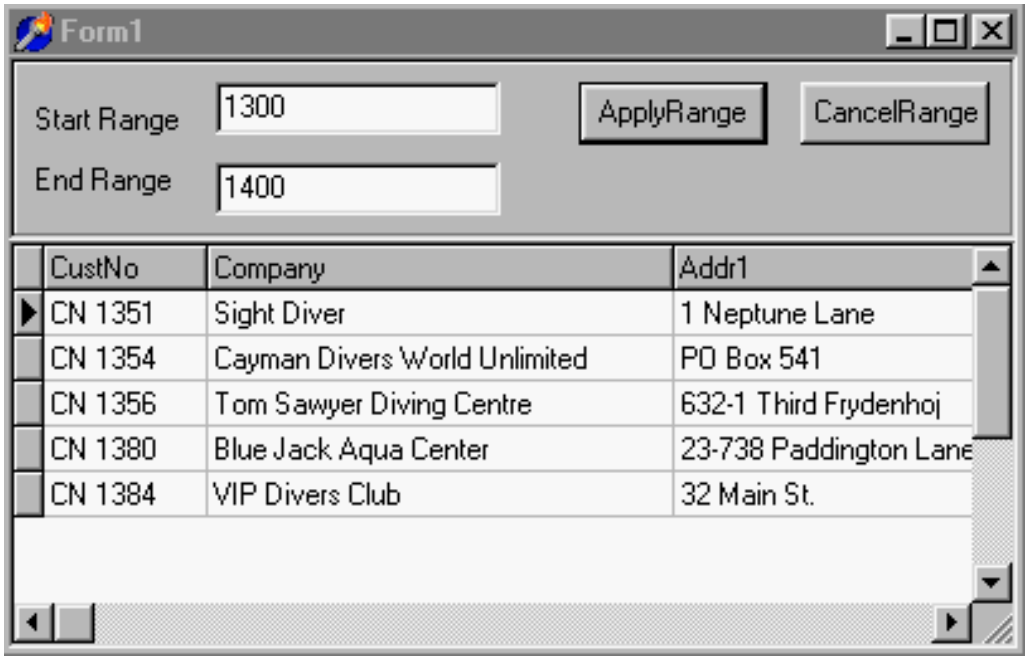

Рис.7

В примере 17 приведены процедуры, реализующие фильтр указанными методами.

Пример 17

```
procedure TForm1.ApplyRangeClick(Sender: TObject);
begin
 Table1.SetRangeStart;
 if Edit1.Text <> '' then
 Table1. Fields[0].AsString := Edit1.Text;
 Table1.SetRangeEnd;
 if Edit2.Text <> '' then
 Table1.Fields[0].AsString := Edit2.Text;
 Table1.ApplyRange;
end;
procedure TForm1.CancelRangeClick(Sender: TObject);
begin
  Table1.CancelRange;
end;
```

1.8 Обновление данных

Известно, что любая таблица базы данных, с которой работает пользователь, подвержена изменению другим пользователем. То есть, пользователь всегда должны расценить таблицу как изменяющуюся сущность. Даже если лицо является единственным пользователем, всегда существует возможность того, что приложение управления базой данных, может иметь два различных пути изменения данных в таблице. Поэтому, пользователь должен регулярно обновлять представление данных таблицы на экране.

Для обновления данных **DataSet** располагает функцией **Refresh**.Функция **Refresh** связана с функцией **Open**, таким образом, что она считывает данные, или некоторую часть данных, связанных с данной таблицей. Например, когда пользователь открывает таблицу, **DataSet** считывает данные непосредственно из файла базы данных. Аналогично, когда пользователь регенерирует таблицу, **DataSet** считывает данные напрямую из таблицы. Вследствие чего, всегда можно использовать эту функцию,

чтобы обновить таблицу. Быстрее и эффективнее, вызывать метод **Refresh**, чем метод **Close** и затем метод **Open**.

