С.В. Лаптева

ВИЗУАЛЬНОЕ ПРОГРАММИРОВАНИЕ В СРЕДЕ BORLAND C++BUILDER 6.0

Учебное пособие

Тольятти 2008

ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ
Тольяттинский государственный университет

Факультет математики и информатики
Кафедра информатики и вычислительной техники

С.В. Лаптева

ВИЗУАЛЬНОЕ ПРОГРАММИРОВАНИЕ В СРЕДЕ BORLAND C++BUILDER 6.0

Учебное пособие

Допущено Учебно-методическим объединением по образованию
в области прикладной информатики в качестве учебного пособия
для студентов высших учебных заведений, обучающихся
по специальности 351400 «Прикладная информатика (по областям)»
и другим специальностям

Тольятти 2008

УДК 681.3.06
ББК 32.973-018
Л24

Рецензенты:
кандидат технических наук, доцент, замдиректора по учебной работе
Самарского государственного аэрокосмического университета
(Тольяттинский филиал) *А.В. Очеповский*;
кандидат технических наук, доцент кафедры информатики и вычислительной
техники Тольяттинского государственного университета *В.В. Сенько*;
кандидат педагогических наук, доцент кафедры информатики
и вычислительной техники Тольяттинского государственного университета
И.П. Дудина.

Л24 Лаптева, С.В. Визуальное программирование в среде Borland C++Builder 6.0 : учеб. пособие / С.В. Лаптева. – Тольятти : ТГУ, 2008. – 196 с.

Учебное пособие знакомит с принципами визуальной работы в среде Borland C++Builder 6.0. В первой части рассматриваются основы языка C++, алгоритмические структуры и структуры данных. Вторая часть книги посвящена вопросам разработки баз данных и информационных систем.

Адресовано студентам специальности «Прикладная информатика (по областям)», аспирантам, преподавателям, а также инженерно-техническим работникам и слушателям курсов по современным системам программирования.

Рекомендовано к изданию научно-методическим советом Тольяттинского государственного университета.

ISBN 978-5-8259-0379-5

© Тольяттинский государственный университет, 2008
© С.В. Лаптева, 2008

ВВЕДЕНИЕ

Проблема быстрого создания эффективных Windows-приложений была впервые решена в среде визуального программирования Delphi, которая завоевала сердца многих начинающих и профессиональных программистов. Однако прошло некоторое время, и разработчики обратили внимание на систему, которая по принципам была похожа на Delphi, но в основе лежал язык программирования C++. Поработав в данной среде, программисты быстро оценили возможности системы, которые увеличивались с каждой версией.

В настоящее время среда быстрой разработки программ Borland C++Builder заняла достойное место среди программного обеспечения по реализации крупномасштабных информационных систем. Программный продукт компании Borland C++Builder – одна из ведущих сред разработки для создания Интернет-приложений, «настольных» и распределенных приложений, а также приложений, основанных на модели клиент/сервер. C++Builder сочетает простоту среды быстрой разработки программ, или *RAD* (*Rapid Application Deveploment – RAD*), с мощью и производительностью языка C++, совместим со стандартом ANSI.

Компания Borland предлагает мощный набор собственных расширений языка C++, обеспечивающий поддержку *VCL* (*Visual Component Library* – библиотека визуальных компонентов). Разработчик может использовать не только готовые компоненты, но и создавать собственные.

Кроме того, Borland C++Builder 6.0 является одним из лучших средств для обучения программированию на языке C++. Основная часть работы по созданию приложений выполняется в интегрированной среде разработки (*Intergrated Deveploment Environment – IDE*) C++Builder. Пакет содержит все необходимое начинающему разработчику Windows-приложений: визуальную среду разработки, программы-мастера, примеры готовых приложений и справочник.

Предлагаемое пособие снабжено богатым иллюстративным материалом, множеством демонстрационных примеров и условно разделено на две большие темы:

1. «*Компонентное программирование*» с разработкой приложений с однодокументным (Simple Document Interface, SDI) и многодокументным (Multiple Document Interface, MDI) интерфейсом.

2. «*Разработка информационных систем*» с построением баз данных, разработкой интерфейса приложения, запросов и отчетов различного уровня сложности.

Это пособие рекомендуется использовать на практических занятиях и лабораторных работах по программированию для студентов, обучающихся по специальности «Прикладная информатика (по областям)»

и смежным специаильностям, а также для всех, кто самостоятельно решил изучить основы языка С++ и систему Borland C++Builder 6.0. Пособие также может быть рекомендовано и для пользователей, знакомых с основами языка С++ и желающих научиться разрабатывать информационные системы с использованием одной из мощных сред программирования.

1. ОСНОВНЫЕ ПРИНЦИПЫ И ПРИЁМЫ РАБОТЫ В СРЕДЕ РАЗРАБОТКИ ПРОГРАММ BORLAND C++BUILDER 6.0

1.1. Начальные сведения о Borland C++Builder 6.0

Borland C++Builder 6.0 – это мощная среда быстрой разработки высокоэффективных информационных систем, включая и приложения для электронного бизнеса. Среда включает обширный набор средств, повышающих производительность труда разработчиков и сокращающих продолжительность реализации программного проекта.

Система представляет собой интегрированную среду разработки, т. е. средства редактирования исходных текстов программ, их компиляции и отладки объединены в одну программу.

Borland C++Builder 6.0 позволяет вывести программиста на качественно новый уровень разработки приложений на C++. К ключевым особенностям данной версии среды относятся следующие:

~ технология быстрой разработки программ;

~ наличие высокопромежуточного программного обеспечения для веб-служб и быстрая разработка веб-приложений;

~ эффективная работа с корпоративными базами данных;

~ полная поддержка стандартных протоколов SOAP, XML, WSDL и XSL;

~ встроенный менеджер объектных запросов Borland VisiBroker для разработки распределенных систем – CORBA-приложений;

~ и др.

Таким образом, Borland C++Builder 6.0 является надежным, высокоэффективным средством разработки крупномасштабных информационных систем различного назначения.

1.2. Элементы интерфейса Borland C++Builder

C++ Builder, как и любая программа Windows, может быть запущена на выполнение выбором соответствующей команды из главного меню Windows:

Пуск → Программы → Borland C++ Builder 6 → C++ Builder 6.

При этом загружается файл *bcb.exe*, находящийся в папке *Borland\CBuilder6\Bin*.

Интерфейс C++ Builder 6.0 представлен несколькими окнами, которые частично занимают пространство рабочего стола (рис. 1.1). В отличие от дочерних окон обычных приложений, окна среды C++Builder могут свободно перемещаться по экрану.

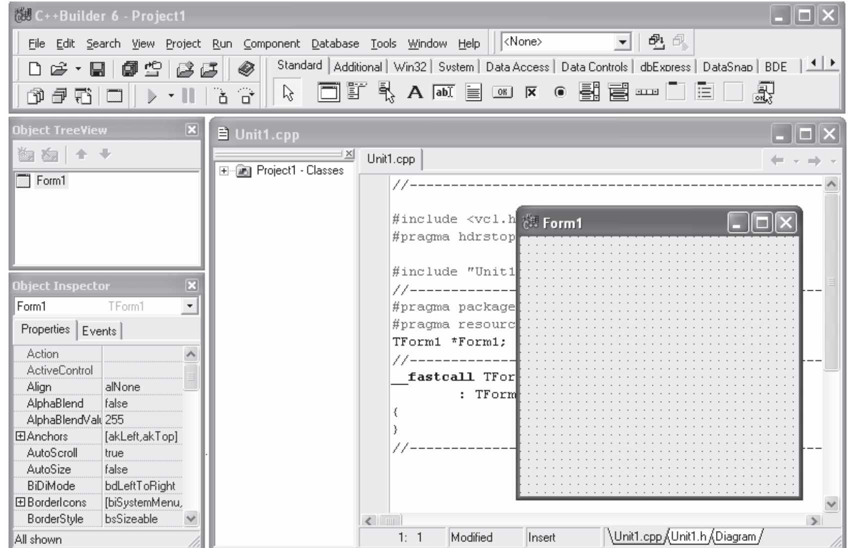


Рис. 1.1. Окно пустого проекта

Вид экрана после запуска C++Builder включает в себя пять окон:

~ главное окно *C++ Builder 6*;

~ окно стартовой формы *Form1*;

~ окно инспектора объектов *Object Inspector*;

~ окно просмотра списка объектов *Object TreeView*;

~ окно редактора кода *Unit1.cpp*, в котором оформляется текст программы.

Главное окно программы содержит меню программы, кнопки панели быстрого доступа к командам меню и палитру компонентов. Остальные окна системы предназначены для разработки приложений методом *визуального программирования*. Это окно редактора формы, окно инспектора объектов и окно просмотра иерархии объектов.

1.2.1. Главное окно проекта

В верхней части экрана расположено окно проекта с именем *Project1*, которое по умолчанию задается новому приложению. Под заголовком находится строка меню, под ней панель инструментов и палитра компонентов.

Главное меню содержит большой список команд, многие из которых «дублируются» из среды Windows и его приложений:

1. Разделы меню ***File*** (файл) позволяют создать новый проект, новую форму, открыть ранее созданный проект или форму, сохранить проекты или формы в файлах с заданными именами.

2. Разделы меню ***Edit*** (правка, редактирование) позволяют выполнять обычные для приложений Windows операции обмена с буфером Clipboard, а также дают возможность выравнивать группы размещенных на форме компонентов по размерам и местоположению.

3. Разделы меню ***Search*** (поиск) позволяют осуществлять поиск и контекстные замены в коде приложения.

4. Разделы меню ***View*** (просмотр) позволяют вызывать на экран различные окна, необходимые для проектирования.

5. Разделы меню ***Project*** (проект) позволяют добавлять и убирать из проекта формы, задавать опции проекта, компилировать проект без его выполнения и делать много других полезных операций.

6. Меню ***Run*** (выполнение) дает возможность выполнять проект в нормальном или отладочном режимах, продвигаясь по шагам, останавливаясь в указанных точках кода, рассматривая значения переменных и т. д.

7. Меню ***Component*** (компонент) позволяет создавать и устанавливать новые компоненты, конфигурировать палитру компонентов, работать с пакетами.

8. Разделы меню ***Database*** (база данных) позволяют использовать инструментарий для работы с базами данных.

9. Меню ***Tools*** (инструментарий) включает ряд разделов, позволяющих настраивать интегрированную среду разработки и выполнять различные вспомогательные программы.

10. Разделы меню ***Window*** (окно) позволяют переключаться между различными оконами, которые были открыты в текущем сеансе работы с проектом.

11. Меню ***Help*** (справка) содержит разделы, помогающие работать со встроенной в C++Builder справочной системой.

1.2.2. Кнопки панели инструментов

Кнопки панели инструментов предназначены для работы с файлами проекта. Пиктограммы приведены в табл. 1.

Таблица 1

Описание пиктограмм стандартной панели инструментов

Пиктограммы	Команда меню	Пояснение команды
	File | *New* | *Other*	Создать проект или другой компонент проекта
	File | *Open* *File* | *Reopen*	Открыть файл проекта, модуля, пакета. Кнопочка со стрелкой справа от основного изображения соответствует команде Reopen, позволяющей открыть файл из списка недавно использовавшихся

Продолжение табл. 1

Пиктограммы	Команда меню	Пояснение команды
	File | Save As *File | Save* *(CTRL+S)*	Сохранить файл модуля, с которым в данный момент идет работа
	File | Save All	Сохранить все (все файлы модулей и файл проекта)
	File | Open Project *(Ctrl+F11)*	Открыть файл проекта
	Project | Add to Project *(Shift+F11)*	Добавить файл в проект
	Project | Remove from Project	Удалить файл из проекта
	Help | C++Builder Help	Вызов страницы встроенной справки
	View | Units *(Ctrl+F12)*	Переключиться на просмотр текста файла модуля, выбираемого из списка
	View | Forms *(Shift+F12)*	Переключение на просмотр формы, выбираемой из списка
	View | Toggle Form/ Unit *(F12)*	Переключение между формой и соответствующим ей файлом модуля
	File | New Form	Включить в проект новую форму
	Run | Run (F9)	Выполнить приложение. Кнопочка со стрелкой справа от основного изображения позволяет выбрать выполняемый файл, если вы работаете с группой приложений
	Run | Program Pause	Пауза выполнения приложения и просмотр информации CPU. Кнопка и соответствующий раздел меню доступны только во время выполнения приложения
	Run | Trace Into *(F7)*	Пошаговое выполнение программы с заходом в функции
	Run | Step Other *(F8)*	Пошаговое выполнение программы без захода в функции

1.2.3. Палитра компонентов

Палитра компонентов – это набор кнопок, за каждой из которых закреплена действующая подпрограмма или элемент интерфейса Windows. Компоненты объединены в группы по общим признакам (*Standard*, *Additional* и т. д.), каждая из которых вызывается «щелчком» мыши на соответствующей закладке. Описание некоторых групп компонентов приведено в табл. 2.

Таблица 2

Описание основных вкладок (групп компонентов) палитры компонентов

Обозначение вкладки	Описание вкладки
Standard	*Стандартная*. Содержит наиболее часто используемые компоненты
Additional	*Дополнительная*. Содержит дополнительные компоненты.
System	*Системная*. Содержит такие компоненты, как таймеры, плееры и ряд других
Data Access	Доступ к данным, в C++Builder 6.0 большинство компонентов, размещающихся ранее на этой странице, перенесено на страницу BDE
Data Controls	Компоненты отображения и редактирования данных
BDE	Доступ к данным через Borland Database Engine
ADO	Связь с базами данных через Active Data Objects – множество компонентов ActiveX, использующих для доступа к информации баз данных MS OLE DB
Interbase	Прямая связь с базой данных Interbase, минуя BDE и ADO
Qreport	Компоненты для подготовки отчетов
Dialogs	Диалоги, системные диалоги типа «открыть файл» и др.
Samples	Образцы, различные, интересные, но не до конца документированные компоненты
ActiveX	Примеры компонентов ActiveX

1.2.4. Окно главной формы

Основой почти всех приложений C++Builder является форма. Ее можно понимать как типичное окно Windows. Форма является основой, на которой размещаются другие компоненты. Во время проектирования форма покрыта сеткой из точек, во время выполнения приложения эта сетка, конечно, не видна.

Сама форма – это уже готовая к исполнению программа.

1.2.5. Окно инспектора объектов

Окно инспектора кода имеет две страницы. Выше них имеется выпадающий список всех компонентов, размещенных на форме. В нем можно выбрать тот компонент, свойства и события которого необходимо установить.

Страница редактора свойств объектов (Properties) предназначена для редактирования значений свойств объектов. В терминологии визуального проектирования *объекты* – это диалоговые окна и элементы управления (поля ввода и вывода, командные кнопки, переключатели и др.). *Свойства объекта* – это характеристики, определяющие вид, положение и поведение объекта. Например, свойства *Width* и *Height* задают размер (ширину и высоту) формы, свойство *Caption* – текст заголовка.

Страница событий (Events) составляет вторую часть *инспектора объектов*. На ней указаны все события, на которые может реагировать выбранный объект.

1.2.6. Окно редактора кода

В окне *Редактор кода* следует набирать текст программы. В начале работы над новым проектом окно редактора кода содержит сформированный C++Builder шаблон программы.

1.3. Пример создания проекта «Вычисление силы тока»

Для демонстрации возможностей C++Builder и технологии визуального проектирования и событийного программирования разработаем программу, используя которую можно вычислить силу тока в электрической цепи. Сила тока вычисляется по известной формуле I = U/R, где U – напряжение источника (вольт); R – величина сопротивления (Ом). Вид диалогового окна программы во время ее работы (после щелчка на кнопке <*Вычислить*>) приведен на рис. 1.2.

Рис. 1.2. Внешний вид программы «Сила тока»

Для создания данного проекта необходимо выполнить следующие действия:

1. Создать проект командой *File | New Application*.
2. Активизировать форму *Form1* «щелчком» левой кнопкой мыши.
3. В Инспекторе объектов свойству *Caption* присвоить значение «Сила тока».
4. Для ввода информационного сообщения поместить компонент *Label* из палитры компонентов *Standard*. Для этого выполнить «двойной щелчок» мышью по пиктограмме *Label* (см. прил. 4).
5. С помощью курсора мыши задать компоненту *Label1* необходимые размеры и положение на форме. Установить значения свойств: *AutoSize – false, WordWrap – true, Caption* – «Введите напряжение и величину сопротивления, затем нажмите кнопку <Вычислить>».
6. Аналогично поместить на форму компоненты *Label2* и *Label3* для вывода информации о назначении полей ввода. С помощью курсора задать необходимые размеры и положения на форме. Установить значения свойств: *AutoSize – true, WordWrap – false, Caption* – «Напряжение (вольт)» (*Label2*) и «Сопротивление (Ом)» (*Label3*).
7. Аналогично поместить на форму компонент *Label4* для вывода результата расчета (величины тока в цепи). С помощью курсора задать необходимые размеры и положения на форме. Установить значения свойств: *AutoSize – false, WordWrap – true, Caption* – «».
8. Для ввода исходных данных поместить на форму компонент *Edit* из палитры компонентов *Standard*. Для этого выполнить «двойной щелчок» мышью по пиктограмме *Edit*. С помощью курсора мыши задать компоненту *Edit1* необходимые размеры и положение на форме. «Очистить» значение свойства *Text*.
9. Аналогично разместить на форме еще один компонент *Edit2*.
10. Для вычисления необходимо использовать компонент *Button* (кнопка). Чтобы разместить на форме кнопку, нужно «щелкнуть» на пиктограмме *Button*, а затем произвести «щелчок» в любом месте формы.
11. В Инспекторе объектов в свойстве *Caption* вместо *Button1* записать «Вычислить».
12. Для завершения программы аналогично разместить компонент *Button2* и установить значения свойства *Caption* – «Завершить».
13. По желанию можно установить цвет фона, размер и цвет шрифта компонентов, используя свойства *Color* и *Font*.

Окончательный вид формы разрабатываемого приложения приведен на рис. 1.3.

Рис. 1.3. Окончательный вид формы

14. Перейти в программный код обработки события нажатием кнопки <*Вычислить*>. Для этого необходимо произвести «двойной щелчок» на указанной кнопке, после чего на экране появится окно программного кода с мигающим курсором, дающим возможность ввести необходимый текст (рис. 1.4).

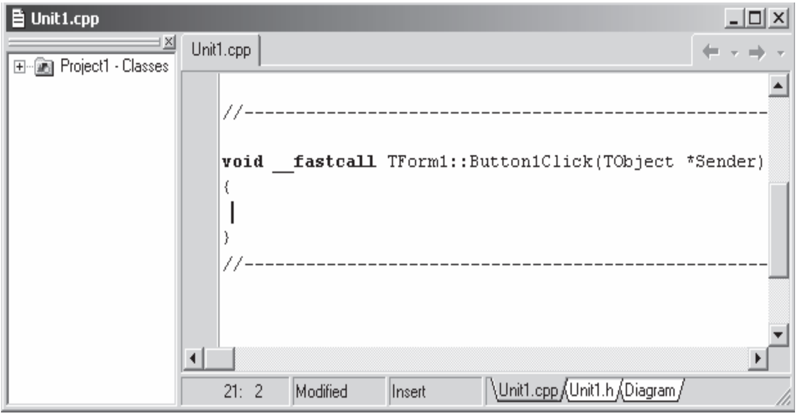


Рис. 1.4. Окно редактора кодов с процедурой обработки события нажатием кнопки <Вычислить>

15. Ввести следующий текст программы:

// КОММЕНТАРИЙ К ПРОГРАММЕ ОФОРМЛЯЕТСЯ
// С ПОМОЩЬЮ ДВОЙНОГО СЛЕША
// Объявление переменных – напряжение u, сопротивление
// r, сила тока i
float u;

```
float r;
float i;
      // Получение данных из полей ввода
u = StrToFloat(Edit1->Text);
r = StrToFloat(Edit2->Text);
      // Вычисление силы тока
i = u/r;
      // Вывод результат в поле метки
Label4->Caption = "Ток: " + FloatToStrF(i,ffGeneral,7,3);
```

16. Перейти в программный код обработки события нажатием кнопки <*Завершить*>. Для этого необходимо произвести «двойной щелчок» на указанной кнопке и в окне программного кода набрать текст программы:

```
void __fastcall TForm1::Button2Click(TObject *Sender)
{
      Form1->Close();
}
```

В результате щелчка на кнопке <*Завершить*> программа должна завершить работу. Чтобы это произошло, надо закрыть окно программы методом *Close*.

Для сохранения проекта необходимо воспользоваться командой *File→Save all,* после чего запустить программу на выполнение командой *Run*.

Функция *StrToFloat* преобразует строковые значения из полей *Edit1->Text* (содержимое поля *Edit1*) и *Edit2->Text* (содержимое поля *Edit2*) в их численные представления и сохраняет результаты в переменных *u* и *r*. Вычисленная величина силы тока выводится в поле *Label4* путем присваивания значения свойству *Caption*. Для преобразования числа в строку символов (свойство *Caption* – строкового типа) используется функция *FloatToStrF*.

Необходимо также заметить, что программный код описывается в файле с расширением*.cpp*, сам проект хранится в файле*.bpr*.

Вопросы для самоконтроля

1. Перечислить и кратко охарактеризовать окна интерфейса среды *C++Builder*, появляющиеся при загрузке системы.
2. Охарактеризовать назначение окна формы. В чем его отличие от самой формы?
3. С какой целью используется компонент *Label*? Какие его свойства используются в программе?

4. Каково назначение компонента *Edit*? С помощью какого свойства осуществляется доступ к его содержимому во время разработки и работы программы?
5. Каким образом можно назначить кнопке выполнение тех или иных действий?
6. Какими способами можно осуществить закрытие формы во время работы программы?
7. Каково назначение функции *StrToFloat* при обработке строковых значений?

2. УПРАВЛЯЮЩИЕ СТРУКТУРЫ С++

Любое выражение, завершающееся точкой с запятой, рассматривается в С++ как *оператор*. Например,

```
a=b*c;              // операция присваивания
i++;                // операция инкремента
function(int x);    // операция вызова функции
```

Условие в языке программирования – это выражение логического типа (*boolean*), которое может принимать одно из двух значений: «истина» (*true*) или «ложь» (*false*).

Из простых условий при помощи логических операторов можно строить сложные условия. В качестве логических операторов используются: «логическое И» (**&&**), «логическое ИЛИ» (**||**), «отрицание» (**!**).

2.1. Реализация выбора

2.1.1. Оператор ветвления

Условный оператор позволяет проверить некоторое условие и в зависимости от результатов проверки выполнить ту или иную последовательность действий.

Структура условного оператора имеет следующий вид:

```
if (условие) // комментарий в программе
{
 // команды, которые выполняются, если условие истинно
}
else
{
 // команды, которые выполняются, если условие ложно
}
```

В качестве примера использования оператора *IF* рассмотрим программу вычисления стоимости телефонного разговора. Известно, что стоимость разговора по телефону в воскресные дни ниже, чем в будни.

Интерфейс формы разрабатываемого приложения имеет следующий вид (рис. 2.1):

Рис. 2.1. Диалоговое окно «Стоимость разговора»

Необходимо помнить, если день недели – суббота или воскресенье, то стоимость уменьшается на величину скидки. Цена минуты разговора и величина скидки задаются в тексте программы как константы.

Для ввода исходных значений (длительность разговора, номер дня недели) используются поля редактирования (компонент *Edit*), а для вывода результата и пояснительного текста – метки (*Label*). Для выполнения вычислений и выхода с формы используются кнопки: *обычная* (*Button*) и *событийная* (*BitBtn*).

В табл. 3 представлено описание компонентов формы приложения.

Таблица 3

Компоненты формы приложения

Компонент	Назначение
Edit1	Поле ввода длительности разговора в минутах
Edit2	Поле ввода номера дня недели
Label1, Label2	Текстовая информация, служащая для ввода пояснительного текста о назначении полей ввода
Label3	Текстовая информация, служащая для вывода результата вычисления – стоимости разговора
Button1	Кнопка, активизирующая процедуру вычисления стоимости разговора
BitBtn1	Кнопка, активизирующая процедуру, которая закреплена *автоматически* за данной событийной кнопкой

Программа производит вычисления после нажатия на кнопку <*Вычислить*>. В данной ситуации генерируется событие *OnClick*, обрабатывающееся функцией *TForm1.Button1Click*. Для перехода в программный код данного события необходимо произвести «двойной щелчок» по кнопке, после чего указатель будет установлен внутри соответствующей функции.

Для кнопки *BitBtn1* (вкладка *Additional*, компонент *BitBtn*) необходимо установить в Инспекторе объектов:

~ свойство *Kind*, которое позволяет задать для данной (особой) кнопки уже существующую процедуру, например, закрытие формы через константу *bkClose*;

~ свойство *Caption*, которому можно задать значение «*Выход*», поставив перед буквой «В» символ амперсанда (**&**); данный символ позволяет сделать символ, стоящий после него, «горячей клавишей».

В листинге 2.1 приведен полный код файла *u_lab2_1.cpp*. В описательной части структуры программы объявлены и/или инициализированы глобальные переменные и константы, такие как *Tim, Summa, Day* и *PAY, DISCOUNT* соответственно.

Листинг 2.1. Вычисление стоимости телефонного разговора

```
#include <vcl.h>
#pragma hdrstop

#include "u_lab2_1.h"
    //---------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;

    // цена одной минуты разговора 0.45 рубля
float PAY=0.45;
    // скидка 20%
int DISCOUNT=20;
    // длительность разговора, стоимость разговора
float Tim, Summa;
    // день недели
int Day;
    //---------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
: TForm(Owner)
{
}
    //---------------------------------------------------------
void __fastcall TForm1::Button1Click(TObject *Sender)
{
    // получить исходные данные (время) из текстового пола
 Tim= StrToFloat(Edit1->Text);
    // получить день недели из текстового поля
 Day=StrToInt(Edit2->Text);
    // вычислить стоимость разговора
 Summa=PAY*Tim;
    // Если день недели – суббота или воскресенье,
    // то уменьшить стоимость на величину скидки
 if ((Day==6) || (Day==7))
 {
 Summa=Summa*(100 – DISCOUNT) / 100;
 }
    // Вывод результата вычисления в текстовое поле,
    // предварительно преобразовав вещественное
    // значение в строковое
Label3->Caption="К оплате "+FloatToStr(Summa)+" руб.";
}
```

В приведенном фрагменте описано и подключение различных стандартных модулей, включая и файл *u_lab2_1.h*. Файл с таким расширением получил название *интерфейсного*, так как в нем описывается (по умолчанию) интерфейс формы – компоненты и их свойства.

2.1.2. Оператор выбора

Оператор *SWITCH* является обобщением оператора *IF* и дает возможность выполнить одно из нескольких действий в зависимости от значения *выражения*.

Формат оператора представляется следующим образом:

```
switch (выражение)
{
    case значение_1: {группа_команд_1; break};
    case значении_2: {группа_команд_2; break};
    …
    case значение_N: {группа_команд_N; break};
    default: {группа_команд; break};
}
```

где от значения *выражения* зависит, какая группа команд будет выполняться; а *значение_i* представлено константами, разделенными запятыми.

Приведем пример использования *SWITCH:*

```
switch (n_day)
{
case 6: day= "Суббота!"; break;
case 7: day= "Воскресенье!"; break;
default: day= "Рабочий день"; break;
}
```

В качестве примера использования оператора *SWITCH/CASE* рассмотрим программу, которая переводит вес из фунтов в килограммы, учитывая, что в разных странах фунт «весит» по-разному. В диалоговом окне программы, изображенном на рис. 2.2, для выбора страны используется список.

Для оформления списка на форме используется компонент *ListBox*, значок которого располагается на вкладке *Standard*.

В табл. 4 приведены свойства компонента *ListBox*.

Рис. 2.2. Диалоговое окно программы «Перевод веса»

Таблица 4

Свойства компонента ListBox

Свойство	Определяет
Name	*Имя компонента*. В программе используется для доступа к свойствам компонента
Items	Элементы списка
ItemIndex	*Номер выбранного элемента списка*. Номер первого элемента списка равен нулю
Font	*Шрифт*, используемый для отображения элементов списка

Свойство *Items* содержит элементы списка. Значения, выводимые в поле списка, могут быть сформированы во время создания формы приложения или динамически (во время работы программы).