Обращаем внимание на то, что обновление **TTable** может иногда привести к неожиданным результатам. Например, если один пользователь рассматривает запись, которая уже была удалена другим пользователем, то она исчезнет с экрана в тот момент, когда будет вызван **Refresh**. Аналогично, если другой пользователь редактировал данные, то вызов **Refresh** приведет к динамическому изменению данных. Конечно, маловероятно, что один пользователь будет изменять или удалять запись в то время, как другой просматривает ее, но все же это возможно.

1.9 Закладки

Часто бывает полезно отметить текущее местоположение в таблице так, чтобы можно имелась возможность возвратиться к этому месту в дальнейшем. **DataSet** обеспечивает эту возможность посредством трех методов:

- **function** GetBookmark: TBookmark – устанавливает закладку в таблице.
- **procedure** GotoBookmark(Bookmark: TBookmark) – переводит указатель на закладку.
- **procedure** FreeBookmark(Bookmark: TBookmark) – освобождает память.

Как видно, вызов метода **GetBookmark** возвращает переменную типа **Tbookmark**. Метод **TBookmark** содержит достаточное количество информации, чтобы **DataSet** мог найти местоположение, к которому относится этот **TBookmark**. Поэтому пользователь может передавать **Tbookmark** функции **GotoBookmark**, которая немедленно возвратит указатель к местоположению, связанному с закладкой. Обращаем внимание на то, что вызов метода **GetBookmark** выделяет память для **TBookmark**, так что пользователю необходимо вызывать метод **FreeBookmark** до завершения работы программы, или перед каждой попыткой повторного использования метода **GetBookMark**.

1.10 Создание связанных курсоров

Связанные курсоры позволяют определить отношение один ко многим (**one-to-many relationship**). Например, иногда полезно связать таблицы, на-

пример CUSTOMER и ORDERS так, чтобы каждый раз, когда пользователь выбирает имя заказчика, то он видит список заказов связанных с этим заказчиком. Иначе говоря, когда пользователь выбирает запись о заказчике, то он может просматривать только заказы, сделанные этим заказчиком.

Рассмотрим пример программы, которая использует связанные курсоры. Дизайн программы сводится к установке в форму двух экземпляров объектов TTable, двух экземпляров TDataSource и двух экземпляров элементов управления TDBGrid. Далее необходимо присоединить первый набор к таблице CUSTOMER, а второй к таблице ORDERS. Программа в этой стадии имеет вид, приведенный на рис 8.