Для формирования списка строк во время создания формы приложения необходимо выделить размещенный на форме компонент *ListBox*, в окне *Object Inspector* выбрать свойство *Items* и «щелкнуть» на кнопке запуска *Редактора списка строк* (рис. 2.3).

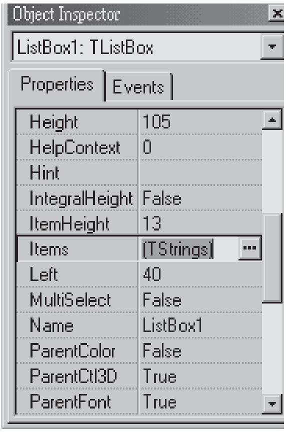


Рис. 2.3. Окно Object Inspector

В открывшемся диалоговом окне *String List editor* (редактор списка строк) необходимо набрать список, каждый элемент которого записывается в отдельной строке (рис. 2.4). Ввод каждого элемента списка должен заканчиваться нажатием клавиши <*Enter*>. После ввода всех элементов списка следует нажать кнопку <*Ок*>.

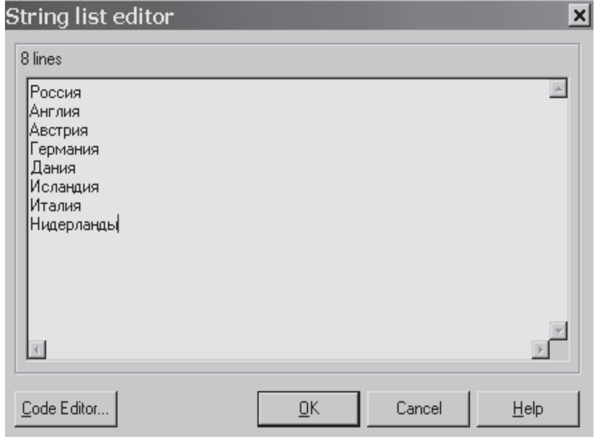


Рис. 2.4. Окно редактора списка строк

Свойство *ItemIndex* характеризует номер выбранного элемента списка. Если ни один из элементов списка не выбран, то его значение равно –1.

В табл. 5 описаны компоненты формы приложения.

Таблица 5

Компоненты формы приложения

Компонент	Содержит
ListBox1	Список стран. Выбирается страна, для которой выполняется пересчет денежной единицы
Edit1	Поле ввода числового значения – веса в фунтах
Label1, Label2, Label3	Текстовая информация, служащая для вывода пояснительного текста о назначении полей ввода
Label4	Текстовая информация, служащая для вывода результата пересчета денежной единицы
Button1	Кнопка, служащая для активизации процедуры пересчета веса из фунтов в килограммы
BitBtn1	Кнопка, служащая для активизации процедуры, которая закреплена *автоматически* за данной событийной кнопкой

Процедура пересчета денежной единицы, которая выполняется в результате нажатия кнопки <*Вычислить*>, умножает вес в фунтах на коэффициент, равный весу в килограммах одного фунта. Значение коэффициента зависит от номера выбранного элемента списка, т. е. от страны.

В листинге 2.2 приводятся две функции, которые обрабатывают два события: первое событие «срабатывает» при создании формы (метод *FormCreate*), а второе – при нажатии на кнопку <*Вычислить*>. Для перехода в программный код первого события необходимо сделать форму активной (в «Инспекторе объектов» текущим компонентом является форма), затем перейти на вкладку *Events* окна инспектора объектов и произвести «двойной щелчок» в зоне значения события *OnCreate*. После данного действия система переходит в программный код метода *FormCreate*.

Эта процедура может быть использована для инициализации переменных программы, в том числе и для добавления элементов в список. В приведенном тексте программы закомментированный фрагмент описывает способ добавления элементов в компонент программным путем.

Рекомендуется выполнить приведенный пример, а затем изменить программу так, чтобы в ней для заполнения списка использовалось событие *OnCreate* и метод *ListBox1.Items.Add(‘ ‘).*

Листинг 2.2. Пересчет веса (из фунтов в килограммы)

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
      // Данные команды не используются, так как значения списка
      // заданы инструментальным способом – через свойства
      // компонента ListBox; в противном случае можно задать
      // программным путем, как описано ниже

// ListBox1->Items->Add("Россия");
// ListBox1->Items->Add("Англия");
// ListBox1->Items->Add("Австрия");
// ListBox1->Items->Add("Германия");
// ListBox1->Items->Add("Дания");
// ListBox1->Items->Add("Исландия");
// ListBox1->Items->Add("Италия");
// ListBox1->Items->Add("Нидерланды");

      // Установка первого элемента списка в качестве текущего
      // (индексация начинается с нуля)
ListBox1->ItemIndex=0;
}
      //-----------------------------------------------------------
void __fastcall TForm1::Button1Click(TObject *Sender)
{
      // объявление переменных, характеризующих количество фунтов,
      // количество килограммов, коэффициент веса
float funt, kg, k;
      // установка коэффициента веса в зависимости от страны
switch (ListBox1->ItemIndex)
{
      case 0: k=0.4059; break;
      case 1: k=0.453592; break;
      case 2: k=0.56001; break;
      case 6: k=0.31762; break;
      default: k=0.5; break;
}
      // считывание из текстового поля количества фунтов
funt=StrToFloat(Edit1->Text);
      // вычисление кг
kg=k*funt;
      // вывод в текстовое поле результата (вещественное число
```

```
        //предварительно преобразовано в строковое значение
  Label3->Caption= Edit1->Text + " фунт(а/ов) – это " +
                              FloatToStrF(kg,ffFixed,6,3) + " кг.";
}
```

Функция *FloatToStrF* подробно описана в предыдущем пункте.

2.2. Реализация циклов

Циклы реализуются в программе с использованием операторов: *FOR, WHILE* и *DO WHILE.*

2.2.1. Оператор цикла с известным количеством повторений

Оператор *FOR* используется в том случае, если некоторую последовательность команд необходимо выполнить несколько раз, при этом число повторений заранее известно. Такой цикл называется *циклом с известным количеством повторений*.

Формат оператора представляется следующим образом:

```
for (инициализация; выражение; модификации)
{
    группа_команд;
}
```

где *инициализация* – задание некоторой переменной-счетчику начального значения, *выражение* – запись, описывающая условие выполнения цикла, *модификации* – операция, выполняющаяся после каждой итерации цикла.

В качестве примера использования оператора *FOR* рассмотрим программу, которая вычисляет сумму первых 10 элементов ряда: 1+1/2+1/3+... (значение *i*-го элемента ряда связано с его номером формулой 1/i). Диалоговое окно программы (рис. 2.5) должно содержать, по крайней мере, три компонента: поле метки (*Label1*), командную кнопку (*Button1*) и событийную кнопку (*BitBtn1*).

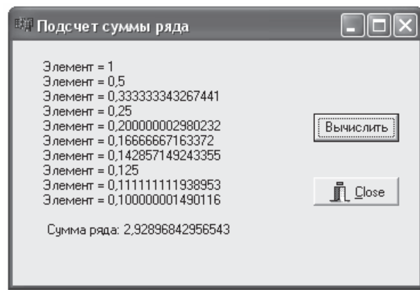


Рис. 2.5. Программа «Подсчет суммы ряда»

Вычисление суммы ряда и вывод результата выполняет процедура обработки события *OnClick*, текст которой приведен в листинге 2.3.

Листинг 2.3. Вычисление суммы ряда

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
      // объявление переменных: элемента ряда и количества элементов
float elem, N;
       // инициализация переменной суммы элементов ряда
float summ=0;
      // организация цикла с заданным количеством повторений – 10
for (N=1; N<=10; N++)
{
      // вычисление очередного элемента ряда
elem = 1/N;
      // вычисление частичной суммы ряда
summ = summ + elem;
       // вывод в текстовое поле промежуточного результата
Label1->Caption= Label1->Caption + "Элемент = " +
                                        FloatToStr(elem)+ char(13);
}
      // вывод в текстовое поле результата, который предварительно
      // был переведен из вещественного значения в строковое
Label1->Caption= Label1->Caption + "\n \n Сумма ряда: " +
                                              FloatToStr(summ);
}
```

В данном листинге рассматривается переход на новую строку двумя способами:

1) с помощью команды *char*(13);

2) с помощью кода *n*, расположенном внутри строкового значения.

Первый способ удобнее использовать при «склеивании» строковых значений, а второй – в случае, если строка представляет собой единое целое и ее не нужно разбивать на части. Необходимо заметить, что код '\n' часто встречается в других языках программирования, например, в Прологе.

2.2.2. Организация цикла с предусловием

Оператор *WHILE* используется в том случае, если некоторую последовательность команд надо выполнить несколько раз, причем необходимое число повторений во время разработки программы неизвестно и может быть определено только во время ее работы, т. е. в процессе

вычисления. Типовыми примерами использования цикла *WHILE* являются вычисления с заданной точностью, поиск в массиве или в файле.

Формат оператора представляется следующим образом:

```
while (условие)
{
    группа_команд;
}
```

где *условие* – это выражение логического типа, определяющее условие выполнения цикла.

Группа команд будет повторяться, пока условие истинно.

В качестве примера использования оператора *WHILE* рассмотрим программу, которая вычисляет значение числа π с точностью, задаваемой пользователем во время работы программы.

В основе алгоритма вычисления лежит тот факт, что сумма ряда приближается к значению π/4 при достаточно большом количестве членов ряда.

$$1 - 1/3 + 1/5 - 1/7 + 1/9 - ... + (-1)^{n+1} * 1/(2n-1) + ...$$

Вид диалогового окна программы во время ее работы приведен на рис. 2.6. Пользователь вводит точность вычисления в поле ввода (*Edit1*). После нажатия командной кнопки <*Вычислить*> (*Button1*) программа вычисляет значение числа π и выводит результат в поле метки (*Label2*). Кнопкой <*Close*> (*BitBtn1*) закрывается форма.

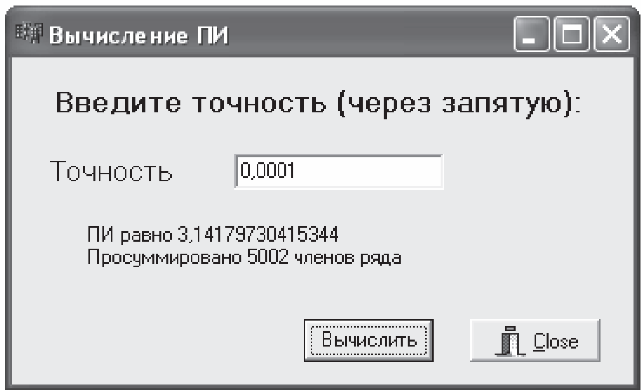


Рис. 2.6. Диалоговое окно программы “Вычисление ПИ”

Как и в предыдущих примерах, основную работу в задаче выполняет процедура обработки события *OnClick*, текст которой приведен в листинге 2.4.

Листинг 2.4. Процедура вычисления числа π

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
      // объявление переменных: π , точность, элемент и
      // количество итераций N
float pi, t, elem, n;
      // объявление промежуточного значения: количество
      // итераций N целого типа
short nn;
      // задание начального значения для переменной π  и
      // количества итераций N
pi=0;
n=1;
      // считывание вещественного значения из текстового поля
t = StrToFloat(Edit1->Text);
      //задание начального значения для элемента
elem=t+1;
      // пока выполняется условие, будет выполняться и
      // последовательность действий
while (elem >= t)
{
       // вычисление очередного элемента итерации
elem = 1/(2*n-1);
       // преобразование вещественного числа N в целое число NN
nn = FormatFloat("0",n).ToInt();
       // если число четное, то выполняется вычитание; в
       // противном случае – сложение
if (nn%2==0)
{ pi-=elem; }
else
{ pi+=elem; }
       // переход на следующую итерацию
n++;
}
      // получение истинного значения числа π
pi = pi*4;
      // вывод в текстовое поле результата, который
      // предварительно преобразован из вещественного значения
      // в строковое
Label2->Caption= "ПИ равно " + FloatToStr(pi)+ char(13) +
                  "Просуммировано " + FloatToStr(n)+ " членов ряда";
}
```

В программном коде приводится команда

nn = FormatFloat(“0”,n).ToInt();

результатом выполнения которой является присвоение переменной *nn* целого числа.

Метод *FormatFloat* класса *AnsiString* позволяет представить числовое (вещественное) значение в определенном формате, при этом возвращаемое значение представляется как строковое *AnsiString*. Синтаксис данного метода следующий:

FormatFloat (формат, числовое_значение)

В качестве формата в данном методе записан «0»; это значит, что при преобразовании все цифры, включая и 0, будут сохранены на своих позициях.

Метод *ToInt*() относится к классу *AnsiString* и используется в случае, если необходимо преобразовать строковое значение в целое число. Запись двух методов через символ «точка» означает, что система последовательно реализует первый метод, а затем второй.

2.2.3. Организация цикла с постусловием

Оператор *DO-WHILE*, как и оператор *WHILE*, используется в том случае, если необходимо организовать цикл, при этом необходимое число повторений во время разработки программы неизвестно и может быть определено только во время работы программы.

Формат оператора представляется следующим образом:

```
do
{
      // группа команд;
 } while (условие);
```

где *условие* – это выражение логического типа, определяющее условие выполнения цикла.

Приведем пример записи оператора *DO-WHILE*:

```
str = “”;
do
{
      str = str + IntToStr(i) + ” “;
      i:=i+1;
} while (i<10);
```

Команды цикла, находящиеся между *DO* и *WHILE*, выполняются до тех пор, пока *условие* истинно.

2.2.4. Оператор безусловного перехода GOTO

Формат оператора представляется следующим образом:

goto метка;

где *метка* – это идентификатор, находящийся перед фрагментом программы, которая должна быть выполнена после оператора *GOTO*.

Метка не описывается в каком-либо разделе программы.

В тексте программы метка ставится перед последовательностью команд, которые должны быть выполнены после оператора *GOTO*. После метки ставится двоеточие.

В качестве примера рассмотрим программу, которая проверяет, является ли число, введенное пользователем, *простым* (делящимся только на единицу и само на себя). Проверить, является ли число N простым, можно делением числа N на два, на три и т. д. до N и проверкой остатка после каждого деления.

Форма приложения «Простое число» изображена на рис. 2.7.

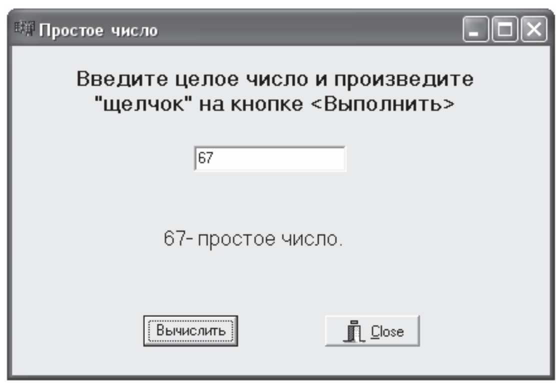


Рис. 2.7. Форма приложения «Простое число»

В табл. 6 перечислены компоненты формы приложения.

Таблица 6

Компоненты формы приложения

Компонент	Назначение
Edit1	Поле ввода исходного числа
Label1	Текстовая информация, служащая для ввода пояснительного текста о назначении поля ввода
Label2	Текстовая информация, служащая для вывода результата проверки
Button1	Кнопка, активизирующая процедуру проверки числа
BitBtn1	Кнопка, служащая для активизации процедуры, которая закреплена *автоматически* за данной событийной кнопкой

В программе используется функция *ShowMessage*, которая позволяет вывести указанное сообщение в информационное окно.

Текст программы приведен в листинге 2.5.

Листинг 2.5. Процедура проверки простого числа

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
      // объявление переменных: элемент n, делитель d и частное r
short n, d, r;
      // считывание значения элемента из текстового поля
n=StrToInt(Edit1->Text);
      // если введенное число равно 1, то программа прекращает
      // работать
if (n<=1)
{
ShowMessage("Число должно быть больше единицы!");
Edit1->Text=" ";
       // переход на метку bye
goto bye;
}
      // задание начального значения делителю
d=2;
      // выполняется последовательность команд, пока выполняется
      // условие
do
{
      // если деление по модулю d не равно 0, то делитель
      // увеличивается на 1; как только делимое и делитель
      // становятся равными, значит система достигла ситуации:
      // нет делителей для числа n
r=n%d;
if (r!=0) d++;
} while (r!=0);
Label2->Caption=Edit1->Text;
if (d==n)
{
Label2->Caption=Label2->Caption+ "- простое число.";
}
else
{
Label2->Caption=Label2->Caption+ "- обычное число.";}
       // пустая метка, которая не требует никаких действий
bye:;
}
```

В литературе по программированию можно встретить суждения о недопустимости применения оператора *GOTO*, поскольку это приводит к запутанности программ. Однако с категоричностью таких утверждений согласиться нельзя. В некоторых случаях использование оператора *GOTO* вполне оправдано. В данной ситуации можно использовать условие, но оператор GOTO является простейшим решением для данной задачи.

Вопросы для самоконтроля

1. Каков принцип работы условного оператора *if*?
2. Какой оператор позволяет выполнить одно из нескольких действий в зависимости от результата вычисления выражения?
3. Каким образом работает оператор цикла *while*?
4. В чем различие операторов *do while* и *while*?
5. Каким образом осуществляется описание и использование меток в программах?
6. Каково назначение компонента *ListBox*? Какие характеристики этого компонента определяют свойства *Name, Items, ItemIndex*?
7. При возникновении каких событий происходит вызов процедур *FormCreate*, *Button1Click*?

3. ПРИНЦИПЫ И ПРИЁМЫ СОЗДАНИЯ ПРОГРАММЫ, УПРАВЛЯЕМОЙ СОБЫТИЯМИ

Принципы и приемы создания программы, управляемой событиями, можно рассмотреть на примере разработки контрольного теста по информатике.

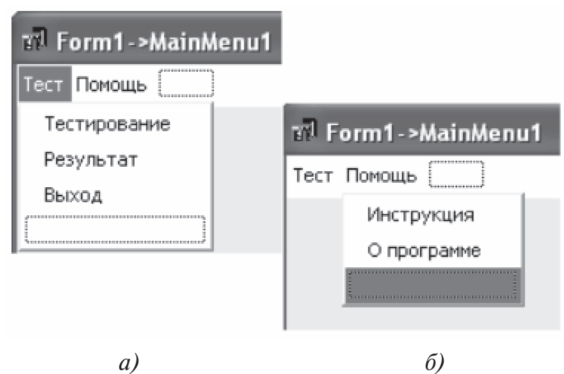
Программа будет иметь графический интерфейс, включающий в себя меню команд и диалоговые окна: с информацией об авторе теста; инструкцией по использованию; вопросами и результатом тестирования.

3.1. Разработка главной формы «Тест по информатике»

При создании пустого проекта (приложения) система создает по умолчанию форму, которая является главной. В случае изменения статуса формы, например, сделать главной форму-заставку, пользователь может воспользоваться командой *Project | Options | Main Form*.

После создания формы «Тест по информатике» необходимо оформить на ней пункты меню с использованием компонента *MainMenu*. Для этого необходимо выполнить следующие действия:

1. Выбрать пиктограмму *MainMenu*, расположенную на стандартной палитре компонентов, и установить компонент *MainMenu* на форме щелчком мыши.
2. Расположить появившийся на форме компонент в одном из ее верхних углов и произвести «двойной щелчок» по пиктограмме. При этом на экране появится окно с заголовком *Form1.Mainmenu1.*



а) *б)*

Рис. 3.1. Команды пунктов меню «Тест» и «Помощь»

3. В *Инспекторе объектов* в свойстве *Caption* ввести значение первого элемента меню «Тест», которое запишется в синем прямоугольнике, расположенном в левом верхнем углу полосы меню (рис. 3.1*а*).

4. Нажатием клавиши <*Enter*> перейти к записи первой команды вертикально спускающегося меню.

5. В свойстве *Caption* ввести значение «Тестирование» и нажать <*Enter*>.

6. Аналогично для следующих команд пункта «Тест» создать команды «Результат» и «Выход».

7. Создать следующий элемент меню – «Помощь» (рис. 3.1*б*). Для этого активизировать мышью прямоугольник, находящийся справа от команды меню «Тест», а затем свойству *Caption* задать значение «Помощь».

8. Создать подменю пункта «Помощь», введя команды «Инструкция» и «О программе».

9. Сохранить файл модуля *Unit1.cpp* под именем *Mainform.cpp*, а файл проекта *Project1.bpr* – под именем *Test.bpr* в каталоге пользователя.

Разрабатываемое приложение состоит из главной формы, в которую встроено меню. Команды «*О программе*» и «*Инструкция*» будут вызывать дополнительные формы в виде информационных окон. Готовые шаблоны для них могут быть выбраны из диалогового окна *New Items*, которое вызывается последовательностью команд *File* → *New* → *Other*...

3.2. Создание окон «О программе», «Инструкция»

Создание окна «О программе» основано на форме *AboutBox* и подключено к главной форме *Mainform*.

Создание такой формы осуществляется командой меню *File*→*New*→*Other*..., которая вызывает диалоговое окно *New Items*. В разделе *Form* необходимо выбрать форму *About Box*. На экране появится стандартное окно *Aboutbox* (рис. 3.2), содержащее следующие элементы:

~ графический рисунок, который вставляется в форму с помощью кнопки *Image* из палитры компонентов *Additional*;

~ текстовые компоненты (*Label*);

~ кнопку <OK>.

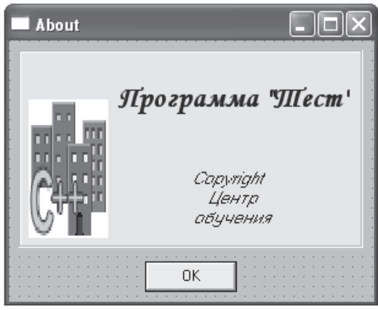


Рис. 3.2. Окно «О программе»

Работа с графическим объектом формы «О программе» заключается в следующем:

1. Активизировать компонент *Image* (рисунок) щелчком мыши. Растянуть границы компонента до размеров вставляемого графического объекта.
2. Задать свойству *Stretch* (масштабирование) компонента *Image* значение *True*. Рисунок «подстроится» под размеры рамки.
3. В «Инспекторе объектов» в свойстве *Picture* щелкнуть по многоточию. Появится окно редактора рисунка *Picture Editor* (рис. 3.3).
4. В окне редактора выполнить команду *Load*. Появится окно *Load Picture*.
5. Выбрать файл по желанию и нажать <Ок>. Выбранный рисунок будет добавлен на форму в выделенную рамку.

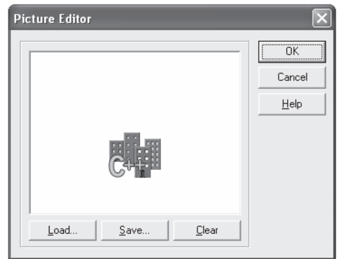


Рис. 3.3. Окно редактора рисунка

6. Удалить с формы текстовые компоненты *Version* и *Comments*.
7. Для компонентов *Product Name* и *Copyright* в «Инспекторе объектов» установить для свойства *AutoSize* значение *false* (это свойство устанавливает ширину *Label* по длине вводимой строки). Свойству *Alignment* установить значение *taCenter* для размещения текста по центру компонента. Свойству *WordWrap* установить значение *true*.
8. Для компонента *Product Name* в свойстве *Caption* записать значение «Тест по информатике». Для компонента *Copyright* в свойстве *Caption* записать значение «Copyright by...» с указанием своей фамилии.
9. Самостоятельно задать компоненту *Label* необходимые размеры. Установить цвет фона, размер, вид и цвет шрифта.
10. Сохранить созданный модуль. Для этого нажать кнопку <*Save Project*> (сохранить файл проекта) в панели инструментов; в окне сохранения модулей ввести *about.cpp*, при этом проект и остальные модули сохраняться автоматически.

Для подключения программного модуля формы «О программе» к главной программе «Тест по информатике» необходимо выполнить следующие действия:

1. Сделать текущим окно главной формы «Тест по информатике».

2. Открыть меню «Помощь» и щелкнуть мышью на команде «О программе». При этом откроется окно «Редактор кода программы» с процедурой обработки щелчка мыши на команде меню «О программе».

3. Подключить *about.h*. Для этого надо перейти в начало кода и ниже *#include “mainform.h”* подключить директиву *#include “about.h”*.

4. Вернуться в место мигания курсора, ввести команду

AboutBox->ShowModal();

Эта команда при выполнении программы будет выводить на экран окно «О программе».

Аналогичным способом разрабатывается форма «Инструкция». Ее создание основывается на форме *Tabbed Pages* и подключении его к главной программе (рис. 3.4).

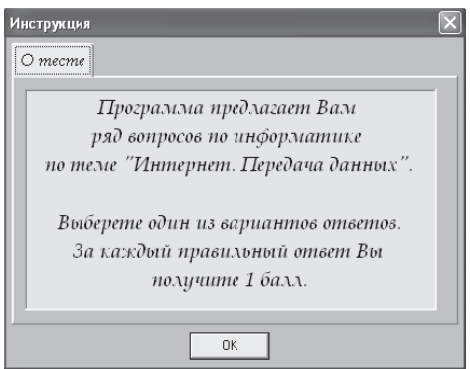


Рис. 3.4. Окно инструкции

Для решения данной задачи необходимо выполнить следующие действия:

1. Открыть диалоговое окно *New Items* командой *File→New→Other...* и в разделе *Form* выбрать форму *Tabbed pages*.

2. В свойстве *Caption* установить значение «Инструкция».

В многостраничном диалоговом окне каждая страница вызывается «щелчком» мыши на ее закладке. В первоначальном виде окно *Tabbed pages* имеет три страницы с закладками. Количество закладок можно изменять. Закладкам можно присвоить другие имена: например, названия «Страница 1», «Страница 2» и «Страница 3» соответственно.

3. Активизировать *TabSheet1* из списка объектов в окне инспектора объектов.

4. Задать свойству *Caption* значение «О тесте».

5. По желанию можно удалить остальные страницы. Для этого надо активизировать ненужную страницу, нажать по ней правой кнопкой мыши и выбрать команду *Delete Page*.

6. Заполнить страницу текстом инструкции. Для этого необходимо на форму разместить компонент многострочного текста, выбрав в палитре компонентов *Standard* элемент управления *Memo* «двойным щелчком» мыши. Растянуть до нужных размеров и в свойстве *Lines* в появившемся окне ввести текст инструкции.

7. Установить цвет и шрифт для текста.

8. Удалить с формы кнопки <*Cancel*> и <*Help*>.

9. Сохранить данный модуль под именем *instruct.cpp*.

10. Открыть окно *Редактор кода* главной программы с процедурой обработки «щелчка» мыши на команде меню «Инструкция».

11. Подключить директиву *#include "instruct.h"*.

12. Вернуться в место мигания курсора, ввести команду

PagesDlg->ShowModal();

Эта команда при выполнении программы будет выводить на экран окно «Инструкция».

13. Проверить работоспособность команды «Инструкция».

3.3. Разработка формы «Тестирование»

На основе той же формы Tabbed pages создадим диалоговое окно с вопросами теста. Для этого необходимо выполнить следующие действия:

1. Создать форму аналогично предыдущему примеру.

2. Удалить на ней кнопки <*Cancel*> и <*Help*>.

3. Задать заголовок и идентификатор объекту формы: свойству *Caption* присвоить значение «Тестирование».

4. Добавить четыре страницы к трем существующим. Для этого вызвать контекстное меню заголовка страницы и в всплывающем списке выбрать команду *New Page* (новая страница).

5. Присвоить каждой вкладке страничного блока названия «Вопрос № 1», «Вопрос № 2» и т. д.

На каждой странице нужно разместить вопрос и варианты ответов. Текст вопроса будет располагаться на цветной панели в рамке. Для этого можно воспользоваться компонентами Panel и Label (рис. 3.5).

6. Активизировать страницу для первого вопроса.

7. На странице разместить компонент в виде панели. Для этого из палитры компонентов *Standard* добавить на первую страницу компонент *Panel* и «растянуть» его мышью до необходимых размеров.

8. В *Инспекторе объектов* свойству *Color* задать любой цвет панели.

9. Установить рамку для панели. Для этого свойству *Bevelinner* (внутренняя фаска) присвоить значение *Lowered* (опущенная), а свойству *Bevelouter* (внешняя фаска) – значение *Raised* (поднятая).

10. На готовую панель добавить компонент *Label*, «растянуть» его на всю панель. Для центрирования текста свойству *Autosize* установить значение *false*, а свойству *Alignment* – *taCenter*.

11. Присвоить свойству *Caption* текущей страницы значение «Специальное устройство, способное передавать по обычной телефонной линии цифровые сигналы, – это...».

12. Установить цвет, вид и размер шрифта.

На диалоговой странице разместить несколько вариантов ответа, среди которых тестируемый должен будет выбрать правильный ответ. Для альтернативного выбора в ряду элементов в палитре Standard существует компонент RadioGroup, который объединяет несколько радиокнопок (зависимых переключателей).

13. Добавить на форму «Тестирование» на страницу первого вопроса компонент *RadioGroup* и в его свойстве *Caption* задать заголовок «Варианты ответов».

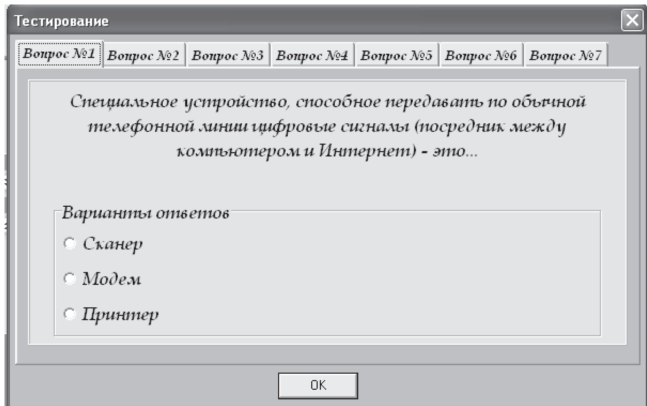


Рис. 3.5. Окно «Тестирование»

14. В свойстве *Items* (элементы) «двойным щелчком» вызвать окно *String List Editor* («Редактор списка строк»).

15. В открывшемся редакторе в первой строке ввести фразу «Сканер», во второй строке – «Модем», в третьей – «Принтер» и «щелкнуть» по кнопке <Ok>. Каждый вариант ответа будет размещен в панели компонента *RadioGroup* напротив соответствующей радиокнопки.

16. Скопировать на вторую страницу диалога готовую панель с первой страницы. Для этого на первой странице выделить панель «щелчком» мыши и выполнить команду *Copy* из меню *Edit*. В *ActivePage* выбрать вкладку «Вопрос 2», активизировать мышью внутреннюю панель *TabSheet2* («щелчок» мышью в центре страницы) и выполнить команду *Paste* из меню *Edit*. Изменить текст вопроса и варианты ответов.

17. Аналогично оформить остальные вопросы и ответы.

18. Сохранить созданный модуль формы «Тестирование» под именем *testir*.

19. Открыть окно редактора кода главной программы с процедурой обработки «щелчка» мыши на команде меню «Тестирование».

20. Подключить директиву *#include "testir.h"*.

21. Вернуться в место мигания курсора, ввести команду

PagesDlg1->ShowModal();

Эта команда при выполнении программы будет выводить на экран окно «Тестирование».

3.4. Разработка формы «Результат» и завершение проекта

Для решения данной задачи необходимо создать форму «Результат» (рис. 3.6), используя шаблон формы *Standard Dialog* из раздела *Dialogs*. На данной форме выводится результат в компонент *Label*. Однако перед открытием данной формы должны быть произведены вычисления, т. е. подсчет результатов.

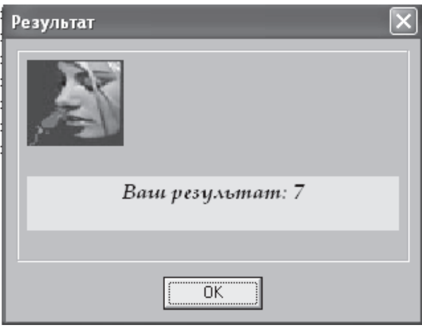


Рис. 3.6. Окно «Результат»

В листинге 3.1 приводится программный код основного модуля, который обрабатывает форму, на которой находится меню программы.

Листинг 3.1. Программа «Тестирование»

```
#include <vcl.h>
#pragma hdrstop

#include "mainform.h"
#include "about.h"
#include "instruct.h"
#include "testir.h"     // интерфейсный файл формы тестирования
#include "result.h"     // интерфейсный файл формы результата
```

```
//-----------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
//-----------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
: TForm(Owner)
{
}
//-----------------------------------------------------------

void __fastcall TForm1::Button1Click(TObject *Sender)
{
      Form1->Close();
}
//-----------------------------------------------------------
void __fastcall TForm1::N7Click(TObject *Sender)
{
      AboutBox->ShowModal();
}
//-----------------------------------------------------------
void __fastcall TForm1::N6Click(TObject *Sender)
{
      PagesDlg->ShowModal();
}
//-----------------------------------------------------------
void __fastcall TForm1::N2Click(TObject *Sender)
{
      PagesDlg1->ShowModal();
}
//-----------------------------------------------------------
void __fastcall TForm1::N3Click(TObject *Sender)
{
      int ball=0;
      if (PagesDlg1->RadioGroup1->ItemIndex==1) ball++;
      if (PagesDlg1->RadioGroup2->ItemIndex==0) ball++;
      if (PagesDlg1->RadioGroup3->ItemIndex==1) ball++;
      if (PagesDlg1->RadioGroup4->ItemIndex==0) ball++;
      if (PagesDlg1->RadioGroup5->ItemIndex==2) ball++;
      if (PagesDlg1->RadioGroup6->ItemIndex==1) ball++;
      if (PagesDlg1->RadioGroup7->ItemIndex==0) ball++;
      OKBottomDlg->Label1->Caption="Ваш результат: " + IntToStr(ball);
```

```
    OKBottomDlg->ShowModal();
}
//---------------------------------------------------------
void __fastcall TForm1::N4Click(TObject *Sender)
{
    Form1->Close();
}
```

В результате тестирования за каждый правильный ответ начисляется 1 балл, и общая сумма баллов (максимум 7 баллов) должна выводится на форму «Результат». В программе результаты тестирования суммируются в переменной *ball* при условии, если пользователь выбрал правильный ответ («правильную» кнопку). В программе верный ответ обрабатывается через порядковый номер (индекс) кнопки в группе. Так, например, в первом вопросе правильным является третий ответ, значит, индекс радиокнопки – 2.

В программном коде строка

```
if (PagesDlg1->RadioGroup1->ItemIndex==1) ball++;
```

позволяет увеличить значение *ball* в случае правильного ответа. В ином случае эта команда не сработает.

В программной строке

```
OKBottomDlg->Label1->Caption="Ваш результат: " + IntToStr(ball);
```

свойству *Caption* компонента *Label1*, установленного в форме *«Результат»*, присваивается значение 'Ваш результат:' и число набранных баллов, преобразованных в строковое значение. Функция *IntToStr(Ball)* переводит целое число, находящееся в переменной *ball*, в формат строки. Команда

```
OKBottomDlg->ShowModal();
```

демонстрирует форму «Результат».

Вопросы для самоконтроля

1. Какой компонент используется для создания на форме командного меню?
2. Каким образом осуществляется подключение модуля к главной программе?
3. С какой целью используется компонент *Memo* и как осуществляется в него ввод текста?
4. С какой целью используется компонент *RadioGroup*? С помощью какого свойства задаются его элементы?
5. Какой метод позволяет вывести форму на экран; закрыть форму?

4. СИМВОЛЫ И СТРОКИ

4.1. Символы

В среде C++Builder присутствует символьный тип данных, характеризующийся ключевым словом *char*.

Значением переменной символьного типа *char* может быть любой отображаемый на экране символ: буква русского или латинского алфавитов, цифра или знак препинания. Символьный тип представляется одиночным символом, заключенным в апострофы.

Переменная символьного типа должна быть объявлена в разделе объявления переменных. Синтаксис объявления символьной переменной:

***char** идентификатор*;

где *идентификатор* – имя переменной символьного типа; *char* – ключевое слово обозначения символьного типа.

Например,

***char** symb1;*

***char** symb2 = 'r';*

где *symb1, symb2* – идентификаторы переменных символьного типа; первый пример – объявление переменной, а второй пример – инициализация переменной.

При записи текста программы вместо символа можно указать его код. Так, например, часто используемый при записи сообщений символ "новая строка" записывается как *char*(13).

4.2. Массивы символов

В среде C++Builder присутствует символьный тип данных, но отсутствует специальный строковый тип.

Строка рассматривается как массив символьных значений *char*, оканчивающийся нулевым символом '\0'. Обращение к строке осуществляется через указатель на первый символ строки, то есть строка является в данной среде указателем.

Строка может быть объявлена либо как массив символов, либо как переменная типа *char**. Каждое их двух приведенных ниже эквивалентных объявлений

***char** w[] = "строка";*

***char** *ww = "строка";*

присваивает строковой переменной начальное значение «строка».

Доступ к отдельным символам переменной осуществляется по индексам. Так, например, обращение w[3] к слову «компьютер» будет относиться к символу «п», так как индексы в строках начинаются с нуля.

Для обработки строк имеется ряд библиотечных функций (работающих именно с массивами символов типа char), некоторые из которых используются наиболее часто:

1) функция *strcat (строка1, строка2)* – конкатенация (склеивание) строковых значений;

2) функция *strcpy (строка1, строка2)* – копирование одной строки в другую (возвращает строковое значение *строка1*);

3) функция *strlen (строка)* – определение длины строки (возвращает целое число);

4) функция *strstr (строка1, строка2)* – поиск в строке заданной подстроки (возвращает NULL или число – позицию, с которой *строка2* в первый раз входит в *строку1*);

5) функция *strcmp (строка1, строка2)* – сравнение двух строк (возвращает значение <0 при *строка1<строка2*; 0 при *строка1=строка2*; >0 при *строка1>строка2*;

6) функция *AnsiStrLower (строка)* – приведение символов строки к нижнему регистру;

7) функция *AnsiStrUpper (строка)* – приведение символов строки к верхнему регистру.

Пример. Разработать программу замены в предложении заданной комбинации символов на другую комбинацию.

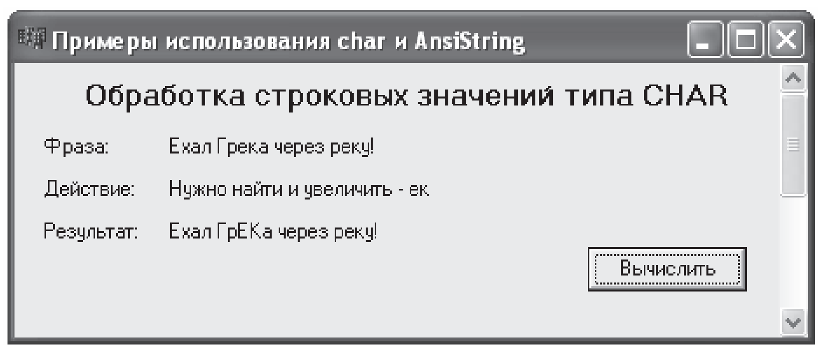


Рис. 4.1. Окно демонстрации строковых значений типа char*

Для данной задачи создается форма с несколькими метками, которые используются для комментариев. Сама фраза, действие и результат выводится в текстовые метки после нажатия на кнопку <*Вычислить*>. Программный код оформлен в методе, описанном для события нажатия на кнопку (листинг 4.1).

*Листинг 4.1. Обработка строк типа char**

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
    // объявление переменной исходной фразы
  char *word1="Ехал Грека через реку!";
    // объявление переменной поиска
  char *word2="ек";
    // объявление переменной замены
  char S1[40]="ЕК";
    // объявление промежуточных переменных
  char *St, S[40];
    // задание первой метке текста исходной фразы
  Label1->Caption=word1;
    // конкатенация строк для второй метки
  Label2->Caption =
  strcat("Нужно найти и увеличить – ",word2);
  St = strstr(word1, word2);                                    // 1
    // если значение St – true
  if (St)                                                       // 2
  {
    *St=0;                                                      // 3
    St +=strlen(word2);                                         // 4
    Label3->Caption = strcat(strcat(strcpy(S,word1),S1),St);    // 5
  }
  else Label3->Caption= "Текст не найден!";
}
```

Необходимо заметить, что команда ***St = strstr(word1, word2)***; записана в программе только для того, чтобы выяснить наличие ненулевого значения; в противном случае система выведет сообщение о том, что текст не найден. Далее в условной конструкции эта переменная обнуляется для дальнейшей работы с фрагментами строк.

Рассмотрим, каким образом осуществляется замена одного текста на другой (по комментариям 1-5 программного кода) с помощью функций, обрабатывающих тип *char**.

В исходном виде массив *word1* представлен как:

‘Е’, ‘х’, ‘а’, ‘л’, ‘’, ‘Г’, ‘р’, **‘е’**, ‘к’, ‘а’, ‘’, ‘ч’, ‘е’, ‘р’, ‘е’, ‘з’, ‘’, р‘’, ‘е’, ‘к’, ‘у’, ‘!’, ‘\0’

После выполнения функции (1) указатель St будет указывать на восьмой символ – букву ‘е’. Командой (3) с использованием указателя (*) на это место (8-й символ) заносится символ конца строки ‘\0’.

После этого исходная строка выглядит следующим образом:

‘Е’, ‘х’, ‘а’, ‘л’, ‘’, ‘Г’, ‘р’, **‘\0’**, ‘к’, ‘а’, ‘’, ‘ч’, ‘е’, ‘р’, ‘е’, ‘з’, ‘’, р‘’, ‘е’, ‘к’, ‘у’, ‘!’, ‘\0’

Функция (4) увеличивает St на длину строки «ек» (на два символа) и начинает указывать на символ ‘а’ (последняя буква слова «Грека»).

Подробно рассмотрим команду (5). Функцией *strcpy(S,word1)* в строку S заносится исходная фраза, но до первого символа конца ‘\0’, то есть

‘Е’, ‘х’, ‘а’, ‘л’, ‘’, ‘Г’, ‘р’, **‘\0’**.

Функция *strcat(strcpy(S,word1),S1)* объединяет описанную выше фразу с символами «ЕК», после чего получается фраза:

‘Е’, ‘х’, ‘а’, ‘л’, ‘’, ‘Г’, ‘р’, **‘Е’, ‘К’**, **‘\0’**.

Так как в строке St указатель стоит на букве ‘а’, то и в состав этой переменной входит уже оставшаяся часть исходной фразы:

‘а’, ‘’, ‘ч’, ‘е’, ‘р’, ‘е’, ‘з’, ‘’, р‘’, ‘е’, ‘к’, ‘у’, ‘!’, ‘\0’

После функции *strcat(strcat(strcpy(S,word1),S1),St)* происходит объединение двух последних последовательностей символов в фразу:

‘Е’, ‘х’, ‘а’, ‘л’, ‘’, ‘Г’, ‘р’, **‘Е’, ‘К’**, ‘а’, ‘’, ‘ч’, ‘е’, ‘р’, ‘е’, ‘з’, ‘’, р‘’, ‘е’, ‘к’, ‘у’, ‘!’, ‘\0’

4.3. Тип срок AnsiString

Тип строк *AnsiString* реализован в C++Builder как класс, объявленный в файле *vcl/dstring.h*. Эти строки с нулевым символом в конце.

Для класса *AnsiString* определены такие операции как операции отношения:!=, <, >, >=, <=, = =. Сравнение происходит с учетом регистра (регистр в языке C++ и других версиях имеет основополагающее значение). Если для строк типа char* существует специальная функция конкатенации, то для типа *AnsiString* существует простой способ соединения строк – склеивание с помощью символа +.

Необходимо обратить внимание на то, что тип *AnsiString* – это класс, значит, все функции данного класса – это методы, которые оформляются через точку:

Строка.метод (аргумент1, аргумент2, ...)

Методы класса AnsiString описаны ниже:

1) метод *AnsiPos (строка)* – возвращает индекс символа первого вхождения *Строки* в строку *Str*.

Пример:

AnsiString Str = “program”;
AnsiString s=”gr”;
short n;
n = Str.AnsiPos(s);

// Результат n=4

2) метод *Delete (позиция, число_символов)* – удаляет из текущей строки *Str* указанное *Число_символов* с заданной *Позиции*.

Пример:

```
AnsiString Str = “program”;
AnsiString q;
q = Str.Delete(2,4);
// Результат q=””
```

3) метод *FloatToStrF (число, формат, длина, количество_после_запятой)* – преобразует значение *Числа*, используя указанный *Формат*, в число определенной *Длины* и *Количества* знаков после запятой.

Пример:

```
AnsiString s;
float a=456.292838383;
s = FloatToStrF (a, ffFixed, 7, 3);
// Результат s = 456.293
```

В качестве второго параметра данная функция использует следующие форматы:

ffFixed	– формат с фиксированной точкой вида “-ddd.ddd...”.
ffGeneral	– основной числовой формат (в том виде, как определено значение переменной).
ffExponent	– научный формат вида “-d.ddd...E+dddd”.
ffNumber	– числовой формат вида “-d,ddd, ddd.ddd...”.
ffCurrency	– денежный формат, при котором число преобразуется в строку с указанием в ней денежной суммы.

4) метод *Insert (строка, позиция)* – вставляет *Строку* в текущую строку *Str*, начиная с указанной *Позиции*.

Пример:

```
AnsiString oldStr = “котер”;
AnsiString newStr = “мпью”;
AnsiString res;
res = oldStr.Insert (newStr, 3);
// Результат res=”компьютер”
```

5) метод *IsEmpty ()* – возвращает значение *true*, если строка пустая.

Пример:

```
AnsiString s = “”;
AnsiString ss = “rows”;
bool a, aa;
a = s.IsEmpty();
aa = ss.IsEmpty();
if (a == true)
```

```
 { Label5->Caption = "true"; }
else
 { Label5->Caption = "false"; }
if (aa)
 { Label6->Caption = "true"; }
else
 { Label6->Caption = "false"; }
```

Необходимо заметить, что выражения условия (*a==true*) и (*a*) эквивалентны;

6) метод *Length ()* – возвращает целое число, характеризующее количество символов в строке;

7) метод *Pos (строка)* – возвращает позицию первого символа первого вхождения *Строки* в текущую строку *Str*. В отличие от метода *AnsiPos* не поддерживает многобайтные символы;

8) метод *SetLength (количество)* – сокращает текущую строку до указанного *Количества* символов (в длину).

Пример:

```
AnsiString Str1="модернизация ";
AnsiString res;
res = Str1.SetLength(6);
// Результат res="модерн"
```

9) метод *StringOfChar (символ, количество)* – возвращает строку, в которой *Символ* повторяется указанное *Количество* раз.

Например,

```
AnsiString s=AnsiString:: StringOfChar ('a', 10);
```

Данная команда позволяет инициализировать переменную s типа *AnsiString* длиною в 10 символов «а». При этом не вводится новая переменная, к которой можно применить метод *StringOfChar*, а используется чисто объектно-ориентированный подход: сначала записывается класс, затем через два «:» записывается метод данного класса. Необходимо также учесть (как и в С++), что переменная объявлена именно через метод-конструктор, который совпадает по названию с именем класса (причем всегда).

Для перевода из строковых значений в числовые также используются методы *ToDouble ()* и *ToInt ()*, которые переводят строку в вещественное и целое числа соответственно.

Методы *Trim (), TrimLeft ()* и *TrimRight ()* позволяют удалять лишние пробелы в строках: все пробелы, слева от строки и справа от строки соответственно.

Для приведения букв к верхнему или нижнему регистрам используются методы *UpperCase ()* и *LowerCase ()* соответственно.

Рассмотрим пример, в котором производятся: 1) вычисление длины предложения (в словах); 2) вставка слова в указанное место предложения (рис. 4.2).

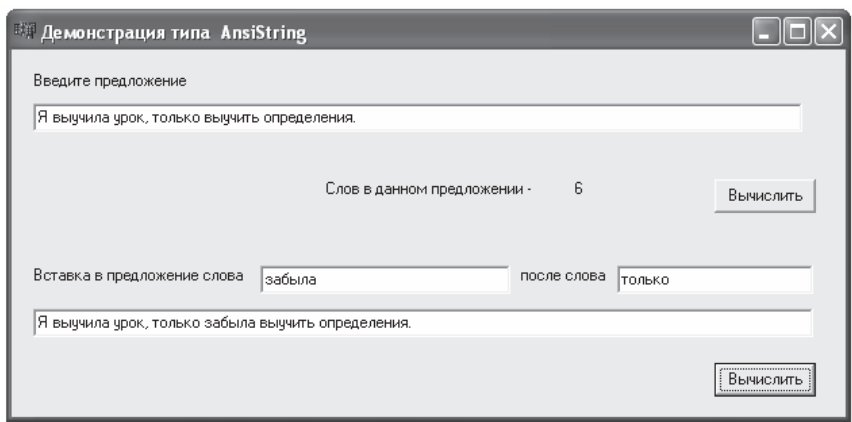


Рис. 4.2. Окно программы «Демонстрация типа AnsiString»

В листинге 4.2 приводится фрагмент программного кода, в котором описываются обработки событий кнопок <Вычислить>.

Листинг 4.2. Демонстрация типа AnsiString

```
      // объявление глобальных переменных
 AnsiString fraza, fraza1, fraza2, fraza3, fr, oldw, neww;
 int pr = 0;
 int pos;
// ------------------------------------------------

void __fastcall TForm1::Button1Click(TObject *Sender)
{
      // считывание фразы из текстового поля и удаление всех
      // пробелов до и после фразы (за исключением внутренних
      // пробелов)
 fraza = Edit1->Text;
 fraza1 = fraza.TrimRight();
 fraza2 = fraza1.TrimLeft();
      // если строка – пустая, то выводится соответствующее
      // сообщение; в противном случае – «вырезается» каждый
      // символ из фразы и по количеству пробелов вычисляется
      // количество слов в предложении
 if (fraza2!="")
```

```
{
      for (int i=0; i<=fraza2.Length(); i++)
      {
                  if (fraza2.SubString(i,1)==" ")
                  {
                              pr++;
                  }
      }
      Label3->Caption=pr+1;
}
else
{
      ShowMessage("Строка пустая!");
      Label3->Caption=0;
}
}
//-----------------------------------------------------------------

void __fastcall TForm1::Button2Click(TObject *Sender)
{
      // считывание фразы с одновременным удалением всех
      // пробелов до и после фразы
fraza3 = ((Edit1->Text).TrimRight()).TrimLeft();
      // считывание пропущенного слова (нового) и слова, после
      // которого пропущенное слово располагается (старого)
oldw=Edit3->Text;
neww=Edit2->Text;
      // поиск позиции, с которой начинается старое слово
pos=fraza3.AnsiPos(oldw);
      // вставка дополнительного пробела после старого слова
fr=fraza3.Insert(" ", pos+oldw.Length()+1);
      // вставка нового слова
Edit4->Text=fr.Insert(neww, pos+oldw.Length()+1);
}
```

Необходимо заметить, что приведенные функции являются основными и в сочетании друг с другом позволяют программисту обрабатывать строковые значения, полученные из различных компонентов.

4.4. Примеры использования строковых значений

Рассмотрим несколько примеров на использование строк и символов.

Пример 1. Продолжим решение задачи, демонстрация которой

представлена на рис. 4.1, добавив на форму компоненты для обработки строковых значений, описанных через класс *AnsiString* (рис. 4.3).

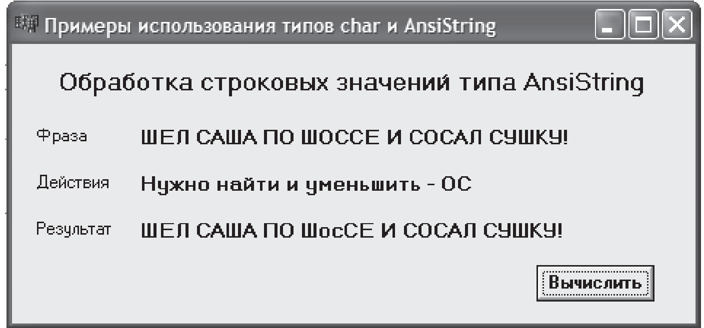


Рис. 4.3. Окно задачи «Примеры использования типов char и AnsiString»

В листинге 4.3 представлен программный код события нажатия на вторую кнопку окна «*Примеры использования char и AnsiString*».

Листинг 4.3. Обработка строк типа AnsiString

```
void __fastcall TForm1::Button2Click(TObject *Sender)
{
      // объявление и инициализация переменных
 AnsiString word_01, word_02, S_01;
 word_01="ШЕЛ САША ПО ШОССЕ И СОСАЛ СУШКУ!";
 word_02="ОС";
 S_01="ос";
 Label4->Caption=word_01;
 Label5->Caption="Нужно найти и уменьшить – "+word_02;
      // определение переменной i, хранящей позицию первого
      // вхождения строки word_02 в строку word_01
 int i=word_01.Pos(word_02);
      // если значение i – true
 if (i)
 {
      // конкатенация различных частей фразы
 Label6->Caption=word_01.SubString(1,i-1)+ S_01 +
 word_01.SubString(i+word_02.Length(),255);
 }
 else Label6->Caption="Текст не найден!";
}
```

Пример 2. Разработать программу, выполняющую следующие операции: 1) конкатенация; 2) перевод к нижнему и верхнему регистрам; 3) нахождение длины строки (рис. 4.4).

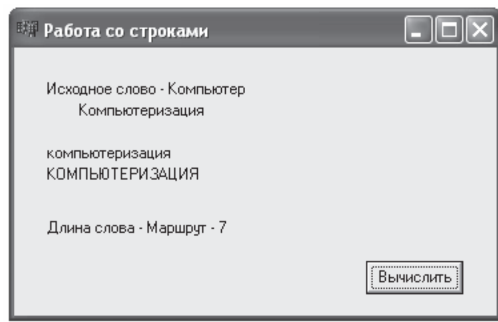


Рис. 4.4. Окно программы «Работа со строками»

В данном примере используется только одна переменная типа *AnsiString*, которая связана со строкой типа *char** операцией конкатенации +. В остальных случаях рассматривается работа с типом *char**.

Листинг 4.4. Работа со строками

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
    // Важно! Типы AnsiString и *char нельзя приравнивать!
AnsiString title;
title = "Исходное слово – ";
char txt[]="Маршрут";
char *txt1="Компьютер";
    // пример 1 – конкатенация
Label1->Caption= title + txt1;
Label2->Caption=strcat(txt1,"изация");
    // пример 2 – при использовании переменной txt, объявленной
    // как массив символов, конкатенация не происходит –
    // генерируется ошибка
    // Label3->Caption= title + txt;
    // Label4->Caption=strcat(txt,"изация");
    // пример 3 – верхний и нижний регистры
Label3->Caption= AnsiStrLower(txt1);
Label4->Caption= AnsiStrUpper(txt1);
    // пример 4 – длина строки
Label5->Caption= "Длина слова Маршрут – " +
IntToStr(strlen(txt));
}
```

Пример 3. Вывести на форму таблицу кодировки букв русского алфавита.

Основную работу выполняет процедура обработки события *OnActivate*, которая формирует и выводит в поле метки (*Label1*) таблицу.

Событие *OnActivate* происходит при активизации формы приложения, и поэтому метод *TForm1.FormActivate* выполняется автоматически после появления формы на экране. Текст метода приведен в листинге 4.4, а диалоговое окно программы – на рис. 4.5.