Дизайн программы

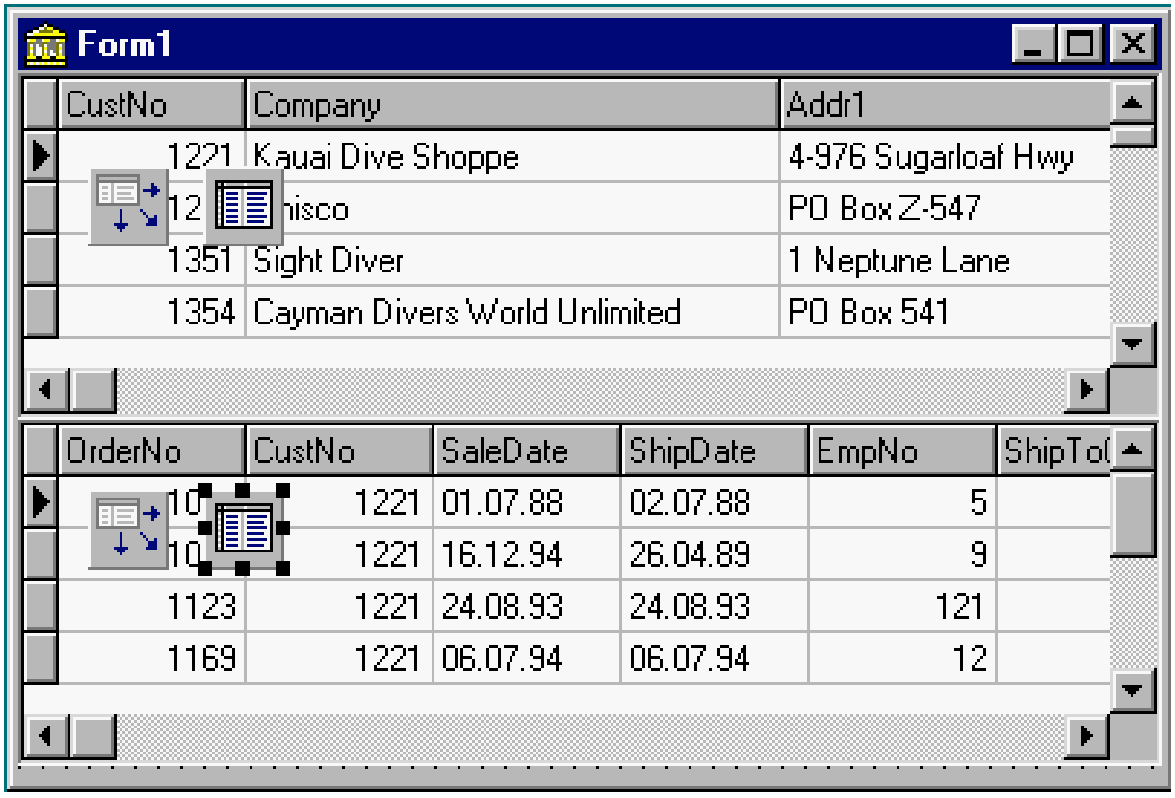

Рис.8

После выполнения дизайна следует связать таблицу ORDERS с таблицей CUSTOMER так, чтобы были видимы только те заказы, которые связанные с текущей записью в таблице заказчиков. В первой таблице заказчик однозначно идентифицируется своим номером – поле **CustNo**. Во второй таблице принадлежность заказа определяется также номером заказ-

чика в поле **CustNo**. Следовательно, таблицы нужно связывать по полю **CustNo.** Следует заметить, что в обеих таблицах поля могут иметь различное название, но должны быть совместимы по типу. Для этого, чтобы установить связь необходимо выполнить три действия.

- Установить свойство Table2.MasterSource = DataSource1.
- Установить свойство Table2.MasterField = CustNo.
- Установить свойство Table2.IndexName = CustNo.

Второе действие выполняется посредством редактора связей (рис. 9).