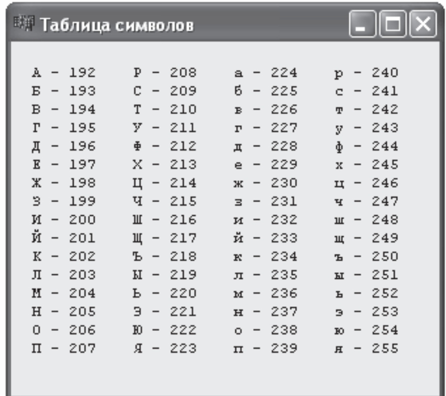


Рис. 4.5. Окно «Таблица символов»

Листинг 4.5. Таблица символов

```
void __fastcall TForm1::FormActivate(TObject *Sender)
{
      // объявление переменных
AnsiString st;
short i,j;
short dec;
      // инициализация переменных
st=””;
dec=192;
      // объявление
for (i=0; i<=15; i++)
 {
      dec=i+192;
      for (j=1; j<=4; j++)
      {
            st=st+char(dec)+” – “+IntToStr(dec)+” “;
            dec=dec+16;
      }
      st=st+char(13);
```

```
 }
Label1->Caption=st;
}
```

Форма рассматриваемого приложения содержит только один компонент – поле метки (*Label1*). Для того чтобы колонки таблицы имели одинаковую ширину, свойству *Label1.Font.Name* следует присвоить имя шрифта, у которого все символы имеют одинаковую ширину, например, *Courier New Cyr*.

Вопросы для самоконтроля

1. Как осуществляется описание переменных строкового и символьного типов?
2. Как получить значение кода требуемого символа?
3. Какое отношение каждый символ имеет к таблице кодов? Какой символ характеризуется кодом #13?
4. В какой ситуации осуществляется вызов процедуры *FormActivate*?
5. Каков синтаксис и назначение функций *Length*, *Pos*, *Copy*, *Delete*?

5. ТИПОВЫЕ ДЕЙСТВИЯ С МАССИВАМИ ДАННЫХ

Массив – это структура данных, которую можно рассматривать как набор переменных одинакового типа, имеющих общее имя. Массивы удобно использовать для хранения однородной по своей природе информации, например, таблиц, коэффициентов уравнений, матриц.

5.1. Объявление массива

С массивами осуществляются следующие действия:

1) Объявление, заключающееся в указании имени массива, типа значений, а также при необходимости размера (длины) массива:

тип имя_массива [];

тип имя_массива [размер];

Например, *massiv[3].*

2) Инициализация, позволяющяя задать каждому элементу массива конкретное значение описанного типа. При этом необходимо помнить, что нумерация индексов в массивах начинается с нуля. Например,

```
massiv[0]=34;
massiv[1]=78;
massiv[2]=65;
```

Отсюда видно, что при индексе, равном 3, будет переполнение массива, а это допускать нельзя. Таким образом, при работе с индексами массива необходимо использовать число, на 1 меньшее, чем объявлено в размере массива.

Также можно *определить* массив, объединив объявление и инициализацию:

```
AnsiString Str[]={"один", "два", "три"};
```

3) Обращение к элементам массива через имя массива и индекс. Для заполнения или считывания элементов массива наиболее часто используется цикл с известным количеством повторений.

```
for (i=0; i<N; i++)
{
 massiv[i]=i*5;
}
```

В случае, если необходимо задать массив как константу, то достаточно перед объявлением указать ключевое слово *const*.

Существуют как *одномерные* массивы (описанные выше), так и многомерные (см. в следующем пункте).

5.2. Вывод элементов массива на экран

Рассмотрим программу, выводящую значения всех элементов массива в диалоговое окно. Предварительно необходимо организовать одномерный массив *team*, элементами которого будут названия команд – участниц футбольного чемпионата.

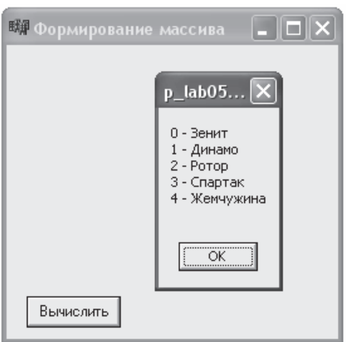


Рис. 5.1. Окно «Формирование массива»

Обработка массива выполняется через нажатие кнопки <*Выполнить*>, после чего генерируется событие *onClick* (листинг 5.1).

Листинг 5.1. Программный код формирования массива

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
      // объявление константы NT
 const short NT=5;
      // объявление массива team размером 5
 AnsiString team[NT];
      // объявление выходной строковой переменной
 AnsiString st;
 short i;
      // задание значений массива
 team[0]="Зенит";
 team[1]="Динамо";
 team[2]="Ротор";
 team[3]="Спартак";
 team[4]="Жемчужина";
      // склеивание элементов массива в строку для вывода в диалоговом
      // окне; предварительно индекс преобразуется в строку
 for (i=0; i<NT; i++)
 {
```

```
 st = st + IntToStr(i) + " – " + team[i] + char(13);
 }
 ShowMessage(st);
}
```

Из программного кода видно, что для вывода информации в диалоговое окно используется функция *ShowMessage(*сообщение*)*, в качестве параметра которой используется строковое значение.

5.3. Ввод элементов массива

Под *вводом* элементов массива понимается получение от пользователя во время работы программы значений элементов массива. Рассмотрим два варианта организации ввода массива:

1) с использованием поля редактирования (компонента *Edit*);
2) с использованием поля *Memo*.

5.3.1. Использование компонента EDIT

Диалоговое окно программы, позволяющее вводить элементы массива с клавиатуры, может выглядеть как на рис. 5.2.

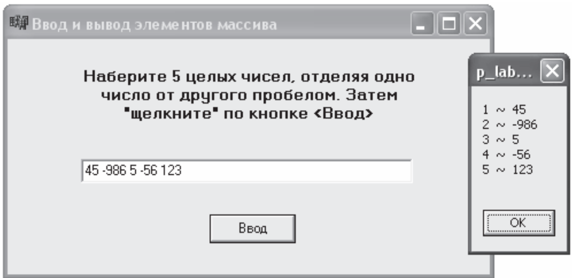


Рис. 5.2. Диалоговое окно программы «Ввод и вывод элементов массива»

После «щелчка» на кнопке <*Ввод*> программа выполнит следующие действия:

1) выделит из введенной строки первую подстроку с числом (например, «45»);
2) преобразует ее в число;
3) присвоит полученное значение первому элементу массива;
4) выделит каждое последующее значение строки (расположенное между пробелами) и осуществит с ним действия п. 2 и п. 3.

Для демонстрации работы программы необходимо создать диалоговое окно, разместив на нем три компонента: метку (*Label1*), поле ввода (*Edit1*) и командную кнопку (*Button1*).

Процедура обработки события *OnClick* для командной кнопки может выглядеть следующим образом (листинг 5.2):

Листинг 5.2. Формирование элементов массива из строки

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
const short Size=5;
short i;
AnsiString a[Size];
AnsiString st1="";

st=Edit1->Text;
for (i=0; i<5; i++)
{
a[i]=GetSub(st);
st=st2;
}

for (i=0; i<Size; i++)
{
st1=st1+IntToStr(i+1)+" ~ "+a[i]+char(13);
}
ShowMessage(st1);
}
```

В данном фрагменте *Size* – именованная константа, определяющая размер массива; *a* – массив; *Edit1* – имя поля ввода/редактирования; *GetSub* – имя пользовательской функции (ее описание приведено ниже), которая выделяет из строки подстроку с указанным номером (в данном случае подстрока – это последовательность символов между двумя пробелами).

Рассмотрим функцию *GetSub*, реализующую ввод элементов массива.

Листинг 5.3. Функция возвращения подстроки с указанным номером

```
AnsiString GetSub(AnsiString st)
{
short p,i;
AnsiString res;
p=st.TrimRight().TrimLeft().AnsiPos(" ");
if (p!=0)
{
res=st.SubString(1,p-1);
```

```
}
else
{
res=st;
}
st2=st.SubString(p+1,st.Length()-p);
return res;
}
```

Для того чтобы из полученной в качестве аргумента строки выделить *n*-ю подстроку (элемент массива), функция *GetSub* сначала удаляет в цикле *for* предшествующие ей *n-1* подстроки, затем находит пробел, который отмечает конец нужной подстроки, выделяет ее и возвращает в качестве значения функции через неявную локальную переменную *result*.

Описание (текст) этой функции следует поместить в описательную часть модуля до всех остальных методов.

5.3.2. Использование компонента Memo

Для ввода символьного массива, занимающего несколько строк, целесообразнее использовать компонент *Memo*, который позволяет вводить несколько строк текста (в отличие от *Edit*).

В табл. 7 приведены свойства компонента *Memo*.

Таблица 7

Свойства компонента Memo

Свойство	Определяет
Name	Идентификатор компонента. В программе используется для доступа к свойствам компонента
Lines	Текст, находящийся в строках поля *Memo*
Height	Высоту поля
Width	Ширину поля
Text	Текст, находящийся в поле ввода-редактирования
Font	Шрифт, используемый для отображения вводимого текста

При использовании компонента *Memo* ввод каждого элемента массива должен заканчиваться нажатием клавиши <*Enter*>. Доступ к находящейся в поле *Memo* строке текста можно получить при помощи свойства *Lines*, указав в квадратных скобках номер нужной строки (строки нумеруются с 0).

На рис. 5.3 представлен интерфейс диалогового окна для ввода элементов массива с клавиатуры. Обработка данных будет происходить после нажатия клавиши <*Ввод*>.

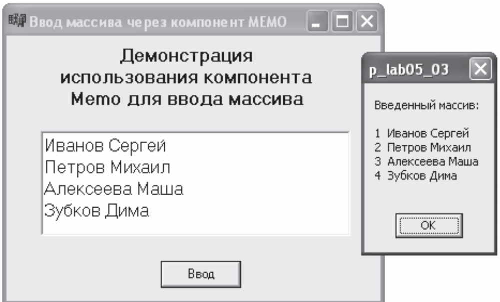


Рис. 5.3. Диалоговое окно приложения «Ввод массива»

Приведенный ниже листинг программного кода обработки события нажатия кнопки <*Ввод*> демонстрирует использование компонента *Memo* для ввода строкового массива.

Листинг 5.4. Обработка элементов списка

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
    const short SIZE=4;
    AnsiString a[SIZE];
    short i, n;
    AnsiString st;
    n=Form1->Memo1->Lines->Count;
    if (n==0)
    {
            ShowMessage("Исходные данные не введены!");
    }
    if (n>SIZE)
    {
            ShowMessage ("Количество введенных строк \n
                                        превышает размер массива.");
            n=SIZE;
    }
    for (i=0; i<n; i++)
    {
            a[i]= Form1->Memo1->Lines->Strings[i];
    }
    if (n>0)
```

```
    {
            st=”Введенный массив: \n\n”;
            for (i=0; i<n; i++)
            {
                    st=st+IntToStr(i+1)+ “ “ + a[i]+char(13);
            }
            ShowMessage (st);
    }
}
```

В процедуре *TForm1.Button1Click* сначала осуществляется проверка на наличие текста в поле *Memo1.* Если в компоненте присутствует текст (в этом случае значение свойства *Count* больше нуля), то процедура сравнивает количество введенных строк и размер массива. Если это количество превышает размер массива, то программа изменяет значение *n*, тем самым подготавливая ввод только первых *SIZE* строк.

После «щелчка» на кнопке <*Ввод*> откроется диалоговое окно *Pr_memo* (рис. 5.3), в которое будут выведены значения элементов массива, полученные из *Memo*-поля.

Вопросы для самоконтроля

1. Каким образом осуществляется объявление массива в программе?
2. Как осуществляется доступ к элементу массива?
3. Опишите принцип организации ввода элементов массива с помощью компонента *Edit.*
4. Какой раздел содержит текст функций и методов, определенных программистом?
5. Что определяют свойства *Lines* и *Text* компонента *Memo*?
6. С помощью какого метода можно определить количество строк в компоненте *Memo*?
7. Каково содержание раздела *interface* при наличии в модуле функций и процедур, определенных программистом?

6. МНОГОМЕРНЫЕ МАССИВЫ

Исходные данные для решения многих задач удобно представляются в табличной форме. В программе для хранения и обработки таких табличных данных используют многомерные массивы. Если при описании массива задано два индекса, массив называется *двумерным*, если *N* индексов – *N-мерным*.

Пример описания двумерного массива целых чисел, состоящего из 3-х колонок и 4-х строк:

int massiv2 [3][4];

Чтобы получить доступ к отдельному элементу многомерного массива, нужно указать значение каждого индекса, например: ***massiv[2][3].***

В качестве примера рассмотрим программу, выполняющую обработку результатов спортивных соревнований олимпиады. Исходные данные для обработки представлены в табл. 8.

Таблица 8

Таблица спортивных соревнований олимпиады

Страна	Количество медалей		
	золотые	серебряные	бронзовые
Австрия	3	5	2
Германия	7	7	8
Италия	2	6	2
Канада	6	5	4
Нидерланды	5	4	2
Норвегия	6	3	5
Россия	10	9	5
США	9	6	4
Финляндия	2	4	6
Швейцария	2	2	3
Япония	5	1	4

Программа вычисляет общее количество медалей, завоеванных представителями каждой страны, и соответствующее количество очков (баллов), которое вычисляется по следующему правилу: за каждую золотую медаль команда получает 7 очков, за серебряную – 6, за бронзовую – 5 очков. Затем программа должна выполнить сортировку таблицы по убыванию количества набранных очков.

Вид диалогового окна программы представлен на рис. 6.1.

Итоги олимпиады

Программа вычислит общее количество медалей, завоеванное спортсменами каждой страны, и количество набранных баллов. Затем список команд будет переупорядочен в соответствие с количеством набранных баллов

Страна	Золотых	Серебряных	Бронзовых	Всего	Баллов
Россия	10	9	5	24	149
Германия	7	7	8	22	131
США	9	6	4	19	119
Канада	6	5	4	15	92
Норвегия	6	3	5	14	85
Нидерланды	5	4	2	11	69
Финляндия	2	4	6	12	68
Австрия	3	5	2	10	61
Япония	5	1	4	10	61
Италия	2	6	2	10	60
Швейцария	2	2	3	7	41

ИТОГИ

Рис. 6.1. Диалоговое окно программы «Итоги олимпиады»

В форме приложения появился новый компонент – *StringGrid* (строковая таблица), значок которого находится на вкладке *Additional* палитры компонентов.

В следующей таблице приведены значения свойств компонента *StringGrid*, используемого программой обработки результатов олимпиады для ввода исходных данных и вывода результатов (табл. 9).

Таблица 9

Описание свойств компонента StringGrid

Обозначение	Свойство	Значение
Name	Идентификатор компонента	Tab1
ColCount	Число колонок таблицы	6
RowCount	Число строк таблицы	12
FixedCols	Число зафиксированных слева колонок таблицы. Зафиксированные колонки выделяются цветом и при горизонтальной прокрутке таблицы остаются на месте	0
FixedRows	Число зафиксированных сверху строк таблицы. Зафиксированные строки выделяются цветом и при вертикальной прокрутке таблицы остаются на месте	1

Продолжение табл. 9

Обозначение	Свойство	Значение
Options.goEditing	Признак допустимости редактирования содержимого ячеек таблицы	TRUE
DefaultColWidth	Ширина колонок таблицы	68
DefaultRowHeight	Высота строк таблицы	16
GridLineWidth	Ширина линий, ограничивающих ячейки таблицы	1

Ячейки первой зафиксированной строки таблицы используются в качестве заголовков ее колонок. Во время создания формы приложения установить значения элементов массива *Cells* нельзя, так как элементы массива доступны только во время работы программы. Поэтому значения элементов массива *Cells*, соответствующих первой строке и первой колонке таблицы, устанавливает процедура обработки события *OnActivate* (листинг 6.1), которое происходит во время активизации формы приложения.

Листинг 6.1. Формирование массива

```
void __fastcall TForm1::FormActivate(TObject *Sender)
{
   Tab1->Cells[0][0]=”Страна”;
   Tab1->Cells[1][0]=”Золотых”;
   Tab1->Cells[2][0]=”Серебряных”;
   Tab1->Cells[3][0]=”Бронзовых”;
   Tab1->Cells[4][0]=”Всего”;
   Tab1->Cells[5][0]=”Баллов”;
   Tab1->Cells[0][1]=”Австрия”;
   Tab1->Cells[0][2]=”Германия”;
   Tab1->Cells[0][3]=”Италия”;
   Tab1->Cells[0][4]=”Канада”;
   Tab1->Cells[0][5]=”Нидерланды”;
   Tab1->Cells[0][6]=”Норвегия”;
   Tab1->Cells[0][7]=”Россия”;
   Tab1->Cells[0][8]=”США”;
   Tab1->Cells[0][9]=”Финляндия”;
   Tab1->Cells[0][10]=”Швейцария”;
   Tab1->Cells[0][11]=”Япония”;
}
```

Программа обработки исходной таблицы, текст которой приведен ниже, запускается при щелчке на командной кнопке <ИТОГИ>.

Листинг 6.2. Подсчет итогов

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
      // объявление переменных номеров колонки и строки таблицы
int c, r;
int s; // всего медалей у команды
int p; // очков у команды
int m; // номер строки с максимальным количеством баллов
AnsiString buf[6]; // буфер для обмена строк
int i; // номер строки используется во время сортировки

      // подсчет общего количества баллов у каждой команды
      // r – строка, c – столбец
for (r=1; r<Tab1->RowCount; r++)
{
   s=0;
   for (c=1; c<=3; c++)
   {
      // если в ячейке нет никаких значений, то система ставит число 0
        if (Tab1->Cells[c][r]!="")
        s=s+StrToInt(Tab1->Cells[c][r]);
        else Tab1->Cells[c][r]="0";
        p = 7*StrToInt(Tab1->Cells[1][r])+6*StrToInt(Tab1->Cells[2][r])+
                                                5*StrToInt(Tab1->Cells[3][r]);
        Tab1->Cells[4][r]=IntToStr(s);
        Tab1->Cells[5][r]=IntToStr(p);
   }
}
      //сортировка массива
for (r=1; r<Tab1->RowCount-1; r++)
{
   m=r;
   for (i=r; i<=Tab1->RowCount-1; i++)
        if (StrToInt(Tab1->Cells[5][i]) > StrToInt(Tab1->Cells[5][m]))
        m=i;
   if (r!=m)
   for (c=0; c<=5; c++)
   {
        buf[c]=Tab1->Cells[c][r];
        Tab1->Cells[c][r]=Tab1->Cells[c][m];
        Tab1->Cells[c][m]=buf[c];
```

```
    }
  }
}
```

Сначала для каждой страны программа вычисляет общее количество медалей и соответствующее ему количество очков. Затем выполняет сортировку таблицы по убыванию количества набранных очков. Во время сортировки для обмена строк таблицы используется одномерный строковый массив *buf*, индекс которого, как и индекс строки таблицы меняется от 0 до 5. Такой прием позволяет наиболее просто выполнить копирование замещаемой строки в буфер и замещение строки содержимым буфера.

Вопросы для самоконтроля

1. Каким образом осуществляется объявление двумерного массива?
2. Как описывается доступ к элементам двумерного массива?
3. Для чего предназначен компонент *StringGrid*?
4. С помощью каких свойств устанавливается число строк и столбцов компонента *StringGrid*?
5. Как можно реализовать заполнение компонента *StringGrid* программным путем?

7. СОЗДАНИЕ ОДНОТАБЛИЧНОЙ БАЗЫ ДАННЫХ

7.1. Организация связи с базами данных в C++Builder

Borland C++Builder не относится к системам управления базами данных (СУБД), т. е. не является специализированной средой, работающей только с объектами баз данных (БД). Система, разработанная фирмой Borland, является универсальной и работает с различными объектами, включая и БД. Чтобы пользователь мог разрабатывать приложения баз данных, необходимо воспользоваться специальным механизмом доступа к объектам данного типа.

Borland Database Engine (BDE) – процессор баз данных фирмы Borland. BDE служит посредником между приложением в C++Builder и базами данных, которые могут быть разработаны в любой БД, с которой среда работает. Приложение C++Builder никогда не обращается непосредственно к базе данных, а только к механизму BDE, сообщая ему псевдоним базы данных и необходимые таблицы в ней.

BDE реализован в виде динамически присоединяемых библиотек *DLL*, которые снабжены API (*Application Program Interface* – интерфейс прикладных программ), названным *IDAPI (Integrated Database Application Program Interface)*. Это список процедур и функций для работы с базами данных, которым и пользуются приложения.

Механизм BDE по псевдониму находит подходящий для указанной базы данных *драйвер.* BDE поддерживает естественный доступ к таким базам данных, как Microsoft Access, FoxPro, Paradox и dBase. Если собственного драйвера нужной СУБД в BDE нет, то используется драйвер ODBC.

ODBC (Open Database Connectivity) – динамическая библиотека, аналогичная по функциям BDE, но разработанная фирмой Microsoft. Работа через ODBC осуществляется несколько медленнее, чем через собственные драйверы СУБД, включенные в BDE, но благодаря связи с ODBC масштабируемость C++Builder существенно увеличилась, и сейчас из C++Builder можно работать с любой значительной СУБД.

BDE поддерживает стандартизованный язык запросов SQL (*Structured Query Language*), позволяющий обмениваться данными с SQL-серверами, такими, как Sybase, Microsoft SQL, Oracle, Interbase. Эта возможность используется особенно широко при работе на платформе клиент/сервер.

Работа с БД начинается с программы *Database Desktop*, которая позволяет загрузить программу по созданию и редактированию таблиц. Обычно вызов *Database Desktop* включен в главное меню C++Builder в раздел *Tools*. Если это не сделано, то полезно включить его туда с помощью команды *Tools / Configure Tools*....

7.2. Типы данных в среде C++Builder 6.0

В среде C++Builder 6.0 используются следующие типы данных (табл. 10).

Таблица 10

Описание типов данных

№ п/п	Тип данных	Обозначение	Назначение
1.	***A**lpha*	Строковый (текстовый)	Содержит любые печатаемые ASCII-символы; размер строки от 1 до 255 символов
2.	***N**umber*	Числовой	Числа в диапазоне от -10^{307} до 10^{308} с 15 значащими разрядами.
3.	***$** (Money)*	Денежный	Числовые значения, сопровождаемые денежным символом.
4.	***S**hort*	Короткий целый	Числа в диапазоне от -32 767 до 32 767
5.	*Long **I**nteger*	Длинный целый	Числа в диапазоне от -2 147 483 648 до 2 147 483 647
7.	***D**ate*	Датовый	Дата, представленная в следующем формате: 00.00.0000 (например, 13.09.2004)
8.	***T**ime*	Тип «Время»	Время, представленное в следующем формате: 00:00:00 (например, 12:10:34 – часы: минуты: секунды)
9.	***@** (Timestamp)*		Дата и время, одновременно представленные в следующем формате: 00:00:00, 00.00.0000 (например, 12:09:23, ˽ 13.09.2004). Между временем и датой задается запятая и пробел
10.	***M**emo*	Примечание	Поле, хранящее текст неограниченной длины. Сам текст примечания хранится в отдельном файле с расширением.**mb** (имя файла такое же, как и имя таблицы). Принимаемые значения: от 1 до 240 (число первых символов, хранящихся в таблице)

Продолжение табл. 10

№ п/п	Тип данных	Обозначение	Назначение
12.	***G**raphic*	Графический	Изображения из файлов в форматах **.bmp**, **.pcx**, **.tif**, **.gif** или **.eps**. Database Desktop преобразует их в формат **.BMP**
13.	***O**LE*	OLE-объект	Данные типа OLE-изображения, звуки, документы. Database Desktop не поддерживает поля этого типа
14.	***L**ogical*	Логический	Принимает только одно их двух значений: **false** или **true (**первый символ**)**
15.	**+** *(Autoincrement)*	Инкремент (автоматический)	Автоматически увеличивающееся на 1 длинное целое. Только для чтения. При удалении записей полей в оставшихся записях не изменяются

Необходимо заметить, что просмотр полей типа «Примечание», «Графический», OLE-объект возможен только в Paradox или в приложениях C++Builder.

7.3. Создание новой таблицы с помощью Database Desktop

Для создания новой таблицы (впервые) необходимо выполнить следующие действия:

1. Создать новый проект командой *File / New / Application*, если новый проект не открылся автоматически после загрузки системы программирования C++Builder.
2. Вызвать программу *Database Desktop* командой *Tools / Database Desktop*, после чего на экране появится окно для работы с таблицами (рис. 7.1).

В состав панели инструментов входят три пиктограммы: *Open Table* – открытие таблицы; *Open Query* – открытие запроса; *Open SQL File* – открытие SQL-файла.

3. Выполнить команду *File / New / Table*.
4. В появившемся окне выбрать таблицу типа *Paradox 7.0*. Нажать <Ok>.

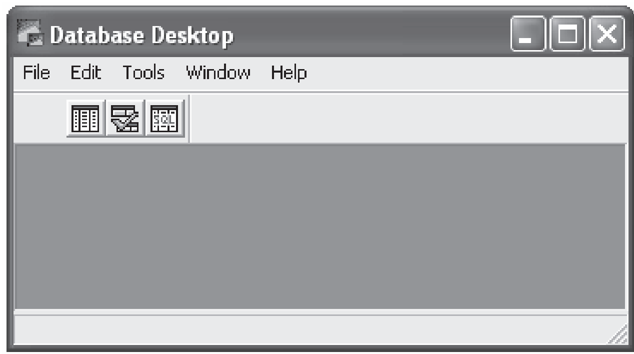


Рис. 7.1. Окно Database Desktop

5. В новом окне ввести названия поля (*Field Name*), типы полей (*Type*) и размеры полей (*Size*) в символах.

Через контекстное меню можно открыть список типов данных и выбрать необходимый тип или ввести букву или символ, закрепленный по умолчанию за тем или иным типом данных (рис. 7.2).

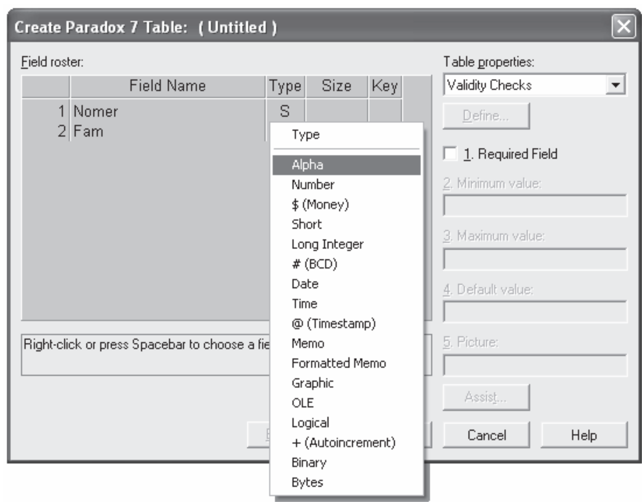


Рис. 7.2. Окно структуры таблицы с демонстрацией возможных типов данных

6. После ввода всех данных по полям установить некоторые свойства таблицы. Для этого необходимо активизировать пункт *Validity Checks* в комбинированном списке *Table Properties*. Данный пункт позволяет проверить правильность вводимых значений.

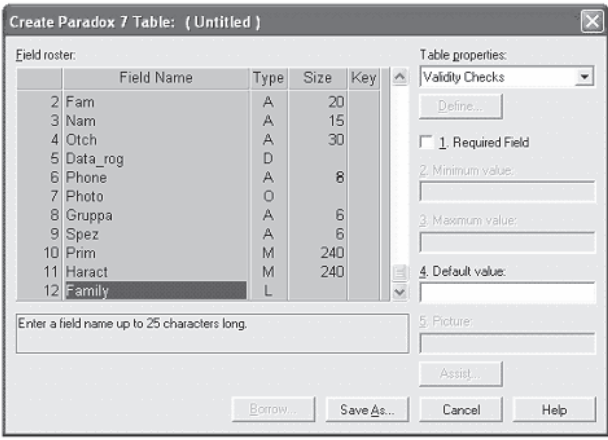


Рис. 7.3. Окно ввода структуры таблицы

7. Нажать кнопку <*Save As...*>, в открывшемся окне в поле *Alias* выбрать псевдоним *WORK* и сохранить таблицу под именем *student.db*.

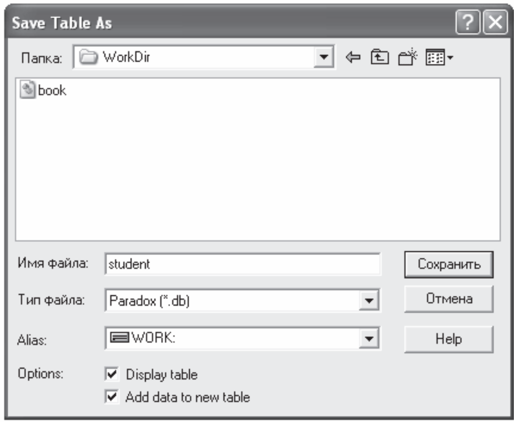


Рис. 7.4. Окно сохранения таблицы

Возможна ситуация, когда таблица не сохраняется. В данном случае необходимо проверить типы полей, их размеры (для строковых значений), а также индексы таблиц.

Для добавления записей в таблицу необходимо открыть таблицу командой *File/Open/Table*, а затем активизировать режим редактирования данных таблицы пиктограммой *Edit Data*.

7.4. Использование маски для строковых полей

В окне задания структуры таблицы присутствует кнопка <*Assist*>. Она открывает соответствующее окно и позволяет задать маску для поля, а также проверить правильность ее написания (рис. 7.5).

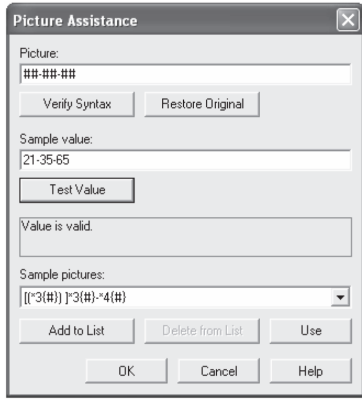


Рис. 7.5. Окно задания маски для тестового поля

Для задания маски строковых значений используются символы, характеризующие следующие символы: # – цифра; @ – любой символ; ? – любая буква.

Для задания маски ввода необходимо выполнить следующие действия:

1. В поле *Picture* задать маску.

2. Произвести «щелчок» на кнопке <*Verify Syntax*>, чтобы проверить правильность введенной маски. Если маска введена правильно, то в отдельной прямоугольной области появится фраза «*The Picture is correct*». Если необходимо удалить введенную маску, то можно воспользоваться кнопкой <*Restore Original*>.

3. После щелчка на поле *Sample value* ввести любой телефон. После ввода первых двух цифр система автоматически ставит дефис (по маске) и переводит курсор на следующую позицию. Необходимо заметить, что пользователь сразу не видит маски; только по мере ввода данных.

4. После ввода маски необходимо кнопкой <*Text Value*> проверить правильность введенного значения. Если в прямоугольной области появляется фраза «*Value is valid*», то введенное значение введено правильно.

5. Для сохранения маски необходимо нажать кнопку <*Add to List*>. В результате этого появится небольшое диалоговое окно «*Save Picture*». В нем необходимо ввести название маски (например, *Phone*), и зафиксировать операцию.

6. Нажатие кнопки <Ok> в текущем окне создания маски позволит установить маску для выбранного поля *Телефон*.

Для использования созданной маски в других таблицах необходимо воспользоваться кнопкой <*Use*>, которая сразу делает активной выбранную из списка маску.

Если необходимо ввести изменения в таблицу, например, добавить маску, то нужно выполнить следующие действия:

1. Выполнить команду *File/Open/Table* и загрузить для работы файл базы данных *student.db*.
2. Нажать пиктограмму <*Restructure*> или выполнить команду *Table / Restructure*.
3. Установить указатель мыши на поле *Phone*.
4. Нажать кнопку <*Assist*> для отображения окна *Picture Assistance*.
5. Установить необходимую маску, возможно, предварительно ее создав.
6. Сохранить введенные изменения, используя кнопку <*Save As...*>.

Необходимо заметить, что в окне структуры таблицы находится флажок *Required Field*, который в активном состоянии не разрешает текущему полю быть пустым. Так, например, для поля *Nomer* нельзя оставлять данный флажок пустым.

7.5. Создание и редактирование псевдонимов баз данных

Существует три альтернативных пути просмотра, создания и редактирования псевдонимов с помощью трех различных программ: *Database Desktop, BDE Administrator* и *Database Explorer*.

Существуют два псевдонима, которые автоматически создает BDE: рабочий (*working*) и частный (*private*). Рабочий каталог – ...*Program Files\Borland\Database Desktop\WorkDir*. Он имеет псевдоним WORK.

Для изменения рабочего каталога используется команда *File/Working Directory*, для изменения частного каталога – *File/Private Directory*.

Для просмотра информации о псевдонимах в *Database Desktop* необходимо выполнить следующие шаги:

1. Выполнить команду *Tools/Alias Manager*.
2. В левой части открывшегося окна в комбинированном списке *Database Alias* выбрать псевдоним WORK и просмотреть имеющуюся информацию об этой области (рис. 7.6): название псевдонима, тип драйвера и путь к файлу-драйверу.
3. Выбрать псевдоним *ibLOCAL* и просмотреть всю информацию об этой области (рис. 7.7): имя сервера (SERVER NAME), имя

пользователя (USER NAME), режим открытия БД (OPEN MODE), размер КЭШа схем (SCHEMA CACHE SIZE), драйвер языка (LANGDRIVER), режим обработки запросов SQL (SQLQRYMODE), доступ к QBE и SQL (SQLPASSTHRU), пароль (PASSWORD).

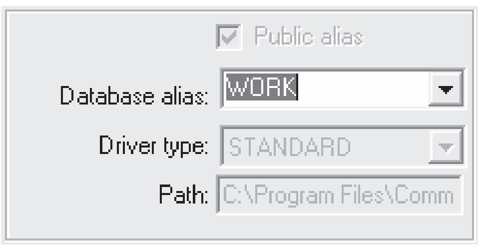


Рис. 7.6. Фрагмент окна

Рис. 7.7. Окно Alias Manager для псевдонима IBLocal

Для создания нового псевдонима используется кнопка <*New*>; для сохранения – кнопка <*Save As*>; для удаления области – кнопка <*Remove*>.

Наиболее удобным средством для работы с псевдонимами является *BDE Administrator.*

Для создания нового псевдонима необходимо выполнить следующие действия:

1. Свернуть окно *C++Builder 6.0* и закрыть программу *Database Desktop*.
2. В группе программ среды C++Builder 6.0 (через кнопку «*Пуск*») запустить программу *BDE Administrator* (рис. 7.8).

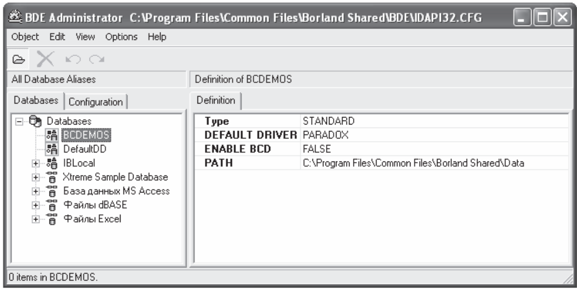


Рис. 7.8. Окно BDE Administrator

В левой части окна расположен список всех псевдонимов (областей). «Двойной щелчок» по области позволяет отразить в правой части окна всю информацию о псевдониме.

3. Отключить пиктограмму *Open or Close* (в противном случае команда *Object/New* будет недоступна).
4. Выполнить команду *Object/New*.
5. В открывшемся окне выбрать драйвер для нового псевдонима (рис. 7.9); по умолчанию это STANDARD для БД Paradox:

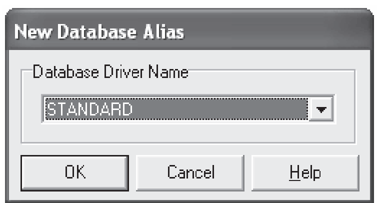


Рис. 7.9. Окно задания драйвера

6. В открывшемся окне *BDE Administrator* (рис. 7.10) в левой части окна ввести имя псевдонима – *aliasDB*, а в правой части окна установить следующие параметры:

~ тип драйвера – STANDARD;

~ драйвер – для Paradox;

~ путь к базе данных устанавливается через кнопку обзора;

~ параметр ENABLE BCD определяет возможность транслирования числовых полей в значения с плавающей запятой или в коды BCD. Если задать значение true, то поля DECIMAL и NUMERIC преобразуются в коды BCD.

7. Сохранить созданный псевдоним командой *Object/Save As* с тем же именем – *aliasDB*.
8. Закрыть *BDE Administrator*.

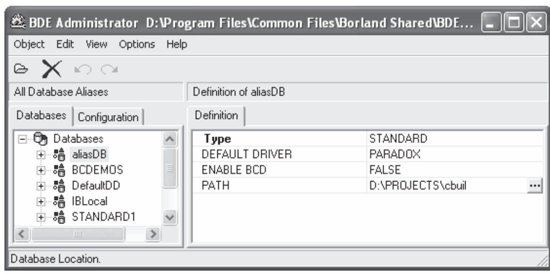


Рис. 7.10. Окно отображения данных о псевдонимах

Работа с *SQL Explorer* производится аналогично работе с программой *BDE Administrator* (окно организовано по тем же принципам).

Для сохранения уже существующей таблицы *student* в новой области необходимо выполнить следующие действия:

1. Открыть *Database Desktop*.
2. Открыть таблицу *student.db* из области WORK.
3. Перейти в структуру таблицы и нажать кнопку <*Save As*>.
4. Сохранить таблицу в новой области под псевдонимом *aliasDB*.
5. Закрыть таблицу и проверить наличие ее в области *aliasDB* любым известным способом.

Вопросы для самоконтроля

1. Каким образом организуется работа с базой данных в среде Borland C++Builder?
2. Какие механизмы доступа к данным используются в среде Borland C++Builder? В чем отличие их организации?
3. Каким образом построена работа с базой данных драйвера ODBC?
4. Какие типы данных имеют фиксированную длину? Охарактеризовать.
5. Что такое псевдоним?
6. Охарактеризовать последовательность создания таблицы.
7. Каким образом осуществляется установка минимального и максимального значений по умолчанию?
8. Каким образом задается псевдоним базы данных?
9. С какой целью используется маска при задании строковых полей?
10. Каким образом осуществляется ввод данных в таблицу?
11. Каким образом отредактировать структуру и записи таблицы?

8. ВИЗУАЛЬНЫЕ КОМПОНЕНТЫ ДЛЯ РАБОТЫ С БАЗОЙ ДАННЫХ

8.1. Компоненты формы, работающие с базой данных

Среда C++Builder 6.0 имеет множество компонентов, которые позволяют организовать доступ к базам данных. Эти компоненты расположены на страницах *Data Access* (доступ к данным) и *Data Control* (управление данными).

Каждое приложение, работающее с базами данных, должно иметь определенный набор компонентов формы.

Компоненты – наборы данных (*data set*) непосредственно связаны с таблицей (базой данных). К таким компонентам относятся *Table, Query, StoredProc*.

Компонент источника данных (*data source*) необходим для связи между набором данных и визуальными компонентами формы – *DataSource*.

Компоненты визуализации и управления данными, такие как *DBGrid, DBText, DBEdit* и другие.

Настройка каждого компонента имеет свое значение и должна быть произведена в определенном порядке (насколько это возможно).

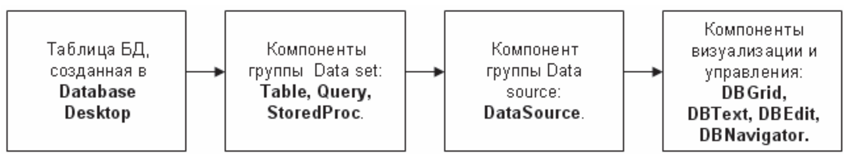


Рис. 8.1. Последовательность создания и настройки компонентов приложения

На рис. 8.1 приведена схема последовательности настройки каждого из компонентов формы.

8.2. Разработка формы просмотра данных таблицы

Для разработки простого приложения (рис. 8.2), которое позволяет просмотреть содержимое таблицы BOOK.db, необходимо выполнить следующие действия:

1. Установить на форму следующие компоненты:

~ *Table* со страницы *BDE* панели компонентов (компонент виден только в режиме редактирования);

~ *DataSource* со страницы *Data Access* (компонент виден только в режиме редактирования);

– *DBGrid* со страницы *Data Control* (компонент виден в любом режиме и представляет собой таблицу-сетку).

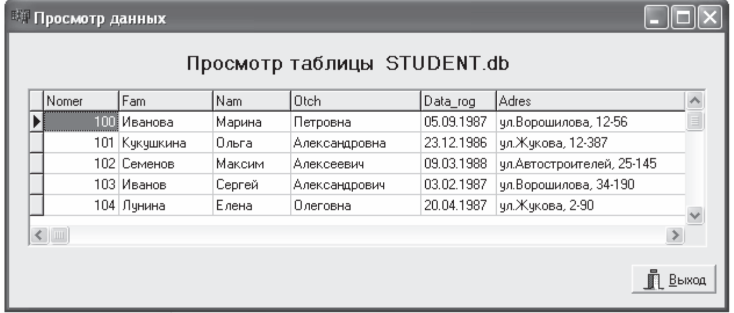


Рис. 8.2. Форма просмотра данных таблицы

2. Установить на форме метку – компонент *Label* со страницы *Standard* и задать ей свойство *Caption:* «Просмотр таблицы STUDENT.db».
3. Выделить метку как объект и установить традиционным для Windows способом данную метку посередине формы в верхней ее части (как заголовок).
4. Сохранить изменения в проекте.
5. Запустить проект на выполнение и убедиться, что форма будет иметь следующий вид (рис. 8.3):

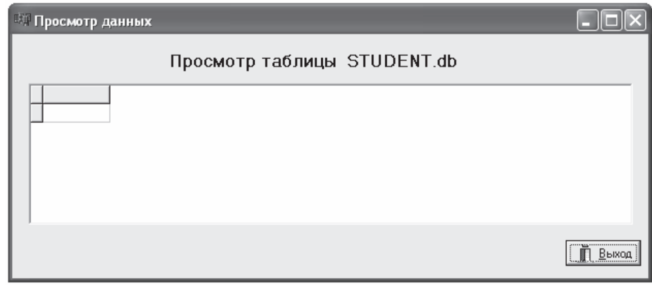


Рис. 8.3. Окно приложения в рабочем режиме

Как видно из рисунка, видимыми являются компоненты *Label* и *DBGrid*. Компоненты *Table* и *DataSource* видимы только при разработке приложения (в режиме редактирования).

6. Вернуться в режим разработки проекта.
7. Установить в указанном порядке (рис. 8.4) следующие свойства для установленных компонентов:

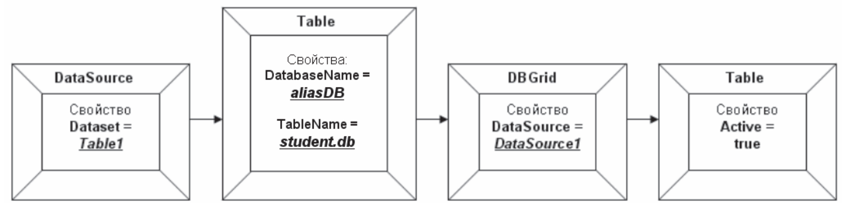


Рис. 8.4. Последовательность задания свойств для компонентов *DataSource, Table и DBGrid*

В компоненте *DataSource* оформляется ссылка на компонент таблицы; в данном случае идентификатор таблицы (свойство *Name*) представлен как *Table1*.

В компоненте *DBGrid* также оформлена ссылка на компонент связи между таблицей и компонентом *Table*; это элемент формы – *DataSource*, имеющий идентификатор *DataSource1* (свойство *Name*). Безусловно, что идентификаторы компонентов (свойства Name) могут быть изменены.

8. Проверить, что после активизации таблицы (установки свойства *Active* в положение *true*) в компоненте *DBGrid1* должны появиться записи таблицы *student.db*.

После задания параметров для описанных компонентов можно запускать приложение на выполнение.

8.3. Использование навигатора для работы с данными

Существует множество других компонентов формы, которые позволяют передвигаться по записям, просматривать отдельную запись, а также многое другое.

Для движения по записям в системе C++Builder существует компонент, который содержит совокупность кнопок для работы с отдельными записями – *навигатор*, представленный компонентом *DBNavigator* со страницы *Data Controls*.

Для добавления навигатора (рис. 8.5) в существующий проект необходимо выполнить следующие действия:

1. Установить на форме компонент *DBNavigator* со страницы *Data Controls* панели компонентов.
2. Активизировать навигатор и установить для него свойство *DataSource*, значение которого равно *DataSource1* (идентификатор компонента *DataSource*).
3. Вернуться в режим редактирования проекта.
4. Активизировать компонент навигатора.

Рис. 8.5. Описание кнопок навигатора

5. В инспекторе объектов найти свойство *Hints* и нажать кнопку обзора (с тремя точками). После чего на экране появится окно «*String List Editor*» со списком названий кнопок навигатора.

Свойство *Hints* компонента *DBNavigator* представлено некоторым списком, поэтому в инспекторе объектов присутствует фраза (*TStrings*).

6. Заменить все английские названия кнопок на русские названия, используя рис. 8.6.

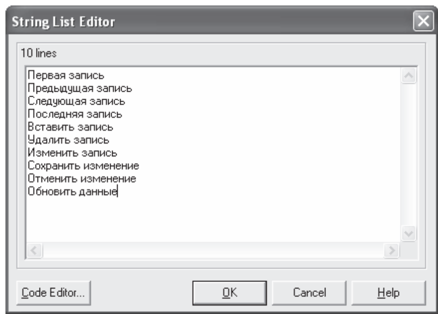


Рис. 8.6. Список названий кнопок навигатора

7. Для активизации заданных подсказок установить значение свойства *ShowHint – true*.

8. Сохранить проект и запустить на выполнение.

9. Проверить работу подсказок подведением указателя мыши к любой кнопке навигатора (рис. 8.7).

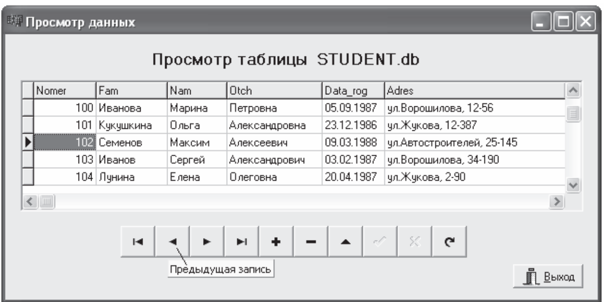


Рис. 8.7. Демонстрация работы подсказок кнопок навигатора

10. Добавить на форму несколько компонентов *DBEdit* со страницы *Data Controls*. Например, для просмотра значений полей «Фамилия», «Имя», «Дата рождения» и «Группа».

11. Добавить на форму столько же компонентов *Label* для подписи полей. Свойство *Caption* позволит задать названия компоненту-метке (рис. 8.7).

12. Задать для каждого компонента *DBEdit* свойства:

~ свойство *DataSource* – значение *DataSource1* (для доступа к таблице student.db);

~ свойство *DataField* – значение *Fam* для фамилии; значение *Nam* для имени; значение *Data_rog* для даты рождения; значение *Gruppa* для группы.

13. Убедиться в том, что данные текущей записи отображены в полях *DBEdit* (рис. 8.8). Это связано с тем, что текущей записью в таблице является первая запись, а текстовые поля привязаны к таблице.

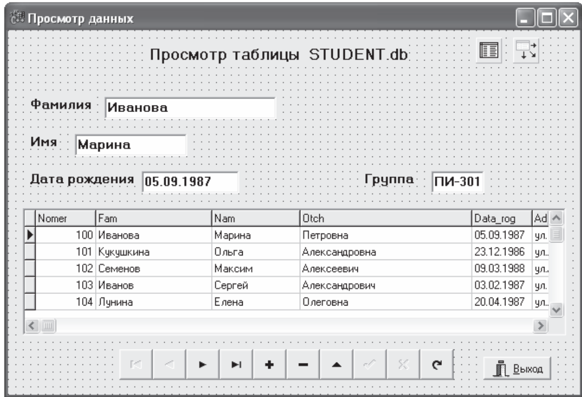


Рис. 8.8. Вид формы в режиме разработки проекта

14. Сохранить проект и запустить на выполнение.

Просмотр данных в загруженном приложении позволяет наблюдать следующее: значения полей типа OLE и Memo не просматриваются; они представлены названиями типов полей (рис. 8.9).

Adres	Phone	Photo	Gruppa	Spez	Prim	Haract	Family
ул.Ворошилова, 12-56	34-98-88	(ParadoxOle)	ПИ-301	351400	(Memo)	(Memo)	False
ул.Жукова, 12-387	32-54-90	(ParadoxOle)	ПИ-302	351400	(Memo)	(Memo)	False
ул.Автостроителей, 25-145		(PARADOXOLE)	ПИ-201	351400	(MEMO)	(MEMO)	False
ул.Ворошилова, 34-190	33-90-11	(PARADOXOLE)	ПИ-302	351400	(MEMO)	(Memo)	False
ул.Жукова, 2-90	34-89-73	(ParadoxOle)	ПИ-303	351400	(Memo)	(Memo)	False

Рис. 8.9. Фрагмент просмотра данных в компоненте DBGrid

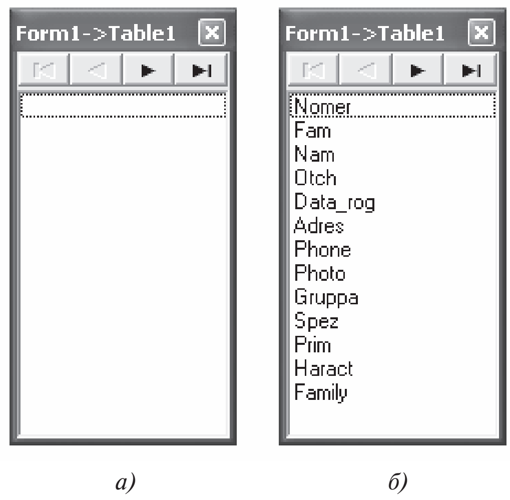
Как видно из рисунка, поля таких типов не просматриваются ни в *Database Desktop*, ни в компоненте *DBGrid*. Работа с графическими данными и полями примечаний возможна в режиме просмотра конкретной записи.

8.4. Установка подписей полей в компоненте DBGrid

Как видно на рис. 8.9, в компоненте *DBGrid* каждый столбец имеет имя, характеризующее название поля. Безусловно, что названия полей не должны быть на латинском языке.

Для изменения заголовков полей в компоненте *DBGrid* необходимо выполнить следующие действия:

1. Открыть проект (или создать), в котором все настройки для просмотра данных в *DBGrid* будут выполнены.
2. Произвести «двойной щелчок» на компоненте *Table* (рис. 8.10*а*).
3. Вызвать контекстное меню открывшегося окна «Редактор полей» *Form1->Table1*.
4. Выполнить команду *Add All Fields*. В результате чего в текущем окне появится список всех полей таблицы *student.db* (рис. 8.10*б*).



а) *б)*

Рис. 8.10. Окно Редактора полей

5. Выделив поле *Nomer*, убедиться в том, что «Инспектор объектов», содержащий свойства объектов, изменился в зависимости от текущего объекта.
6. Изменить значение свойства *DisplayLabel* на значение «Номер».

7. Изменить для всех полей «Редактор полей» заголовки согласно информации, которую несет каждый из столбцов таблицы.

8. Сохранить проект и запустить на выполнение (рис.8. 11).

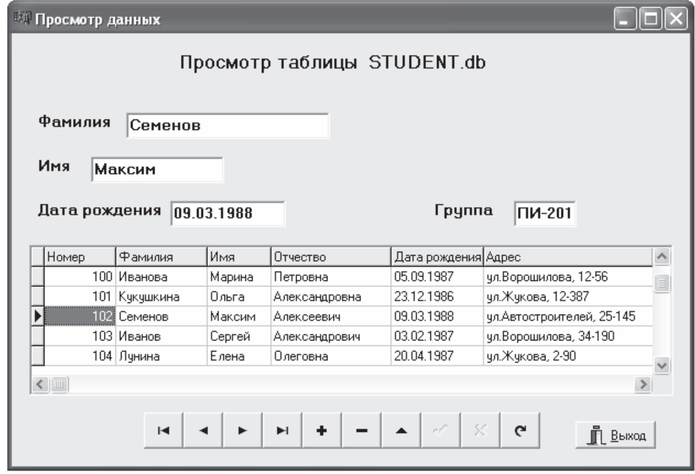


Рис. 8.11. Окно приложения

Вопросы для самоконтроля

1. Какие визуальные компоненты используются для просмотра и обработки данных из таблицы?
2. Чем отличается компонент Button от компонента *BitBtn*?
3. Какие свойства необходимо установить для источника данных для просмотра информации из таблицы?
4. Каково условие просмотра таблицы в компоненте *DBGrid*?
5. Каково назначение навигатора? Какие кнопки он имеет?
6. Некоторые кнопки навигатора работают всегда, а некоторые при определенных обстоятельствах не функционируют. При каких условиях и какие кнопки работают от случая к случаю?
7. Каким образом установить заголовки таблицы на русском языке?

9. СОЗДАНИЕ МНОГОТАБЛИЧНОЙ БАЗЫ ДАННЫХ

9.1. Установка индексов для связывания таблиц

В теории баз данных существуют различные типы связей, устанавливаемые между двумя таблицами. В настоящее время различают три вида связей, которые задаются между родительской и дочерней таблицами:

1) *«один-к-одному»* – только одной записи родительской таблицы соответствует одна запись дочерней таблицы;

2) *«один-ко-многим»* – только одной записи родительской таблицы соответствует одна или более записей дочерней таблицы;

3) *«много-ко-многим»* – одной записи родительской таблицы соответствует одна или более записей дочерней таблицы, и наоборот.

Допускается, что в родительской таблице могут находиться записи, которые не связаны с дочерней таблицей (в дочерней таблице отсутствуют записи для связи с родительской). Кроме того, в настоящее время связь «много-к-одному» потеряла актуальность, хотя в некоторых СУБД направление связи имеет большое значение для обработки данных и их просмотра.

Рассмотрим две таблицы из базы данных ДЕКАНАТ: *student.db* и *ozenka.db*, которые связаны отношением *1:M*. Данное отношение объясняется тем, что у каждого студента за время учебы должно быть большое количество оценок по различным зачетам и экзаменам. Таблица ОЦЕНКА представлена в прил. 1.

Для связи таблиц воспользуемся следующим рисунком (рис. 9.1).

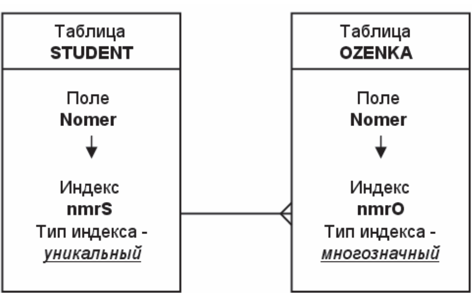


Рис. 9.1. Описание индексов таблиц СТУДЕНТ и ОЦЕНКА

Для этого необходимо в программе *Database Desktop* выполнить следующие действия:

1. Создать еще одну таблицу *ozenka.db* согласно прил. 1.
2. Ввести несколько записей с учетом значений поля *Nomer* таблицы СТУДЕНТ.

3. Открыть таблицу *student.db* и перейти в структуру таблицы.

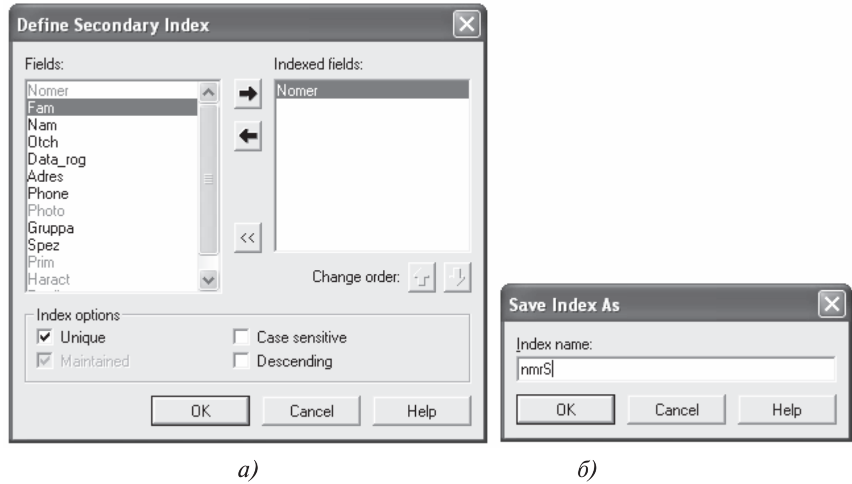
4. В столбце *Key* поля *Nomer* нажать пробел для установки в нем звездочки (первичный ключ).