Общий вид редактора связей

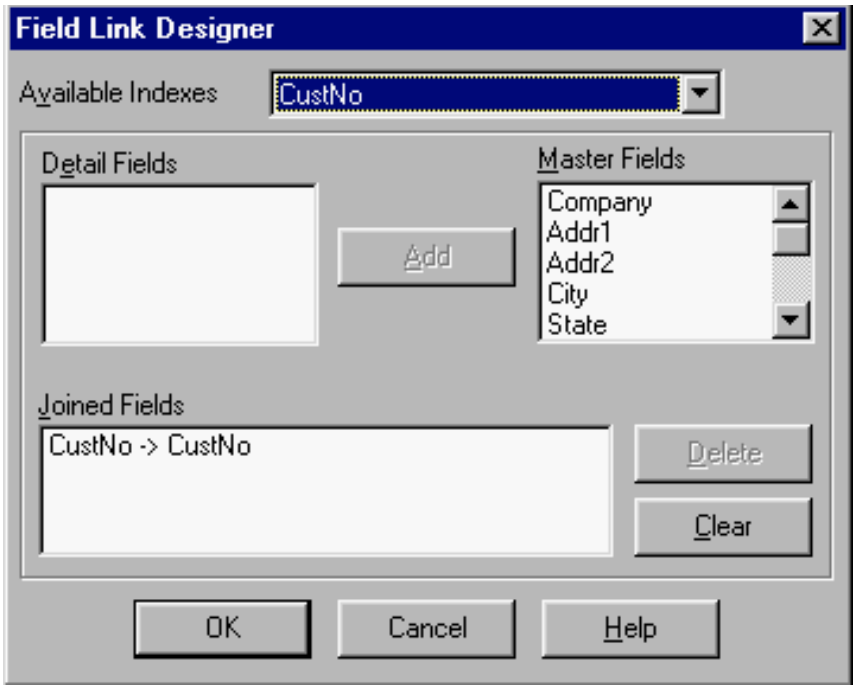

Рис.9

В период дизайна и после запуска приложения видно, что обе таблицы связаны вместе, и всякий раз, когда пользователь выполнит перемещение на новую запись в таблице CUSTOMER, в таблице ORDERS будут отображаться только те записи, которые принадлежат соответствующему заказчику.

Связанные курсоры позволяют также определить отношение многие ко многим (**mamy-to-many relationship**). Для реализации такого отношения связанные таблицы должны содержать комплексные индексы, состоящие из нескольких полей. Причем, поле входящее в состав комплексного индекса может иметь и собственный индекс.

ТЕМА 2 ПРИМЕР КОНСТРУИРОВАНИЯ БАЗЫ ДАННЫХ В СРЕДЕ DELPHI

Сформулируем основную концепцию разработки базы данных:

- Справочник должен представлять локальную базу данных, в которой каждая таблица должна соответствовать тематическому разделу справочника.
- Каждая таблица должна иметь одинаковую структуру.
- Таблица должна содержать поисковое и описательное поле.
- Приложение управления базой данных должно обеспечивать просмотр и запись данных.
- В процессе создания базы данных приложение должно обеспечить возможность загрузки ранее подготовленных файлов примеров.
- В процессе работы со справочником должна быть обеспечена защита от случайного повреждения данных.
- С целью обеспечения возможности перемещения на другие компьютеры таблицы базы данных должны размещаться в папке приложения, т.е. без изменения конфигурации ядра базы данных.

На основании сформулированной концепции можно приступить к созданию базы данных и разработке приложения управления базой данных. Работу предлагается выполнять в следующем порядке:

1. Используя утилиту **DataBaseDesctop,** создайте в формате **Paradox** таблицы **Funct1**, **Funct2**, … **Funct(n)** соответственно количеству разделов. Структура всех таблиц должна быть одинаковой. Каждая таблица должна иметь содержать поля, приведенные в таблице 1.

Таблица 1

	Field Name	**Type**	**Size**	**Key**
	FunctName	A	20	*
	Description	F	240	

2. Создайте новую папку будущего проекта и сохраните в ней созданные таблицы. Используя команду **Alias Manager** меню **Tools** утилиты **DataBaseDesctop**, создайте временный псевдоним будущей базы данных, например **DelphiFunction**.
3. В среде **Delphi** создайте новый проект и сохраните его с уникальным именем в папку, содержащую таблицы базы данных.
4. Выполните дизайн приложения управления базой данных согласно рисунку 10 и таблице 2.