5. В поле *Table properties* выбрать пункт *Secondary Indexes*.

6. Через кнопку <*Define*> открыть окно *Define Secondary Index*.

7. Кнопкой «*Стрелка вправо*» перевести в правый список поле *Nomer*; установить опцию «*Unique*» для первичного индекса (рис. 9.2*а*).

8. После нажатия кнопки <Ok> в открывшемся окне сохранить индекс под именем *nmrS* (рис. 9.2*б*).



а) *б)*

Рис. 9.2. Установка индексов таблицы

По умолчанию данные в поле индекса сортируются по возрастанию (*Ascending*); при необходимости изменения направления сортировки используется флажок *Descending*.

9. Сохранив все изменения в структуре таблицы, выйти из структуры таблиц.

В случае, если индекс не устанавливается, необходимо проверить значения в поле, по которому создается индекс; возможно, что в этом поле существуют повторяющиеся значения, а этого не должно быть.

10. Открыть таблицу *ozenka.db* (она была создана и заполнена в начале задания).

11. Пиктограммой <*Restructure*> войти в структуру таблицы *ozenka.db*.

12. Активизировать свойство таблицы *Secondary Indexes*.

13. Кнопкой <*Define*> перейти в окно создания индекса.

14. Создать <u>*вторичный*</u> (не уникальный) индекс *nmrO* по полю *Nomer* (рис. 9.3).

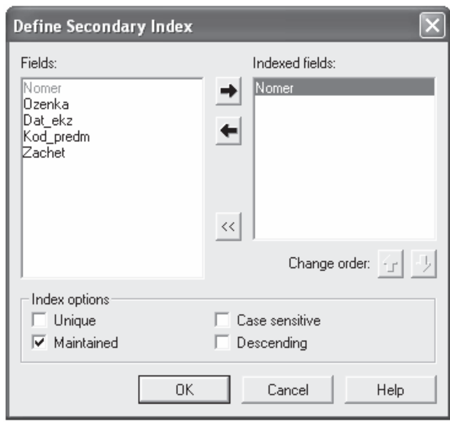


Рис. 9.3. Фрагмент окна структуры таблицы

Для работы с индексами используются следующие кнопки:

<*Define*> – для создания нового индекса;

<*Modify*> – для модификации уже существующего индекса;

<*Erase*> – для удаления выделенного индекса.

15. Выйти из структуры таблицы *ozenka.db*, сохранив все изменения.

На рис. 9.4 приведен фрагмент окна структуры таблицы ОЦЕНКА. В представленном списке находятся два индекса (не уникальных). Второй индекс *predmO* предназначен для связи с таблицей ПРЕДМЕТ (см. прил. 1).

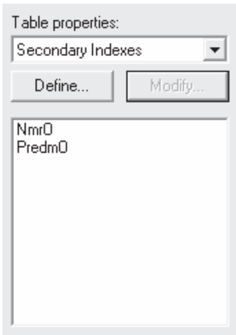
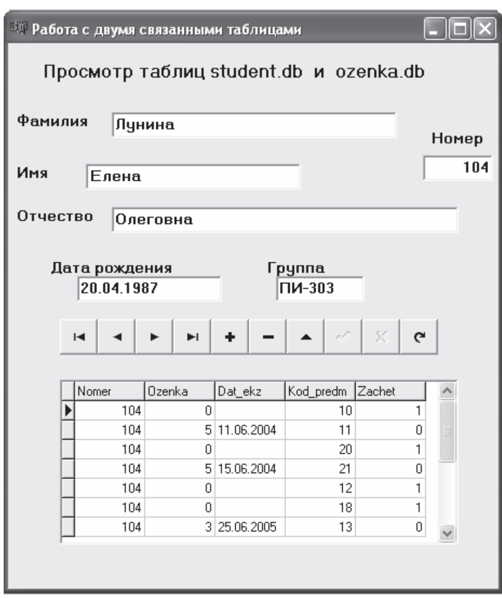


Рис. 9.4. Фрагмент структуры таблицы со списком индексов

После установки индексов начинается организация многотабличных баз данных.

9.2. Разработка формы просмотра данных двух таблиц

Как известно, для просмотра записей из одной таблицы достаточно использовать компонент *DBGrid* или несколько компонентов *DBEdit*, которые настроены на определенную таблицу. Для просмотра двух таблиц (с отношением 1:М) необходимо воспользоваться и компонентами *DBEdit* для отображения данных из родительской таблицы, и компонентом *DBGrid* для демонстрации записей из дочерней таблицы.



Nomer	Ozenka	Dat_ekz	Kod_predm	Zachet
104	0		10	1
104	5	11.06.2004	11	0
104	0		20	1
104	5	15.06.2004	21	0
104	0		12	1
104	0		18	1
104	3	25.06.2005	13	0

Рис. 9.5. Форма просмотра двух таблиц

Для создания формы просмотра данных двух таблиц (рис. 9.5) необходимо выполнить следующие действия:

1. Создать новый проект и сохранить пустой проект на диске и задать форме название – «*Работа с двумя связанными таблицами*».
2. Добавить на форму метку *Label* и задать ей заголовок – «*Просмотр таблиц student.db и ozenka.db*».
3. Добавить на форму две пары компонентов *Table* и *DataSource*. Пусть компоненты *Table1* и *DataSource1* описывают <u>*родительскую*</u> таблицу *student.db*, а компоненты *Table2* и *DataSource2* – <u>*дочернюю*</u> таблицу *ozenka.db*.
4. Открыть «Редактор полей» («двойной щелчок» на компоненте *Table1*) первой таблицы и «перетащить» из «Редактора полей» необходимые поля на форму.

Необходимо заметить, что «перетаскивание» полей позволяет автоматически настроить их на соответствующую таблицу и соответствующие поля. При этом на форме появится само поле и комментарий в виде метки. Достаточно изменить свойство *Caption* у метки и расположить текстовые компоненты надлежащим образом.

5. Добавить на форму компонент *DBNavigator* и настроить его на родительскую таблицу.

6. Добавить на форму компонент *DBGrid* и настроить его на отображение данных второй (дочерней) таблицы.

7. Настроить все компоненты формы соответствующим образом с учетом родительской и дочерней таблиц.

8. Сохранить изменения и запустить проект на выполнение.

9. Убедиться в том, что связь между таблицами не наблюдается, то есть навигатор перемещается по записям первой таблицы, а данные второй таблицы не изменяются, и табличный указатель находится на одной и той же записи (рис. 9.6).

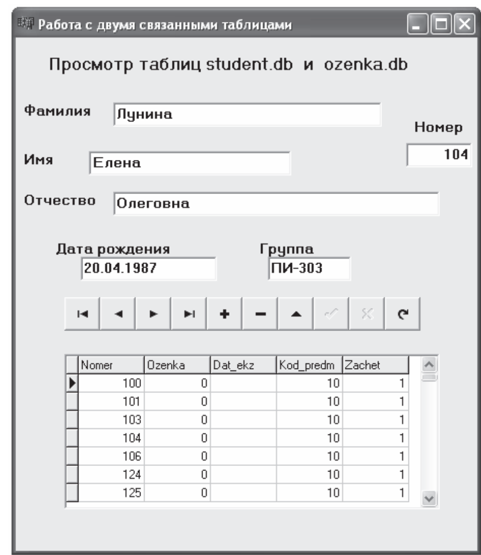


Рис. 9.6. Приложение без связи таблиц

Из рис. 9.6 видно, что связи между таблицами нет, так как в нижней части формы должны отображаться данные, равные номеру студента, то есть 104.

10. Вернуться в режим редактирования проекта.

11. У компонентов *Table11* и *Table2* значение свойства *Active* изменить на значение *False*.

Если пользователю необходимо изменить данные в таблице, которая активна на форме, то произойдет сбой программы-проекта. ***Обязательно*** нужно ***дезактивировать*** таблицу, предназначенную для работы на форме, а потом производить с ней различные изменения в *Database Desktop*. В противном случае, система будет извещать пользователя о том, что таблица в использовании (*use*).

12. Активизировать компонент *Table2* и установить свойство *MasterSource* в значение *DataSource1*.

13. Для свойства *MasterFields* активизировать кнопку обзора (рис. 9.7).

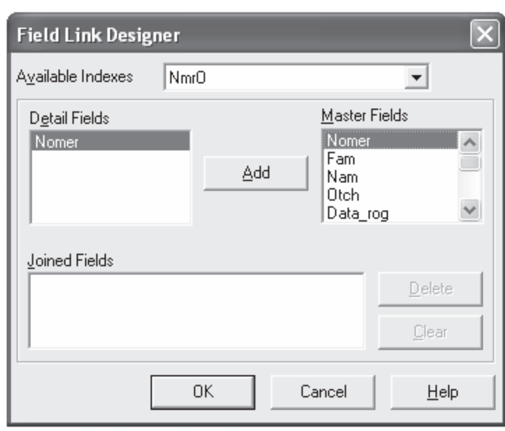


Рис. 9.7. Уставновка связи между таблицами

14. Выбрать из списка *Available Indexes* индекс *nmrO* таблицы ОЦЕНКА.

15. В списке *Detail Fields* (поля дочерней таблицы) выбрать поле *Nomer*.

16. Выбрать в мастер-таблице (родительской) поле *Nomer*.

17. После нажатия кнопки <*Add*> в поле *Joined Fields* появится запись:

Nomer –>Nomer

Это значит, что две таблицы связались по полю *Nomer*.

18. Активизировать компоненты *Table1* и *Table2* – значение свойства *Active* должно быть *true*.

Необходимо убедиться в работе двух таблиц: при переходе на другую запись родительской таблицы в компоненте *DBGrid* отражаются только те записи дочерней таблицы, которые соответствуют в текущий момент записи родительской таблицы.

Вопросы для самоконтроля

1. Какие типы отношений существуют для связи таблиц? Охарактеризовать их.
2. Что представляет собой первичный ключ?
3. Какие особенности характеризуют вторичный индекс по отношению к первичному?
4. Как задается первичный индекс?
5. Как устанавливается связь между таблицами на форме просмотра двух взаимосвязанных таблиц?
6. Ситуация: таблица не открывается, выдается ошибка об использовании таблицы. Охарактеризовать ситуацию и ее разрешение.

10. ДОПОЛНИТЕЛЬНЫЕ ВОЗМОЖНОСТИ ПО РАБОТЕ С БАЗАМИ ДАННЫХ

10.1. Ввод данных в поля типа OLE и Memo

Как уже говорилось ранее, просмотреть информацию из полей *OLE* и *Memo* невозможно ни в программе *Database Desktop*, ни в компоненте *Grid*. Это возможно в ситуации, когда данные из таблицы просматриваются по одной записи. Данный режим необходим для того, чтобы ввести нужную информацию в таблицу.

Предположим, что все графические файлы (фотографии) студентов существуют на диске в определенном каталоге.

Для ввода в указанные поля данных необходимо произвести следующие действия:

1. Установить на форме компоненты *Table*, *DataSource* и *DBNavigator*.
2. Настроить все компоненты на таблицу СТУДЕНТ; свойство *Active* компонента *Table* должно быть *false*.
3. Произвести «двойной щелчок» на компоненте *Table* и из «Редактора полей» «перетащить» необходимые поля на форму (рис. 10.1).

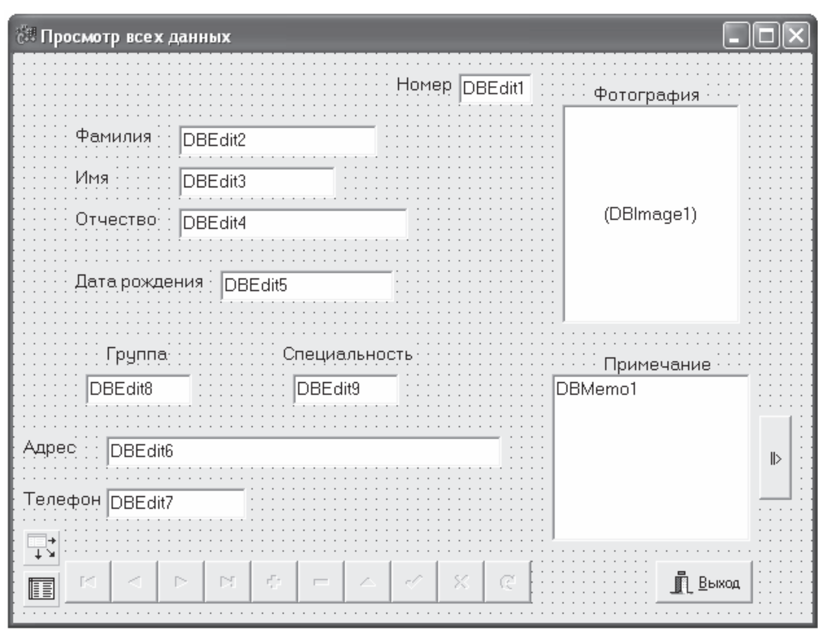


Рис. 10.1. Рабочий режим приложения

Необходимо заметить, что все поля, которые были установлены на форме «перетаскиванием», автоматически настраиваются на конкретную таблицу согласно компонентам *Table* и *DataSource*.

4. Активизировать таблицу (свойство *Active*) и запустить на выполнение приложение.

5. Используя навигатор, остановиться на записи, в которой нет графического файла в поле Photo.

6. Свернуть приложение и обратиться к редактору просмотра графических файлов.

7. В буфер обмена скопировать графический файл (можно воспользоваться комбинацией клавиш <Ctrl-C>).

8. Вернуться к приложению и в программном режиме произвести «щелчок» на компоненте *DBImage1*. Компонент активизируется.

9. Командой вставки <Ctrl-V> вставить графическое изображение в активное окно компонента *DBImage1*.

10. Аналогичным образом вставить фотографии для всех необходимых студентов.

Необходимо заметить, что при просмотре записей через компонент *DBGrid* наличие данных в полях типа *DBImage* и *Memo* характеризуется прописными буквами.

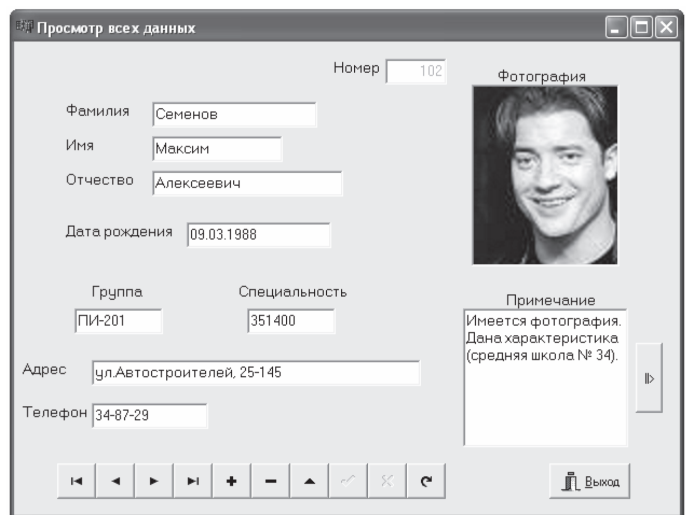


Рис. 10.2. Программный режим приложения

Ввод данных в поле *Memo* осуществляется также в программном режиме при условии, что данное поле не заблокировано для ввода данных.

10.2. Вычисляемые поля. Ниспадающий список

Иногда при просмотре данных из каких-либо таблиц необходимо иметь дополнительные столбцы, которые не существуют в таблицах, а рассчитываются при работе приложения. Поля, которые создаются в рабочем порядке на основе уже имеющихся данных, получили название *вычисляемых полей (calculated fields)*.

Рассмотрим пример создания нового поля «Возраст» при выполнении проекта «Просмотр таблицы СТУДЕНТ». Для выполнения данного задания необходимо выполнить следующие действия:

1. Установить на форме компоненты *Table, DataSource, DBGrid* и *DBNavigator*.
2. Настроить все компоненты формы для просмотра данных таблицы СТУДЕНТ и проверить работу компонентов.
3. Дезактивировать таблицу (свойства *Activate = false*).
4. Произвести «двойной щелчок» на компоненте *Table1* и открыть «Редактор полей».
5. Через контекстное меню выполнить команду *Add All Fields* (добавить все поля).
6. Через контекстное меню выполнить команду *New Field*, после чего откроется окно с одноименным названием (рис.10.3).
7. Ввести название поля (*Vozr*), выбрать тип данных (*Integer*) и активизировать опцию *Calculated*.

Рис. 10.3. Окно создания нового поля

После нажатия кнопки *<Ok>* название поля должно появиться в списке полей. Если система вывела диалоговое окно с предупреждением, то, вероятно, пользователь забыл дезактивировать таблицу.

8. Перейти на вкладку *Events* «Инспектора объектов» при активизации компонента *Table1*.

9. В событии *OnCalcFields* произвести «щелчок» мыши.

10. В открывшемся окне программного кода проекта в добавленную функцию *Table1CalcFields* ввести код (листинг 10.1):

Листинг 10.1. Фрагмент приложения по обработке вычисляемого поля

```
void __fastcall TForm1::Table1CalcFields(TDataSet *DataSet)
{
     //объявление беззнакового целого числа d (год рождения студента)
unsigned short d;
     //объявление строковых переменных для промежуточного вычисления
String s, s1;
     //считывание из поля дня рождения Table1Data_rog строкового
      значения в s
s=Table1Data_rog->AsString;
     //вырезка из дня рождения s четырех последних символов –
     года рождения
s1=s.SubString(7,4);
      //преобразование года рождения в целое число d
d=StrToInt(s1);
     //установка в поле таблицы (в каждой записи) значения возраста
     // ТекущийГод – ГодРождения
Table1Vozr->Value = Year-d;
}
```

В программном коде данные о дате рождения считываются не из поля *Data_rog*, а из поля *Table1Data_rog*, идентификатор которого автоматически создается системой именно для программирования. Идентификатор *Table1Vozr* также был задан автоматически при создании нового поля (область *Component*). Идентификатор *Year* инициирован далее.

11. Активизировать форму и войти в программный код события *OnCreate*.

12. Ввести в программный код данного события команду по считыванию текущей даты:

```
Date().DecodeDate(&Year, &Month, &Day);
```

В данной команде используется стандартная функция *Date()*, которая возвращает значение текущей даты, например, в формате 23.09.2007. Метод *DecodeDate* позволяет распределить по переменным *Year*, *Month* и *Day* значения из текущей даты.

13. Для использования переменных *Year, Month, Day* необходимо объявить их глобальными в начале программного модуля*.cpp*:

```
unsigned short Year, Month, Day;
```

14. При необходимости в программный код события *OnCreate* добавить активизацию таблицы, а в программный код события *OnDestroy* формы добавить дезактивацию таблицы.

15. Сохранить проект и запустить на выполнение (рис. 10.4):

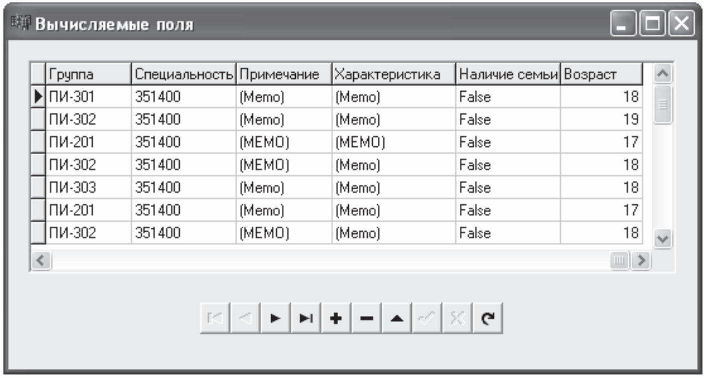


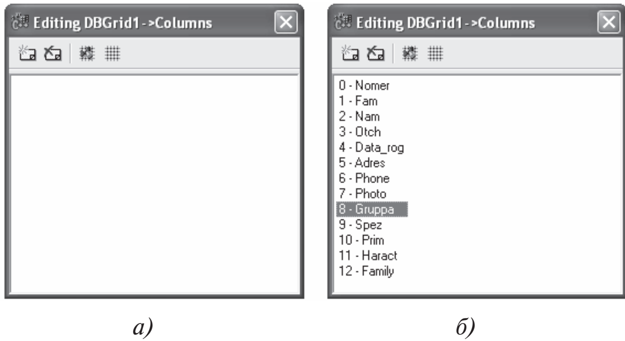
Рис.10.4. Окно с вычисляемым полем «Возраст»

Из рис. 10.4 видно, что при вводе данных часто повторяется одна и та же информация, например группа. Для упрощения работы с такими данными можно использовать так называемые *ниспадающие списки*. Необходимо помнить, что информация о группах в данной задаче не представлена в виде справочника, т. е. отдельной таблицы.

Рассмотрим создание списка групп. Для этого необходимо выполнить следующие действия:

1. После открытия проекта просмотра данных таблицы СТУДЕНТ необходимо дезактивировать таблицу.

2. Войти в «Редактор столбцов» компонента DBGrid «двойным щелчком» мыши (рис. 10.5*а*).



а) *б)*

Рис. 10.5. Окно Редактора Столбцов компонента DBGrid

3. Через контекстное меню добавить все поля командой *Add All Fields*.

4. Активизировать поле *Gruppa* в «Редакторе столбцов» (рис. 10.5*б*).

5. Установить свойство *ButtonStyle = cbsAuto* (разрешение на возможность появления ниспадающего списка при вводе и/или редактировании данных).

6. Произвести «двойной щелчок» на свойстве *PickList*, в результате чего открывается окно *String List Editor*.

7. Ввести список групп.

После сохранения проекта можно проверить работу списка групп (рис. 10.6).

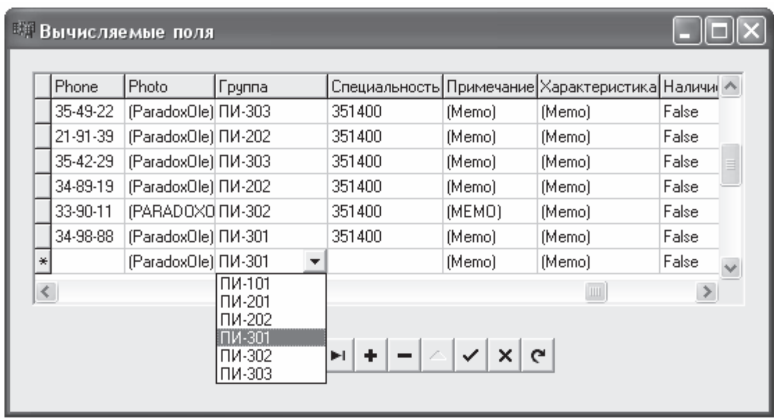


Рис. 10.6. Использование ниспадающего списка ввода данных

Необходимо заметить, что ниспадающие списки используются в ситуациях, когда невыгодно использовать новую таблицу. Например, если по группам, кроме названий, нет никаких данных, то созданная отдельная таблица будет содержать всего два поля: поле первичного ключа и название группы. Кроме того, сам список не является большим.

Иная ситуация складывается, когда речь идет о специальности. В данном случае есть смысл вывести информацию о специальностях в отдельную таблицу и ввести такие данные, как:

- ~ код специальности (для связи с другими таблицами);
- ~ название специальности;
- ~ шифр специальности;
- ~ головной вуз;
- ~ ФИО председателя УМО (учебно-методического объединения) и т. п.

В данной системе специальность оформлена в таблице СТУДЕНТ для простоты работы с данными. Как говорилось ранее, в представенной задаче используется упрощенный вариант базы данных (наименьшее количество таблиц).

10.3. Использование маски ввода

При создании таблицы в программе *Database Desktop* использовалась маска ввода для поля *Телефон*. Для удобства ввода данных по какому-либо шаблону можно также использовать специальные компоненты, предназначенные для формы, – *MaskEdit* и *DBEdit* (через «Редактор полей»).

Компонент *MaskEdit* является самостоятельным компонентом в отличие от *DBEdit*, который отражает данные конкретного поля.

Свойство *EditMask* всех компонентов использует следующие шаблоны маски (табл. 11).

Таблица 11

Символы шаблона маски

Символ	Описание символа шаблона маски
!	Наличие символа – замена недостающих символов пробелами; отсутствие – пробелы размещаются в конце строки
>	Все следующие за ним символы записываются в верхнем регистре (пока не сменится регистр)
<	Все следующие за ним символы записываются в нижнем регистре (пока не сменится регистр)
<>	Анализ регистра не производится
\	Используется для того, чтобы различать обычные символы или специальные. Так, например, комбинация символов \> означает, что это не шаблон маски, а обычный символ «больше». Другими словами, данный символ блокирует признак шаблонности для идущего за ним символа
L	Только буква
l	Буква или ничего
A	Только буква или цифра
a	Буква, цифра или ничего
C	Любой символ
c	Любой символ или ничего

Продолжение табл. 11

Символ	Описание символа шаблона маски
0	Только цифра
9	Цифра или ничего
#	Цифра, +, – или ничего
:	Разделитель для времени
/	Разделитель для даты
_	Автоматическая вставка пробела

Компонент *MaskEdit* расположен на вкладке *Additional,* и для него достаточно вызвать редактор маски через свойство *EditMask*.

Для создания шаблона маски текстового поля таблицы необходимо выполнить следующие действия:

1. Установить на форме все необходимые компоненты и настроить их на просмотр какой-либо таблицы, например, КЛИЕНТ.
2. Вызвать «Редактор полей» и добавить в него все поля текущей таблицы.
3. Активизировать необходимое поле и через свойство *EditMask* вызвать окно *Input Mask Editor* (рис. 10.8).
4. Воспользовавшись правым списком, выбрать за основу какую-либо маску, например, маску даты.
5. В поле *Input Mask* отредактировать маску или ввести собственный вариант.
6. В поле *Character for Blanks* автоматически появляется символ для заполнения, который вводится в третье поле маски.
7. В поле *Test Input* ввести пример даты (рис. 10.8).
8. Нажав кнопку <Ok>, перейти к форме проекта.

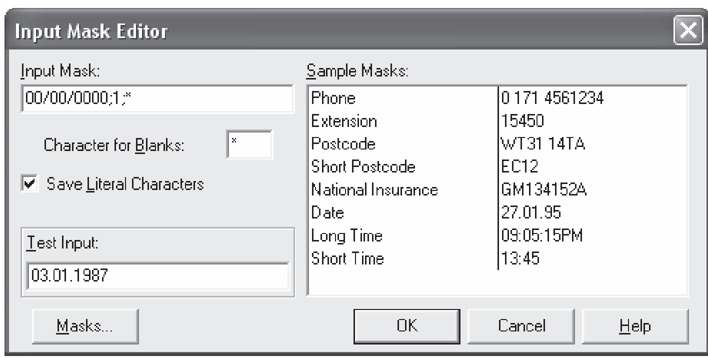


Рис. 10.8. Окно ввода шаблона маски

На рис. 10.9 приведен интерфейс формы просмотра данных таблицы КЛИЕНТ. В данном примере специально подобраны поля, для которых необходимо в «хорошей» программе определить маски ввода данных.

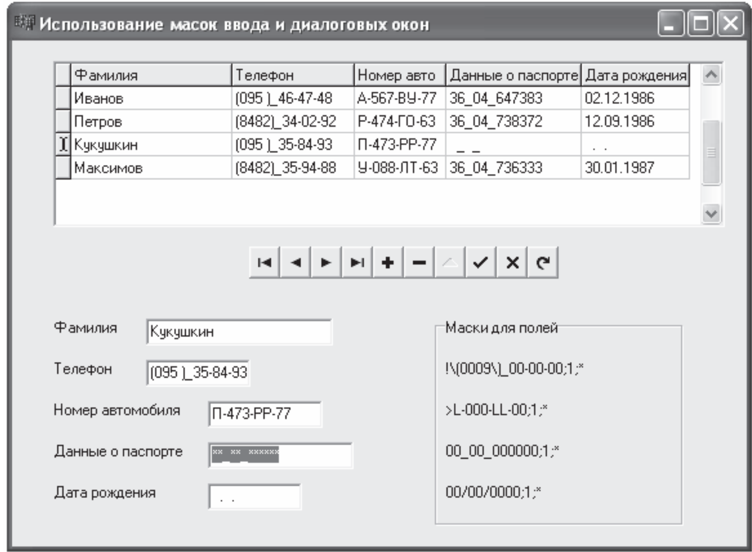


Рис. 10.9. Приложение с примерами масок

Как видно из рисунка, при вводе данных и/или редактировании маска ввода автоматически появляется в необходимом поле (если она для него определена). В правой части формы приведена панель с масками, которые определены для полей, находящихся в левой части формы.

Разработка приложений затрагивает одни и те же данные. В связи с этим возникает потребность в создании некоторого архива масок для других приложений. Это возможно с помощью создания файла с расширением *dem*. Этот файл должен иметь в каждой строке через разделитель | три блока данных:

название маски | пример – вводимые символы | маска

Создание двух первых масок (согласно вышеописанному проекту) заключается в следующих шагах:

1. Создать текстовый (через блокнот) файл.
2. Ввести информацию о масках в следующем формате:
   Телефон с кодом города | 8482509824 | \(0009\)_00-00-00;1;*
   Номер автомобиля | A837FF63 | >L-000-LL-00;1;*
3. Сохранить файл под любым именем с расширением.*dem* в каталоге..*\Borland\CBuilder\Bin*.

Input Mask Editor
Input Mask: \(0009\)_00-00-00;1;*
Character for Blanks: *
Save Literal Characters
Test Input: (****)_**-**-**
Sample Masks:
Телефон с кодом города (8482)_50-98-24
Номер автомобиля A-837-FF-63
Masks... OK Cancel Help

Рис. 10.10. Окно Input Mask Editor

4. Проверить работу файла масок, загрузив его кнопкой <*Masks*> окна *Input Mask Editor*.

Как уже говорилось, наличие масок в текстовых полях (при вводе и/или редактировании) позволяет пользователю чувствовать себя спокойно в вопросе правильного ввода данных. Это подсказка и помощь для пользователя инфрмационной системы.

10.4. Дополнительные возможности просмотра данных

В различных приложениях бывают ситуации, когда необходимо вывести ту или иную информацию по требованию. Например, в предыдущем приложении существует кнопка в правой нижней части формы (рис. 10.2), которая дополнительно открывает окно с характеристикой и фотографией (для подтверждения).

Несмотря на то, что данные берутся из одной таблицы, сложность данной задачи заключается в том, что нужно передать информацию на другую форму (рис. 10.11).

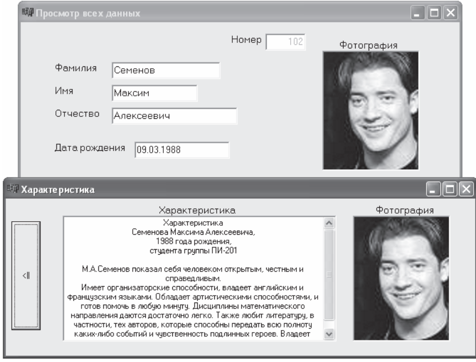


Рис. 10.11. Приложение с передачей данных на другую форму

Для создания второй формы, отображающей необходимую информацию из той же таблицы, нужно выполнить следующие действия:

1. Создать новую форму, например, под именем *Form2* (с учетом того, что главная форма имеет идентификатор *Form1*).
2. Расположить на форме компоненты *DBMemo* и *DBImage* традиционным образом (но не «перетаскиванием»).
3. Выполнить команду *File/Include Unit Hdr*, что позволит второй форме видеть необходимую информацию с первой формы. Активной должна быть именно вторая форма.
4. Организовать на первой форме, например, кнопку для вызова второй формы с указанием команды: *Form2->Show();*
5. Необходимо заметить, что вторая форма может быть невидима для первой, поэтому нужно в программном модуле первой формы подключить заголовочный файл второй формы: ***#include "un_db_2_2.h";***
6. Активизировать вторую форму и установить следующие свойства:

– для компонента *DBMemo*

DataSource ***= Form1->DataSource1***
DataField ***= Haract***

– для компонента *DBImage*

DataSource ***= Form1->DataSource1***
DataField ***= Photo***

Рассмотрим второй пример вывода данных из одной таблицы, используя компонент *DBCtrlGrid* (рис. 10.12).

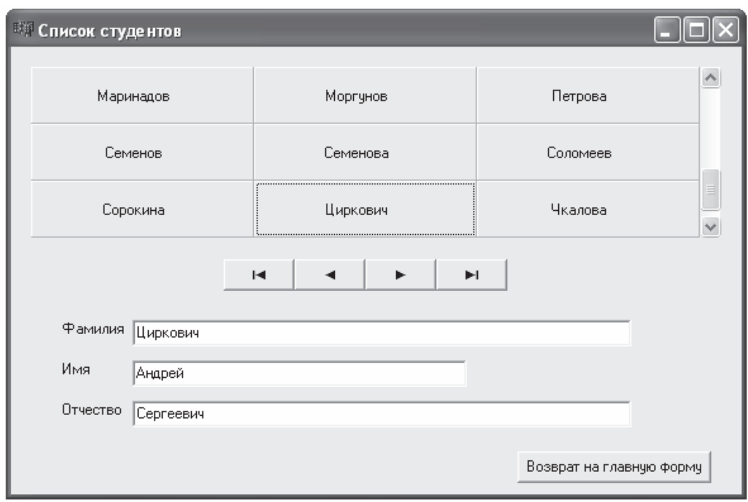


Рис. 10.12. Приложение с компонентом DBCtrlGrid

Необходимо заметить, что компонент *DBCtrlGrid* позволяет в виде списка представить последовательность фамилий (и/или других полей). Переход по фамилиям осуществляется через клавиши движения курсора или через упрощенный вариант навигатора.

Что касается последнего, то для упрощения стандартного навигатора было использовано свойство *VisibleButtons*, в котором видимыми были установлены только первые четыре кнопки (*nbFirst, nbPreor, nbNext* и *nbLast* – константы кнопок в навигаторе).

В нижней части формы установлены стандартные компоненты *DBEdit*, которые позволяют автоматически просмотреть содержимое текущей записи. Это сделано для проверки компонента *DBCtrlGrid*.

Для создания такового приложения необходимо выполнить следующие действия:

1. Установить на форме компоненты *Table*, *DataSource*, *DBCtrlGrid* и из «Редактора полей» перетащить на форму поля фамилии, имени и отчества (которые уже настроены на соответствующую таблицу).

2. Активизировать компонент *DBCtrlGrid* и установить для него свойство *DataSource,* равное значению *DataSource1* (рис. 10.13), свойство *ColCount* = 3 (количество столбцов), свойство *RowCount* = 3 (количество строк).

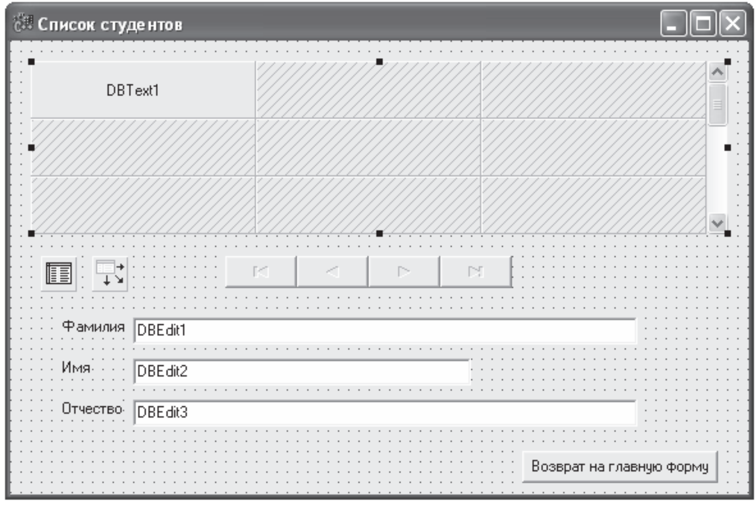


Рис. 10.13. Установка компонента DBCtrlGrid

3. Установить в первую ячейку (специально для этого предназначенную) компонент *DBText* (например, с идентификатором *DBText1*).

4. Задать компоненту *DBText1* следующие значения свойств: *Data Source* = *DataSource1*, *DataField* = *Fam*.

5. После активизации таблицы на форме появятся соответствующие значения компонентов *DBText* и *DBEdit* (соответствующие данным первой записи таблицы).

Рассмотрим третий пример вывода данных (рис. 10.14). В просмотре данных будут участвовать две таблицы: СТУДЕНТ и ОЦЕНКА; для вывода информации о предмете будет использована третья таблица ПРЕДМЕТ (с учетом связи между таблицами ПРЕДМЕТ и ОЦЕНКА). Последняя из трех таблиц необходима для того, чтобы получать полную информацию об оценках.

Связь между двумя таблицами

Моргунов	ПИ-202
Васечкина	ПИ-301
Иванова	ПИ-301

Номер	Оценка	Дата экзамена	Код предмета	Зачет	Преподаватель	Семестр
100	0		10	1	Операционные системы	1
100	5	11.06.2004	11	0	Операционные системы	2
100	0		20	1	Информационная культура	1
100	4	15.06.2004	21	0	Информационные технологии	2
100	0		12	1	Базы данных	3
100	0		18	1	Высокоуровневые методы и язь	3

Рис. 10.14. Окно просмотра данных таблиц СТУДЕНТ, ОЦЕНКА и ПРЕДМЕТ

Реализуем сначала первый этап проекта: просмотр данных из двух таблиц с использованием компонентов *DBCtrlGrid* для первой таблицы и *DBGrid* – для второй.

Для создания данной части проекта необходимо выполнить следующие действия:

1. Создать новый проект, сохранив на диске все нужные файлы; запустить на выполнение программу *DataBase Desktop* и загрузить таблицу *student.db*.

2. Создать для таблицы СТУДЕНТ еще один индекс (не уникальный и составной) по полям *Группа* и *Фамилия*, сохранив индекс под именем *grupfam* (на рис. 10.15 в нижней части формы записаны невидимые метки, напоминающие о необходимых индексах).

Индекс *grupfam* необходим таблице не для связи (для этого уже существует индекс *nmrS*), а для сортировки данных при просмотре. Вторая таблица – ОЦЕНКА – использует индекс *nmrO*, который используется для связи с таблицей СТУДЕНТ (индекс связи *nmrS*).

3. Вернуться в рабочий режим проекта.

4. Установить на форме компоненты *Table* и *DataSource* (для первой таблицы – с индексом 1 – *Table1* и *DataSource1*, для второй таблицы – с индексом 2 и т. д.).

5. Каждый компонент *DataSource* настроить на соответствующую таблицу *Table*.

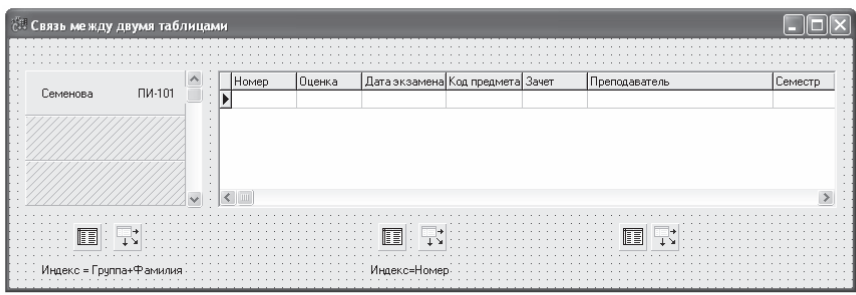


Рис. 10.15. Интерфейс формы просмотра нескольких таблиц

6. Для компонента *Table1* установить свойства: *DatabaseName = aliasDB, TableName = student, IndexName = grupfam*.
7. Установить на форме компонент *DBCtrlGrid* и *DBGrid*, настроив первый компонент на *DataSource1*, а второй – на *DataSource2*.
8. Установить для компонента *DBCtrlGrid* количество столбцов – 1, а количество строк – 3.
9. В первой ячейке текущего компонента установить компоненты *DBText1* и *DBText2*, настроив их на фамилию и группу соответственно.
10. Используя «Редактор полей», изменить заголовки столбцов компонента *DBGrid1*.
11. Для компонента *Table2* установить следующие свойства: *Master Source = DataSource1*, *MasterFields = nmrO, IndexName = Nomer*.
12. Активизировав таблицы, убедиться в том, что связь между таблицами работает.
13. Сохранить проект и проверить его работу, загрузив на выполнение.

Второй этап разработки данного проекта будет заключаться в следующем: необходимо, используя связь между второй и третьей таблицами, вывести в *DBGrid* данные из третьей таблицы (то есть в компоненте *DBGrid* вывести информацию из двух таблиц).

Для этого нужно в проекте (на форме) выполнить следующие действия:

1. Дезактивировать таблицы.
2. У второй таблицы (ОЦЕНКА) вызвать «Редактор полей».
3. Через контекстное меню выполнить команду *New Field*.
4. В открывшемся окне с одноименным названием установить данные согласно табл. 12.
5. Убедиться в том, что система не выдала никаких ошибок и «Редактор полей» получил следующий вид (рис. 10.17):
6. Задать каждому полю заголовки через свойство *DisplayLabel*.
7. Активизировать таблицы.

Таблица 12

Описание опций окна New Field

Опция окна	Описание опции	Что сделать?
Name	Название нового поля	Ввести идентификатор поля, например, *Prep*. Он будет отражаться в списке «Редактор полей»
Component	Название поля для программирования	Идентификатор – *Table2Prep*. Данное название задается автоматически и используется в программировании
Type	Тип данных поля	Из списка выбрать тип данных – *String* (строковый тип)
Size	Размер поля	Ввести длину поля, например, *30*
Field Type	Вид создания нового поля	Из предложенных переключателей выбрать *Lookup* для задания параметров нового поля
Key Fields	Поле связи первой таблицы, предназначенной для отображения данных	Из списка выбрать индекс дочерней таблицы – *Kod_predm* (таблица ОЦЕНКА), по которому она связана с родительской (ПРЕДМЕТ). Именно в дочерней таблице будет отображаться информация из родительской
Dataset	Вторая таблица с полем связи	Из списка выбрать родительскую таблицу – ПРЕДМЕТ (*Table3*)
Lookup Key	Ключевое поле второй таблицы, необходимое для связи с первой таблицей	Из списка выбрать индекс *Kod_predm* родительской таблицы (ПРЕДМЕТ) для связи с дочерней таблицей (ОЦЕНКА)
Result Field	Просматриваемое поле второй таблицы	Из списка выбрать поле *Predmet*, которое будет добавлено из родительской таблицы ПРЕДМЕТ в таблицу ОЦЕНКА для просмотра взаимосвязанных данных ПРЕДМЕТ + ОЦЕНКА

Система дает подсказки для выбора данных в поле *Lookup definition*.

Рис. 10.16. Окно New Field

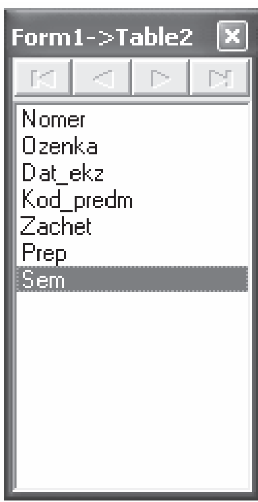


Рис. 10.17. Окно Редактора Полей таблицы Table2

Необходимо заметить, что такой подход – вывод данных из двух взаимосвязанных таблиц – позволяет просматривать содержимое двух таблиц в одном компоненте – *DBGrid*, без использования дополнительных элементов *DBEdit*.

10.5. Диалоговые окна

Очень часто в приложениях используются диалоговые окна для вывода различного рода информации, например, сообщений об ошибке, сообщений-подтверждений и т. п.

Для этого используются различные функции вызова диалоговых окон. Рассмотрим их.

1. Функция *ShowMessage (Msg)*. Сообщение *Msg* типа *AnsiString*.

2. Функция *ShowMessageFmt (Msg, Param, ParamSize),* где Msg – сообщение типа *AnsiString, Param* – массив параметров*, ParamSize* – размер этого массива параметров.

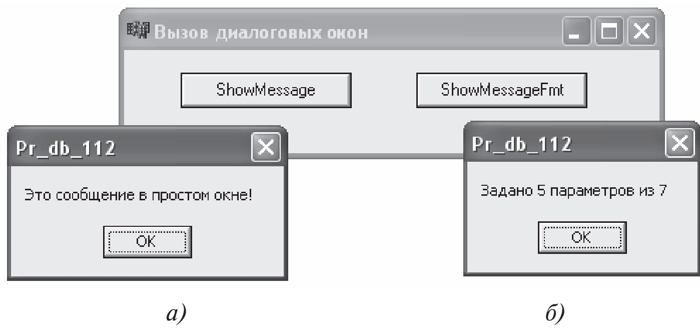
Для передачи массива в функцию используется макрос OPENARRAY. Эта функция описана в стандартном файле *sysdefs.h*. Макрос обеспечивает передачу в функцию открытого массива, содержащего до 19 элементов.

В программном коде события кнопки <*ShowMessage*> введено (рис. 10.18):

ShowMessage (“Это сообщение в простом окне!”);

В программном коде события кнопки <*ShowMessageFmt*> введено:

```
short a=5;
short b=7;
ShowMessageFmt(“Задано%d параметров из%d”,
                              OPENARRAY(TVarRec, (a,b)));
```



а) *б)*

Рис. 10.18. Пример вывода диалоговых окон

3. Функция *MessageDlg* предназначена для задания вопроса и получения на него ответа.

Формат данной функции следующий:

MessageDlg (Msg, DlgType, Buttons, HelpCtx),

где *Msg* – сообщение, *DlgType* – тип диалогового окна (константа), *Buttons* – перечисление необходимых кнопок (константы), *HelpCtx* – экран контекстной справки при нажатии клавиши <F1>.

В табл. 13 приведены константы типов диалоговых окон:

Таблица 13

Константы типов диалоговых окон

Значение	Описание
mtConfirmation	Окно подтверждения, содержащее вопросительный знак (рис. 14.19*а*)
mtInformation	Информационное окно, содержащее символ “i” (рис. 14.19*б*)
mtError	Окно ошибок, содержащее красный стоп-сигнал (рис. 14.20*а*)
mtWarning	Окно замечаний, содержащее желтый восклицательный знак (рис. 14.20*б*)
mtCustom	Заказное окно без рисунка, имеющее заголовок проекта (рис. 14.21)

Параметр *Buttons* представляет собой некое множество, в которое необходимо включать различного вида кнопки. Возможные значения видов кнопок приведены в табл. 14.

Таблица 14

Значения видов кнопок

Значение кнопки	Описание
mbYes	Кнопка с надписью “Yes”
mbNo	Кнопка с надписью “No”
mbOK	Кнопка с надписью “OK”
mbCancel	Кнопка с надписью “Cancel”
mbHelp	Кнопка с надписью “Help”
mbAbort	Кнопка с надписью “Abort”
mbRetry	Кнопка с надписью “Retry”
mbIgnore	Кнопка с надписью “Ignore”
mbAll	Кнопка с надписью “All”

Необходимые кнопки заносятся в *Buttons* операцией “<<”, поскольку параметр *Buttons* является множеством. Если не занести в этот параметр ничего, в окне не будет ни одной кнопки и пользователю придется закрывать окно системными кнопками Windows.

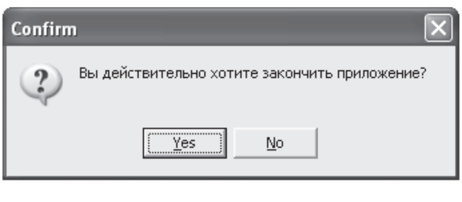
Кроме множества значений, соответствующих отдельным кнопкам, определены три константы, соответствующие множествам часто используемых сочетаний кнопок:

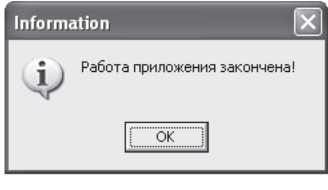
mbYesNoCancel – сочетание кнопок «*Yes*», «*No*» и «*Cancel*»;
mbOkCancel – сочетание кнопок «*Ok*» и «*Cancel*»;
mbAbortRetryIgnore – сочетание кнопок «*Abort*», «*Retry*» и «*Ignore*».

Параметр *HelpCtx* определяет экран контекстной справки, соответствующей данному диалоговому окну. Этот экран справки будет появляться каждый раз, когда пользователь нажмет клавишу <F1>. Если в приложении справка не планируется, то этот параметр должен равняться нулю.

Функция *MessageDlg* возвращает одно из значений – нажатий кнопок:

mrNone	*mrAbort*	*mrYes*
mrOk	*mrRetry*	*mrNo*
mrCancel	*mrIgnore*	*mrAll*





а) *б)*

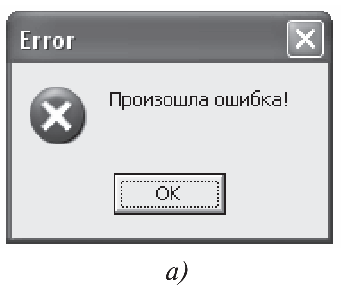
Рис. 10.19. Диалоговые окна при завершении приложения

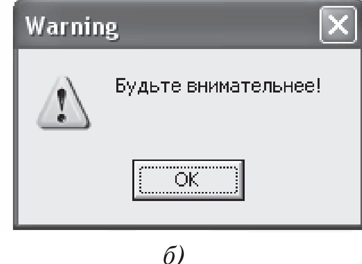
Для вывода таких окон используется следующий фрагмент приложения:

```
void __fastcall TForm1::BtnConfInfClick(TObject *Sender)
{
    if (MessageDlg ("Вы хотите закончить приложение?",
      mtConfirmation, TMsgDlgButtons() << mbYes << mbNo, 0) == rYes)
    {
      MessageDlg ("Работа приложения закончена!", mtInformation,
                                        TMsgDlgButtons() << mbOK, 0);
      Close();
    }
}
```

Для вывода таких окна-ошибки (рис. 10.20*а*) и окна-предупреждения (рис. 10.20*б*) используется код приложения:

```
void __fastcall TForm1::BtnErrWarnClick(TObject *Sender)
{
    MessageDlg("Произошла ошибка!", mtError,
                                        TMsgDlgButtons() << mbOK, 0);
    MessageDlg("Будьте внимательнее!", mtWarning,
                                        TMsgDlgButtons() << mbOK, 0);
}
```





а) *б)*

Рис. 10.20. Демонстрация окна-ошибки и окна-предупреждения

Окно с заголовком проекта можно вывести следующим образом:

```
void __fastcall TForm1::BtnCustClick(TObject *Sender)
{
    switch ( MessageDlg ("Занести запись в базу данных?",
                                  mtCustom, mbYesNoCancel, 0))
    {
      case mrYes: Table1->Post(); break;
      case mrNo: Table1->Cancel(); break;
      case mrCancel: Close();
    }
}
```

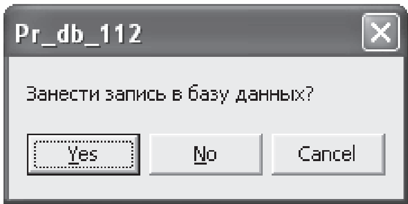


Рис. 10.21. Окно, определенное пользователем

4. Функция *TApplication->MessageBox*.

Основным недостатком ранее рассмотренных функций является отсутствие русификации диалоговых окон и невозможность указать текст заголовка окна. Эти недостатки устраняет функция *TApplication->MessageBox*.

Формат данной функции следующий:

MessageBox (Text, Caption, Flags),

где *Text* – текст сообщения, которое может превышать 355 символов, *Caption* – текст заголовка окна, *Flags* – множество флажков, определяющих вид и поведение диалогового окна.

Рассмотрим значения различных параметров данной функции.

В табл. 15 приведены флажки для кнопок и иконок диалогового окна.

Таблица 15

Флажки кнопок и пиктограмм в диалоговом окне

Флажок	Описание
Кнопки	
MB_ABORTRETRYIGNORE	Кнопки Abort (Стоп), Retry (Повтор) и Ignore (Пропустить)
MB_OK	Кнопка Ok
MB_OKCANCEL	Кнопки Ok и Cancel (Отмена)
MB_RETRYCANCEL	Кнопки Retry (Повтор) и Cancel (Отмена)
MB_YESNO	Кнопки Yes (Да) и No (Нет)
MB_YESNOCANCEL	Кнопки Yes (Да), No (Нет) и Cancel (Отмена)
Пиктограммы	
MB_ICONWARNING	Восклицательный знак (замечание, предупреждение)
MB_ICONINFORMATION	Буква "i" в круге (подтверждение)
MB_ICONQUESTION	Знак вопроса (ожидание ответа)
MB_ICONSTOP, MB_ICONERROR	Знак креста на красном круге (запрет, ошибка)

Данная функция также возвращает одно из значений – нажатий кнопок (табл. 16).

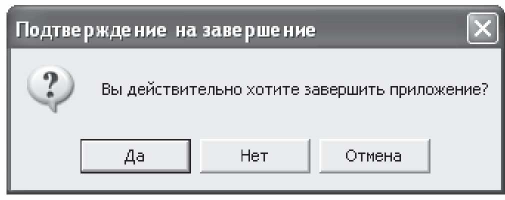
Таблица 16

Константы кнопок

Значение	Численное значение	Пояснение
IDABORT	3	Выбрана кнопка Abort (Стоп)
IDCANCEL	2	Выбрана кнопка Cancel (отмена) и нажата клавиша Esc
IDIGNORE	5	Выбрана кнопка Ignore (Пропустить)
IDNO	7	Выбрана кнопка No (Нет)
IDOK	1	Выбрана кнопка Ok
IDRETRY	4	Выбрана кнопка Retry (Повтор)
IDYES	6	Выбрана кнопка Yes (Да)

Рассмотрим пример, в котором запрашивается подтверждение на завершение работы приложения, а затем (при положительном ответе) выводится диалоговое окно информационного характера о завершении программы (рис. 10.22).

```
if (Application->MessageBox("Вы хотите завершить приложение?",
                            "Подтверждение на завершение",
              MB_YESNOCANCEL + MB_ICONQUESTION)==IDYES)
{
    Application->MessageBox("Работа закончена!",
   "Завершение приложения", MB_OK + MB_ICONINFORMATION);
    Close();
}
```



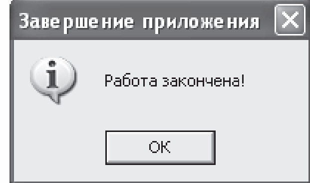


Рис. 10.22. Диалоговые окна подтверждения и информационного характера

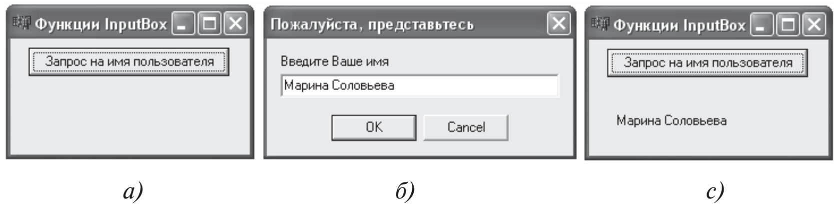
5. Функция *InputBox*.

Диалоговое окно, создаваемое с помощью данной функции, используется для ввода каких-либо данных. Функция возвращает вводимое значение, которое можно использовать затем в приложении для обработки (рис. 10.23*б*).

Формат функции:

InputBox (Caption, Prompt, Default),

где *Caption* – заголовок окна, *Prompt* – комментарий для ввода данных, *Default* – строка, которую функция возвращает в случае, если пользователь отказался от нажатия на кнопку <Ok>.



а) *б)* *с)*

Рис. 10.23. Поэтапная реализация приложения

Следующий фрагмент позволяет организовать диалоговое окно для ввода данных:

```
AnsiString Name= InputBox ("Пожалуйста, представьтесь",
                           "Введите Ваше имя", "Неизвестный");
Label1->Caption = Name;
```

6. Функция *InputQuery*.