Расположение компонентов в форме

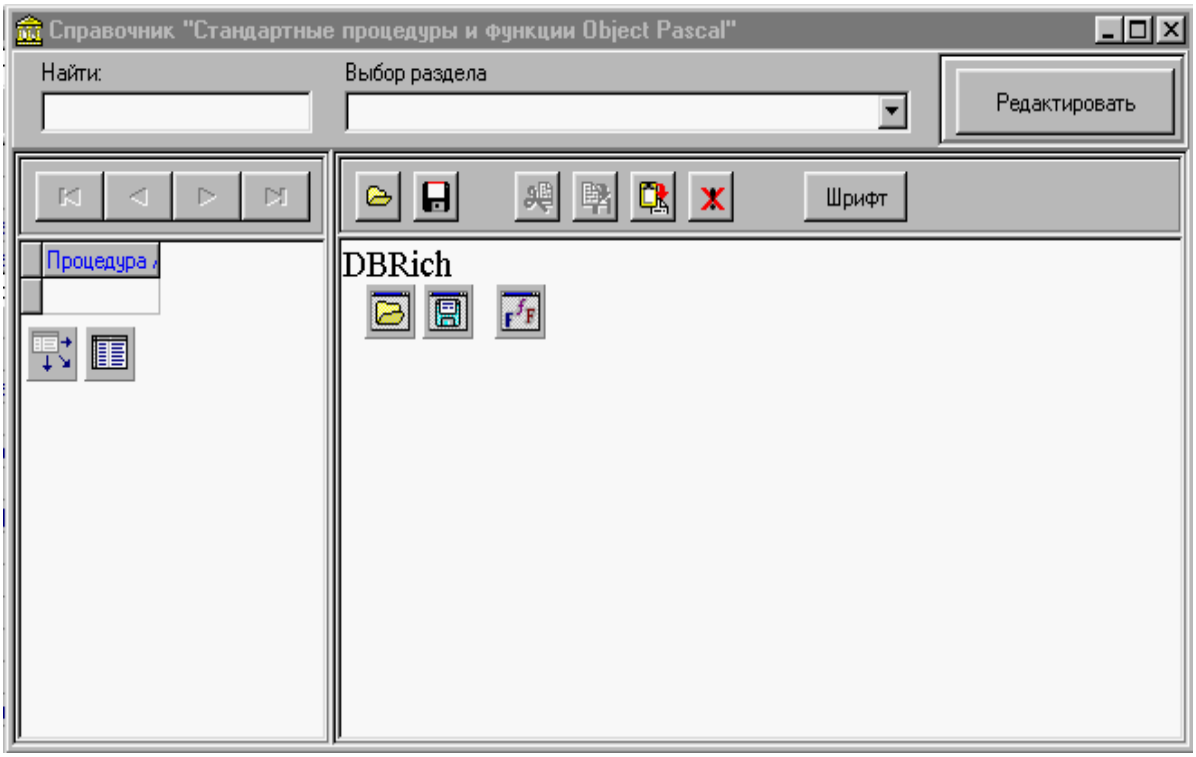

Рис.10

Таблица 2

№ п/п	Компонент	Имя	Расположение
1	TEdit	SearchEdit	Panel1
2	TComboBox	DbBox	Panel1
3	TSpeedButton	EditButton	Panel2, Panel1
4	TDBGrid	DBGrid	Panel3
5	TDBNavigator	DBNavigator	Panel4, Panel3
6	TDBRichEdit	DBRich	Panel5

№ п/п	Компонент	Имя	Расположение
7	TSpeedButton	OpenButton	Panel6, Panel5
8	TSpeedButton	SaveButton	Panel6, Panel5
9	TSpeedButton	CutButton	Panel6, Panel5
10	TSpeedButton	CopyButton	Panel6, Panel5
11	TSpeedButton	PasteButton	Panel6, Panel5
12	TSpeedButton	DelButton	Panel6, Panel5
13	TSpeedButton	FontButton	Panel6, Panel5
14	TTable	FTable	
15	TDataSource	DataSource	
16	TOpenDialog	OpenDialog	
17	TsaveDialog	SaveDialog	
18	TFontDialog	FontDialog	

5. Присвойте имена элементам управления соответственно именам, приведенным в таблице 2.
6. Выберете в списке свойств объекта **FTable** свойство **DataBaseName** и присвойте ему псевдоним Вашей базы данных. Также из списка свойства **TableName** выберете нужную таблицу, например **Funct1**.
7. Свяжите объект **DataSource** с объектом **FTable** посредством свойства **DataSet**.
8. Используя свойство **DataSource** элементов управления **DBGrid**, **DBNavigator**, **DBRich** установите связь с объектом **DataSource**. Также используя свойство **DataField** компонента **DBRich** установите его связь с полем **Description** таблицы базы данных.
9. Посредством свойства **VisibleButtons** установите видемыми соответствуюние кнопки навигатора (рис. 16, 10).
10. Откройте выбранную таблицу базы данных установив свойство **Active** объекта **FTable** в состояние **True** и настройте компонент **DBGrid** в соответствии рисунку 16.10. посредством редактора свойства **Columns**.
11. Закройте выбранную таблицу базы данных путем установки свойства **Active** в состояние **False.**
12. Занесите названия всех разделов справочника в поле списка DbBox.

Прежде чем приступить к разработке программы сформулируем задачи управления. Итак, приложение должно обеспечить два режима функционирования, а именно режим создания базы данных и режим просмотра и поиска записей. В режиме создания и редактирования базы данных, приложение должно обеспечить возможность загрузки информации (описания функций и примеров их применения) из ранее подготовленных файлов, а также возможность их последующего редактирования. В режиме просмотра и поиска записей, база данных должна быть защищена от возможной модификации в результате действий пользователя. Поиск записей по ключевому полю (FunctName) должен производится без учета регистра клавиатуры. Доступ к конкретной таблице базы данных должен выполнятся посредством выбора раздела поиска.

Итак, переключение режима функционирования приложения свяжем с кнопкой **EditButton**. Для этого настроим кнопку в режим «залипания», установив ее свойство **AllowAllUp** в **true**, и присвоив свойству **GroupIndex** значение равное единице. Переключение режимов будет состоять в изменении видимых кнопок навигатора, видимости панели инструментов текстового редактора, а также доступа к поисковому полю. Перед написанием процедуры следует установить свойство **Visible Panel6** в состояние **false** и свойство **ReadOnly DBRich** в **true**. Команда переключения режимов будет выглядеть как:

```
{***** Переключение режимов *****}
procedure TReferenceForm.EditButtonClick(Sender: TObject);
begin
 if EditButton.Down = false then
 begin
 DBNavigator.VisibleButtons  := [nbFirst,nbPrior,nbNext,nbLast];
 SearchEdit.Enabled := true;
 Panel6.Visible := false;
 DBRich.ReadOnly := true;
 end else
```

```
    begin
    DBNavigator.VisibleButtons  := [nbInsert,
    nbDelete,nbEdit,nbPost,nbCancel];
    SearchEdit.Text := ' ';
    SearchEdit.Enabled := false;
    Panel6.Visible := true;
    DBRich.ReadOnly := false;
    end;
  end;
```

Доступ к таблицам базы данных в нашей задаче должен соответствовать выбору раздела в списке разделов справочника, т.е. при выборе содержимого списка **DbBox** должна открываться соответствующая таблица. Тогда реализация команд доступа к таблицам будет иметь следующий вид:

```
{****** Выбор таблиц базы данных ******}
procedure TReferenceForm.DbBoxChange(Sender: TObject);
begin
  FTable.Close;
  FTable.DatabaseName := ExtractFilePath (Application.ExeName);
  if  DbBox.Text= 'Арифметические и математические' then
  FTable.TableName := 'Funct1';
  if  DbBox.Text= 'Операции над строками' then
  FTable.TableName := 'Funct2';
  FTable.Open;
  DBRich.DataField := 'Description';
end;
```

Таким образом, число конструкций **if** DbBox.Text= 'xxxxxx' **then** будет соответствовать количеству разделов спавочника и количеству таблиц в базе данных.

Обратите внимание на первые строки процедуры.

```
FTable.Close;
FTable.DatabaseName := ExtractFilePath (Application.ExeName);
```

В первой строке вытолняется закрытие таблицы с целью избежания исключительной ситуации при выборе следующей таблицы. Вторая строка процедуры указывает полный путь к базе данных, которая в нашем случае находится в папке исполняемого файла. Замена псевдонима на указанный путь к базе данных позволяет переносить таблицы на другой комьютер без изменения файла конфигурации ядра баз данных.

***Примечание**: При установке приложения на другой компьютер, не содержащий **среду Delphi**, необходимо установить на нем ядро управления базами данных.*

Для того, чтобы поиск по индексированному полю **FunctName** выполнялся без учета заглавных и прописных символов, обработчик события изменения данных поля ввода **SearchEdit** должен иметь следующий вид:

```
{***** Поиск записей *****}
procedure TReferenceForm.SearchEditChange(Sender: TObject);
begin
  FTable.SetKey;
  FTable.FieldByName('FunctName').AsString := SearchEdit.Text;
  FTable.GotoNearest;
end;
```

В заключение разработки наделите компонент **DBRich** необходимыми свойствами текстового редактора.

```
{***** Вырезание в буфер *****}
procedure TReferenceForm.CutButtonClick(Sender: TObject);
begin
  DBRich.CutToClipboard;
end;
```

```
{***** Копирование в буфер *****}
procedure TReferenceForm.CopyButtonClick(Sender: TObject);
begin
 DBRich.CopyToClipboard;
end;
{***** Вставка из буфера *****}
procedure TReferenceForm.PasteButtonClick(Sender: TObject);
begin
  DBRich.PasteFromClipboard;
end;

{***** Очистка окна редактора *****}
procedure TReferenceForm.DelButtonClick(Sender: TObject);
begin
   DbRich.SelectAll;
   DbRich.ClearSelection;
end;

{***** Выбор шрифта *****}
procedure TReferenceForm.FontButtonClick(Sender: TObject);
begin
    FontDialog.Font := DBRich.Font;
    if FontDialog.Execute then
    DBRich.SelAttributes.Assign(FontDialog.Font);
end;

{***** Загрузка файла в формате RTF *****}
procedure TReferenceForm.OpenButtonClick(Sender: TObject);
begin
  FTable.Edit;
  DbRich.Clear;
  if OpenDialog.Execute then
    DBRich.Lines.LoadFromFile(OpenDialog.FileName);
```

```
    FTable.Post;
end;

{***** Сохранение файла в формате RTF *****}
procedure TReferenceForm.SaveButtonClick(Sender: TObject);
begin
  if SaveDialog.Execute then
  DBRich.Lines.SaveToFile(SaveDialog.FileName);
end;

{***** Защита от возникновения исключительных ситуаций *****}
procedure TReferenceForm.DBRichSelectionChange(Sender: TObject);
begin
   CopyButton.Enabled := DBRich.SelLength > 0;
   CutButton.Enabled := CopyButton.Enabled;
end;
```

Запустите приложение на выполнение и устраните возможные ошибки. Создайте несколько пробных записей в таблицу и проверьте выполнение поиска нужной записи.

Вторая часть данного задания предполагает создание информационной части базы данных. Это большая коллективная работа, требующая значительного времени и усилий. Примеры по применению процедур и функций необходимо самостоятельно разработать в среде **Delphi** и оформить, используя встроенный редактор. Также можно применить текстовый процессор **Word** или редактор **WordPad** при этом, сохранять файлы примеров необходимо в формате **RTF**. После выполнения указанных работ занесение данного материала в базу банных не вызывает затруднений.

Пример созданной базы данных приведен на рисунке 11.

Поиск функции в справочнике

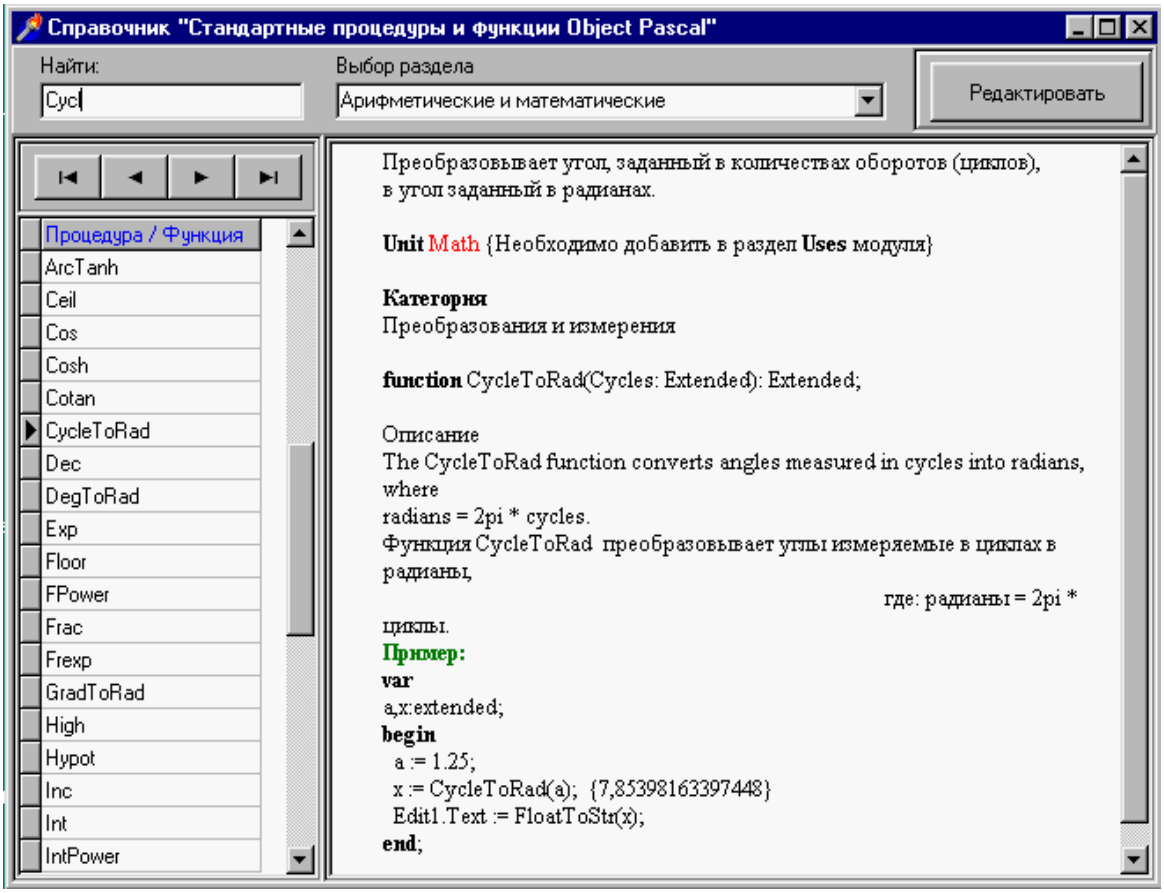

Рис.11

На основании разработанного примера базы данных можно сделать следующие выводы:

1. **На основе объектов доступа к данным TTable и TDataSource легко создавать различные приложения локальных баз данных.**
2. **Компоненты управления данными среды Delphi обладают большой гибкостью и не практически не требуют дополнительного программирования.**

Література

1. Р. Боас, М. Фервай, Х. Гюнтер, Delphi 4 Полное Руководство, – Киев.: BHV, 1998. – 448с.
2. Developer’s Guide for Delphi 3, Borland Inprise Corporation, 100 Enterprise Way, Scotts Valley, CA 95066-3249
3. Developer’s Guide for Delphi 5, Borland Inprise Corporation, 100 Enterprise Way, Scotts Valley, CA 95066-3249
4. Object Pascal Language Guide, Borland Inprise Corporation, 100 Enterprise Way, Scotts Valley, CA 95066-3249
5. Анталогия Delphi, http://www.Torry.ru

Зміст

Навчальне видання

Швачич Геннадій Григорович
Овсянніков Олександр Васильович
Кузьменко Вячеслав Віталійович
Нєчаєва Наталія Іванівна

Системи управління базами даних

Конспект лекцій
Частина 2

Тем. план 2007, поз. 151

Підписано до друку 06.02.07. Формат 60x84 1/16. Папір друк. Друк плоский. Облік.-вид. арк. 2,05. Умов. друк. арк. 2,02 Тираж 100 пр. Замовлення №11

Національна металургійна академія України
49600, Дніпропетровськ – 5, пр. Гагаріна, 4

Редакційно – видавничий відділ НМетАУ