В отличие от предыдущей функция *InputQuery* возвращает значение только в случае положительного ответа. Это значит, что пользователь всегда будет знать, положительный или отрицательный ответ был дан. Если при разработке приложения такой момент обязательно нужно проследить, то есть смысл выбрать функцию *InputQuery,* в противном случае можно воспользоваться предыдущей функцией *InputBox*.

Формат функции:

InputQuery (Caption, Prompt, &Value),

где *Caption* и *Prompt* характеризуют те же параметры, что и в функции *InputBox, Value* – строка редактирования текстового поля в диалоговом окне.

Например, следующий фрагмент программного кода позволяет при положительном ответе вывести приветствие с именем пользователя, а при отрицательном – приветствие незнакомца.

```
AnsiString Name=»Неизвестный»;
if (! InputQuery («Представьтесь», «Введите Ваше имя», Name))
    ShowMessageBox ();
else
    ShowMessageBox («Здравствуйте, » + Name + «!»);
```

Вопросы для самоконтроля

1. Чем отличается работа с полями типа *OLE* и *Memo* от работы с другими полями таблицы?
2. Как вводятся в таблицу значения полей типа OLE и *Memo*?
3. Что такое вычисляемое поле?
4. Каким образом создается *вычисляемое поле*? Описать последовательность действий.
5. С какой целью используется ниспадающий список в режиме просмотра таблицы?
6. Каким образом создается ниспадающий список?
7. Что такое маска ввода?
8. С какими компонентами формы связана маска ввода?
9. Какие шаблоны используются при создании маски ввода?
10. Каким образом можно передать данные с одной формы на другую (в случае просмотра данных из одной таблицы)?

11. Какие дополнительные компоненты системы могут быть использованы для просмотра данных и их поиска?
12. Как можно просмотреть данные из трех взаимосвязанных таблиц?
13. Какие возможные ситуации могут быть связаны с окном New Field? Пояснить, какие типы новых полей можно создавать.
14. Какие виды диалоговых окон можно использовать для диалога с пользователем?
15. Какие функции используются для вывода диалоговых окон?
16. Каким образом формируется функция вывода окна сообщения *MessageDlg* в зависимости от ситуации?

11. РЕАЛИЗАЦИЯ ОСНОВНЫХ ФУНКЦИЙ ПРИЛОЖЕНИЯ

11.1. Разработка пользовательского навигатора

Использование системного навигатора упрощает осуществление многих функций при работе с базами данных. Однако знание различных команд, способствующих разработке приложений, необходимо для создания более сложных функций.

Рассмотрим пример формы (рис. 11.1), которая позволяет:

~ осуществлять переход по записям через собственный навигатор данных таблицы;

~ выполнить через системный навигатор операции манипулирования данными;

~ поиск данных по фамилии и вычисление среднего арифметического всех оценок студента.

Необходимо заметить, что в качестве примера взята таблица СТУДЕНТ, являющаяся результирующей таблицей, то есть объединяющей информацию различных таблиц. Использование связей в данном примере не предусмотрено; основным моментом в данном примере является использование программного кода для обработки информации таблицы.

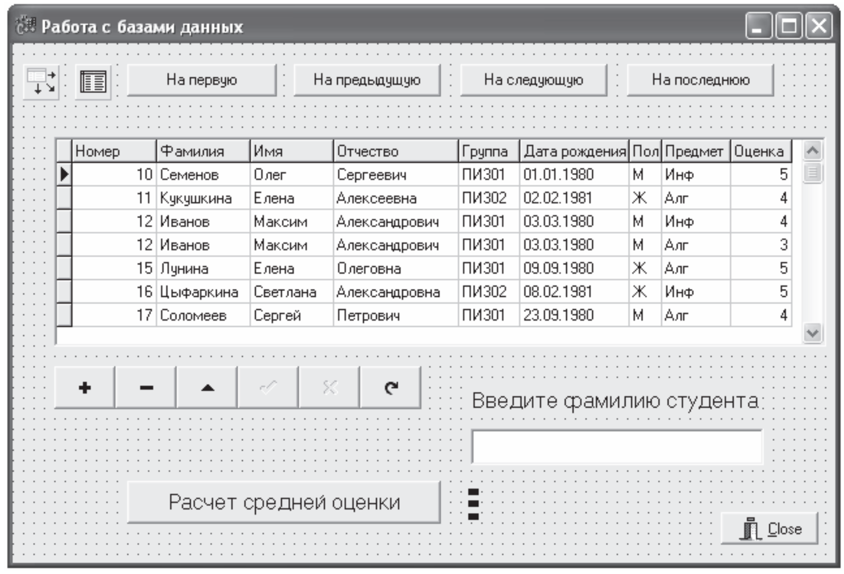


Рис. 11.1. Форма проекта по работе с БД «Студент»

Для реализации данной формы необходимо выполнить следующие действия:

1. Создать таблицу СТУДЕНТ с использованием *Database Desktop*.
2. На главной форме расположить компоненты *Table, DataSource, DBGrid, DBNavigator* и настроить их соответствующим образом.
3. Проверить работу проекта.
4. Активизировать компонент *DBNavigator1* и раскрыть «щелчком» групповое свойство *VisibleButtons*.
5. Для идентификаторов *nbFirst, nbPrior, nbNext, nbLast* установить значение *false* (в результате чего на форме видимы будут только шесть кнопок).
6. Добавить на форму четыре кнопки и задать им соответствующие названия: <*На первую*>, <*На предыдущую*>, <*На следующую*>, <*На последнюю*>. Данный навигатор представим как пользовательский.
7. Войти в программный код кнопки <*На первую*> и ввести следующий фрагмент программы:

```
	// если табличный указатель в текущий момент не находится
	// на первой записи
if (Table1->Bof! = true)
{
	// перевести табличный указатель на первую запись
Table1->First();
}
```

Метод *Bof (Begin Of File)* проверяет начало таблицы, а метод *Eof (End Of File)* – конец таблицы. Метод *First()* осуществляет переход табличного указателя на первую запись таблицы; *Next()* – на следующую запись; *Prior()* – на предыдущую запись; *Last()* – на последнюю запись.

8. Используя описанные выше методы, аналогичным способом оформить программные коды всех кнопок пользовательского навигатора.
9. Для установки русских названий столбцов необходимо использовать *Редактор полей*.
10. Проверить изменения в окне *Object TreeView;* в результате обращения к «Редактору полей» в окне появятся имена полей, которые можно использовать при обращениях к записям таблиц в программном коде (рис. 11.2).

Идентификаторы, представленные на рисунке и установленные по умолчанию, позволяют в дальнейшем разработчику обращаться к ним для обработки полей таблиц.

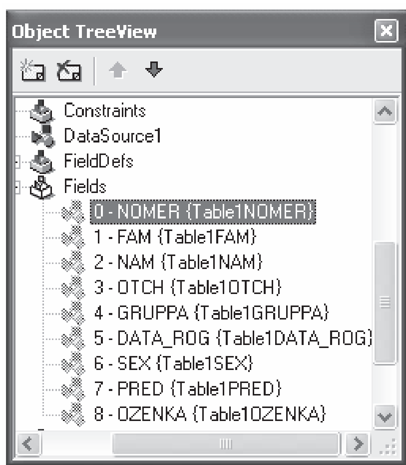


Рис. 11.2. Окно Object TreeView

11.2. Поиск данных в проекте

Необходимо заметить, что для поиска данных в приложениях, работающих на основе баз данных, используются специальные средства, например, язык запросов SQL. В данной работе используются средства системы Borland C++Builder и языка C++.

Для поиска данных используется два подхода:

~ просмотр всех записей для поиска данных согласно критерию;

~ поиск и остановка на конкретной записи, которая удовлетворяет условию.

Рассмотрим ***первый вариант.***

На рис. 11.1 приведена форма с кнопкой, которая позволяет вычислить среднее арифметическое всех оценок конкретного студента. При этом просматривается вся таблица (пока не конец файла).

Листинг 11.1. Программный код кнопки «Расчет средней оценки»

```
        // инициализация переменной, характеризующая среднее арифметическое
        // всех оценок
    float soz=0;
        // инициализация счетчика – количества оценок
    int n=0;
        // установка табличного указателя на первую запись таблицы
    Table1->First();
```

```
        // условие – пока не конец файла
    while (Table1->Eof!=true)
    {
        // условие – если значение поля текущей записи равно значению
        // текстового поля
        if (Table1FAM->Value==Edit1->Text)
        {
        // инкремент счетчика количества оценок
                n++;
                // увеличение суммы на значение новой оценки
                soz=soz+Table1OZENKA->Value;
        }
        // переход на следующую запись
        Table1->Next();
    }
        // условие, контролирующее более одной записи
    if (n>1)
    {
        soz=soz/n;
        Label2->Caption="Средняя оценка студента =
                                    "+FloatToStrF(soz, ffGeneral,3,1);
    }
        // условие, контролирующее одну запись
    if (n==1)
    {
        Label2->Caption="У студента одна оценка!";
    }
        // условие, проверяющее отсутствие записи в таблице
    if (n==0)
    {
        Label2->Caption="Такого студента нет!";
    }
        // переход на первую запись
    Table1->First();
```

Второй вариант

Существует несколько методик поиска записей, которые можно назвать *SetKey*, *FindKey*, *Lookup*.

1. Оператор *SetKey*.

Таблица предварительно должна быть индексирована по тому полю, по которому будет проводиться поиск. Затем таблица устанавливается в состояние поиска *dsSetKey* с помощью метода *SetKey*. В состоянии

dsSetKey набор данных воспринимает последующий оператор как задание ключа поиска. Поэтому после задания данного состояния устанавливается требуемое значение ключа поиска по интересующему полю. В заключение методом *GotoKey* курсор переводится на запись, в которой значение указанного поля равно ключу. Если таких записей несколько, то курсор переводится на первую из них. Если соответствующая запись не находится, то метод *GotoKey* возвращает *false*. Для полей типа строк лучше использовать не метод *GotoKey*, а метод *GotoNearest*. Этот метод перемещает курсор на первую запись, значение поля в которой максимально близко к ключу. То есть он сработает и тогда, когда совпадение не полное. Метод *GotoNearest* можно применять и к цифровым полям. В этом случае он переместит курсор на первую запись, значение поля в которой больше или равно заданному значению ключа.

Например, на форме в текстовое поле вводится фамилия; необходимо найти данную фамилию в таблице; в случае отсутствия фамилии в базе на экран выводится информационное сообщение.

```
Table1->IndexFieldNames = "Fam";
Table1->SetKey();
Table1->FieldByName("Fam")->AsString = EdFam->Text;
if (!Table1->GotoKey())
      ShowMessage ("Запись не найдена");
```

Первый оператор подключает индекс, созданный по полю *Fam*. Второй оператор переводит набор данных в состояние *dsSetKey*. Третьей командой задается ключ поиска, который равен считанному из текстового поля *EdFam* значению. При этом считываемое значение воспринимается системой как строковое (*AsString*). Четвертый оператор осуществляет переход к соответствующей записи или сообщает об отсутствии такой записи.

Поиск фамилии можно осуществить и следующим фрагментом программного кода:

```
Table1->IndexFieldNames = "Fam";
Table1->SetKey();
Table1->FieldByName("Fam")->AsString = EdFam->Text;
Table1->GotoNearest();
```

Даже если точно такой фамилии не найдется, курсор перейдет на наиболее похожую (совпадающую по первым символам).

2. Оператор *FindKey*.

Методика поиска *FindKey* еще богаче по своим возможностям. В этой методике таблица также должна быть проиндексирована по тем ключевым полям, по которым осуществляется поиск. Функция *FindKey* определена следующим образом:

```
bool __fastcall FindKey (const System::TVarRec *KeyValues,
                                        const int KeyValues_Size);
```

Параметр *KeyValues* представляет собой открытый массив: разделяемый запятыми список значений полей, по которым индексирован набор данных, в той последовательности, в которой они входят в индекс. При этом не обязательно перечислять все поля – достаточно перечислить первое или несколько первых. Параметр *KeyValues_Size* определяет индекс последнего поля в массиве, участвующего в поиске. Поскольку индексы начинаются с 0, то *KeyValues_Size* на единицу меньше количества полей, участвующих в поиске.

Вместо *FindKey* для полей строкового типа можно использовать аналогичный метод *FindNearest*, обеспечивающий переход к наиболее совпадающей строке, если полного совпадения не получено. Объявление и параметры этого метода те же, что и в методе *FindKey*. Метод *FindNearest* можно применять и к цифровым полям. В этом случае он переместит курсор на первую запись, значение полей в которой больше или равны заданным значениям ключей.

Метод *Lookup* определен следующим образом:

```
System::    Variant __fastcall Lookup (
            const System::AnsiString KeyFields,
            const System::Variant &KeyValues,
            const System::AnsiString ResultFields);
```

Например, если необходимо найти запись, относящуюся к сотруднику, фамилия которого указана в текстовом поле *EdFam*, и вывести в поле *EdNam* имя сотрудника, то эти операции можно осуществить следующим образом:

```
EdNam->Text = Table1->Lookup("Fam", EdFam->Text, "Nam");
```

Метод *Lookup* не изменяет положения курсора. Он только возвращает значения указанных полей. Они могут использоваться любым способом. В частности, они могут использоваться вместо параметра *KeyValues* в другом операторе *Lookup* или *Locate*. Это открывает широкие возможности формирования сложных запросов по нескольким таблицам.

Рассмотрим еще один пример, который позволяет осуществлять поиск данных при работе с двумя таблицами ПРЕПОДАВАТЕЛЬ и ПРЕДМЕТ (рис. 11.3).

Данный проект позволяет выбрать из комбинированного списка фамилию преподавателя, а затем получить список предметов, которые он преподает.

Трудность разработки данного проекта заключается в том, что при указании источника данных для комбинированного списка данный список автоматически не устанавливается, как обычно это наблюдается в специализированных системах (СУБД). Необходимо прибегнуть к некоторой хитрости для того, чтобы создать список преподавателей (из таблицы ПРЕПОДАВАТЕЛЬ), а затем привязать его ко второй таблице (ПРЕДМЕТ).

Рис. 11.3. Просмотр данных по фамилии

Прежде чем приступить к разработке данного приложения, необходимо описать некоторые моменты.

На рис. 11.4 расположено два набора компонентов для таблиц. Компонент *DBGrid* относится только ко второй таблице, компоненты *DBEdit* – к первой. По интерфейсу формы видно, что в нижней части формы расположены текстовые поля для того, чтобы проверить работу выбора элемента комбинированного списка.

Рис. 11.4. Форма приложения в режиме редактора

Для разработки приложения необходимо выполнить следующие действия:

1. Создать новый проект и установить соответствующие компоненты (рис.11.4).
2. Настроить компоненты *Table1, DataSource1* на таблицу ПРЕПОДАВАТЕЛЬ*, Table2, DataSource2, DBGrid* – на таблицу ПРЕДМЕТ.
3. «Перетащить» компоненты *DBEdit* из «Редактора полей» таблицы ПРЕПОДАВАТЕЛЬ.
4. Активизировать форму и войти в программный код события *OnCreate* во вкладке *Events*.
5. Ввести программный код, учитывая некоторые особенности собственного приложения:

Листинг 11.2. Добавление в комбинированный список значений

```
    // активизировать таблицу ПРЕПОДАВАТЕЛЬ
Table1->Active=true;
    // перейти на первую запись
Table1->First();
    // очистить список фамилий
ComboBox1->Clear();
    // пока не конец файла таблицы
while (!Table1->Eof)
{
      // добавить в комбинированный список строковое значение, взятое
      // из текущей записи поля Фамилия – Table1Famil
      ComboBox1->Items->Add(Table1Famil->AsString);
      // перейти на следующую запись
      Table1->Next();
}
    // установить активным первый элемент списка (индекс 0)
ComboBox1->ItemIndex=0;
    // перейти на первую запись
Table1->First();
    // активизировать вторую таблицу
Table2->Active=true;
```

Необходимо заметить, что введенный программный код позволяет ввести в любой список данные из конкретного поля.

6. Активизировать компонент *ComboBox* и вызвать программный код события *OnChange* на вкладке *Events*.
7. Ввести, учитывая некоторые особенности собственного приложения, следующий код:

Листинг 11.3. Установка связей между таблицами

```
// установить для дочерней таблицы поле связи из родительской
   таблицы
Table2->MasterFields="Kod_prep";
// установить для дочерней таблицы собственное поле связи
Table2->IndexFieldNames="Kod_pp";
// установить поле, по которому существует индекс для поиска
Table1->IndexFieldNames="Famil";
// найти согласно элементу списка нужное значение (см.§4.5)
Table1->FindNearest(&TVarRec(ComboBox1->Text),0);
// передать источник данных компоненту-просмотрщику
DBEdit1->DataSource=DataSource1;
// передать фокус, который позволит активизировать и обновить
   компонент
DBGrid1->SetFocus();
```

Таким образом, можно организовать просмотр данных из нескольких таблиц.

11.3. Фильтрация данных

Фильтрация – процесс отбора в оперативной памяти данных таблиц согласно определенному критерию.

Необходимо заметить, что сама таблица и его содержимое не изменяется; изменения происходят в оперативной памяти, то есть во временной таблице, куда загружается на временное пользование база данных. Предполагается, что с отфильтрованными записями можно производить какие-либо действия, а потом вновь возвращать базу данных к исходному состоянию. Такая ситуация складывается в случаях, когда не нужно изменять таблицу, не нужно создавать лишние файлы (возможно, огромного объема), а просто для удобства выбрать некоторые записи и поработать с ними некоторое время (просто попользоваться данными).

Следующие три рисунка (рис. 11.5–11.7) показывают определенное состояние таблицы СТУДЕНТ:

~ рис. 11.5 – отображение всех данных (разные группы различных специальностей);

~ рис. 11.6 – отображение данных по конкретной группе;

~ рис. 11.7 – отображение данных конкретной специальности.

Прежде чем рассмотреть программный код приложения, необходимо пояснить, какие компоненты формы расположены на ней.

Настройка таблицы на компоненты *Table*, *DataSource* и *DBGrid* не раз рассматривалась в различных параграфах. Поэтому рассмотрим интерфейс нижней части формы.

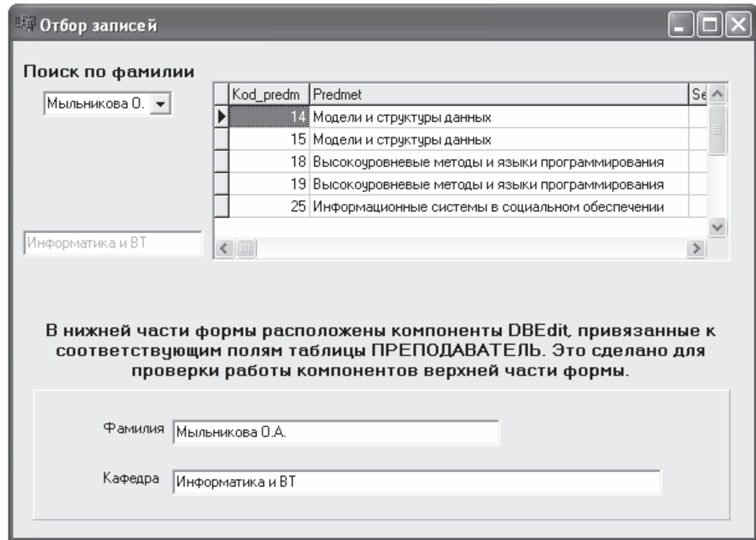


Рис. 11.5. Окно приложения «Фильтрация записей»

Необходимо заметить, что в нижней части формы расположены следующие компоненты, установленные с вкладки *Standard*:

~ *RadioGroup* (группа зависимых переключателей);
~ *ListBox* (список);
~ *Memo* (многострочное текстовое поле);
~ *ComboBox* (раскрывающийся или комбинированный список).

Компонент *Memo* является недоступным для редактирования, так как носит только ознакомительный характер для пунктов соседнего списка. Свойство *Lines* содержит список пунктов; свойство *Enabled* (доступность) находится в состоянии *false* (недоступен).

Свойство *Items* (также как и свойство *Lines* компонента *Memo*) компонента *ListBox* содержит список шифров специальностей.

Компонент *ComboBox* должен заполняться элементами при наступлении события выбора пункта шифра специальности в компоненте *ListBox*. Первоначально можно установить для комбинированного списка свойства недоступности. Это связано с тем, что пользователь случайно может выбирать группу, а ее, к сожалению, нет в списке, так как список формируется после выбора шифра специальности.

Очень часто при разработке приложений возникают ситуации, когда необходимо «заблокировать» действие, которое пользователь может

произвести. Это обычно связано с тем, что пользователь не знает последовательности наступления тех или иных событий, и может нарушить порядок действий. В связи с этим есть смысл делать недоступными те или иные компоненты, пока они не станут «нужными».

Фильтрация может производиться по группе, если последняя установлена, или по специальности, если та выбрана. В данной программе не предусмотрен анализ выбора группы или специальности, хотя при разработке «правильного» приложения такой момент необходимо учитывать обязательно.

Рассмотрим фрагмент программного кода события выбора специальности. Для этого активизируется компонент *LIstBox*, и на вкладке *Events* осуществляется переход в программный код события *OnClick*:

Листинг 11.4. Формирование списка

```
    // очистка списка, необходимая каждый раз при выборе специальности
ComboBox1->Clear();
    // если считанный по индексу пункт (как строковое значение) списка
    равен ...
if (ListBox1->Items->Strings[ListBox1->ItemIndex] == "351400")
{
    // добавить в список элемент как очередной пункт списка
ComboBox1->Items->Add("ПИ-101");
ComboBox1->Items->Add("ПИ-201");
ComboBox1->Items->Add("ПИ-202");
ComboBox1->Items->Add("ПИ-301");
ComboBox1->Items->Add("ПИ-302");
ComboBox1->Items->Add("ПИ-303");
}
    // в этом месте должны быть условия для заполнения списка другими
    // значениями групп
...
    // установка активного элемента списка (первый элемент всегда
    имеет
    // индекс 0)
ComboBox1->ItemIndex=0;
    // установка доступности списка (после выбора шифра специальности)
ComboBox1->Enabled=true;
```

Таким образом, после выбора шифра специальности (в обычном списке *ListBox*) в комбинированном списке появляется список групп согласно специальности.

Следующий шаг связан с описанием программного кода события выбора переключателя.

Необходимо заметить, что для фильтрации необходимо подключать индексы, созданные по полям, которые служат основой для критерия. Так, например, для фильтрации по группе необходимо создать индекс *Группа+Фамилия*, а для фильтрации по специальности – *Специальность+Фамилия*. Второе поле в обеих фильтрациях необходимо для простоты работы с фамилиями, то есть сначала сортировка происходит по группе (специальности), а затем – по фамилии.

Допустим, что существуют (созданные в Database Desktop) индексы *grupfam* и *spezfam*.

Рассмотрим фрагмент программного кода выбора переключателя. Для этого активизируется компонент *RadioGroup* и на вкладке *Events* осуществляется переход по событию *OnClick*:

Листинг 11.5. Обработка переключателей

```
    // активизация индекса grupfam
Table1->IndexName="grupfam";
    // если выбран первый переключатель (его индекс 0)
if (RadioGroup1->ItemIndex==0)
    // признак таблицы – не фильтрованная
Table1->Filtered=false;
else
{
    // если выбран третий переключатель (его индекс 2)
    if (RadioGroup1->ItemIndex==2)
    {
            // активизация индекса spezfam
            Table1->IndexName="spezfam";
            // формирование предложения-критерия
            Table1->Filter="Spez='"+
                ListBox1->Items->Strings[ListBox1->ItemIndex]+"'";
    }
    // если выбран второй переключатель (его индекс 1)
    else
    {
    // формирование предложения-критерия
            Table1->Filter="Gruppa='"+ComboBox1->Text+"'";
    }
    // признак таблицы – фильтрованная
Table1->Filtered=true;
}
```

Далее приведены интерфейсы фильтрации по группе и по специальности.

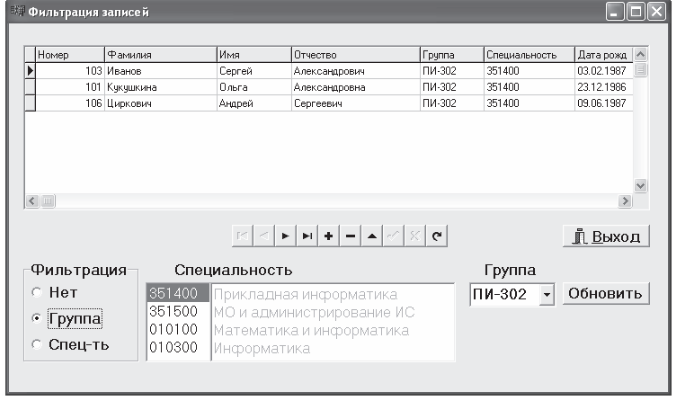


Рис. 11.6. Фильтрация по группе ПИ-302 специальности 351400

Фильтрация записей

Номер	Фамилия	Имя	Отчество	Группа	Специальность	Дата рожд
111	Ванин	Олег	Олегович	М-401	010100	23.05.1986
110	Киселева	Марина	Андреевна	М-101	010100	13.08.1989
107	Павлова	Мария	Петровна	М-201	010100	15.04.1988
109	Петринин	Максим	Александрович	М-501	010100	09.12.1984
114	Ромашина	Ольга	Константиновна	М-501	010100	30.01.1985
108	Семенова	Екатерина	Александровна	М-401	010100	30.01.1986

Выход

Фильтрация: Нет / Группа / Спец-ть

Специальность:
351400 Прикладная информатика
351500 МО и администрирование ИС
010100 Математика и информатика
010300 Информатика

Группа: М-101 Обновить

Рис. 11.7. Фильтрация по специальности «Математика»

Последним шагом в данном приложении является разработка кнопки «*Обновить*». Данная кнопка необходима для того, чтобы обновить данные в таблице после выбора специальности или группы. Поэтому в программном коде события нажатия кнопки необходимо сделать ссылку (или вызвать) на функцию выбора переключателей. Например, выбор переключателей осуществляется через функцию, описанную как

void __fastcall TForm1::RadioGroup1Click(TObject *Sender) {... }

Чтобы вызвать данную функцию для очередного выполнения, необходимо записать имя функции и ее параметры, то есть ***RadioGroup1Click(Sender).***

11.4. Добавление данных

Добавление записей в базу данных является одной из основных функций приложения, обрабатывающего информацию структурированных данных.

Системный анализ задачи предполагает различные варианты добавления данных:

1. Добавление новых студентов.
2. Добавление новых преподавателей.
3. Добавление новых дисциплин.
4. Добавление оценок студентов после каждой сессии.

Реализация данных функций позволяет проследить последовательность действий разработчика. Предполагается, что в приложении на главной форме существует меню, которое позволяет вызвать форму добавления, удаления и просмотра данных. Допустим, что форма добавления данных вызывается аналогичным способом и является лишь частью проекта. Из предложенных вариантов добавления данных первая пара функций реализуется так же одинаково, как и вторая пара функций.

Рассмотрим первую и третью функции добавления данных (рис. 11.8, прил. 2).

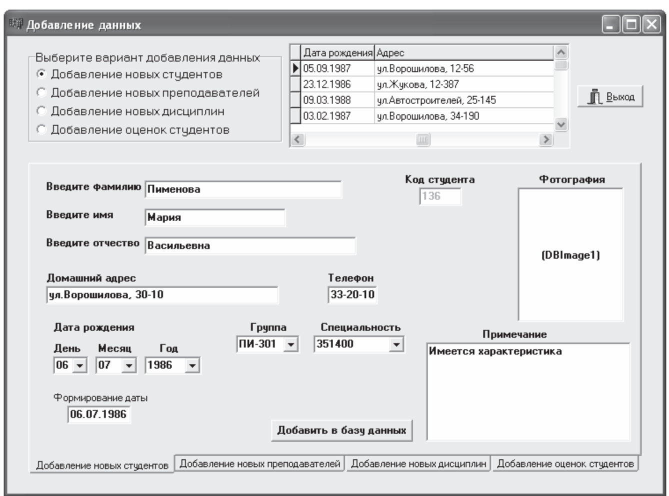


Рис. 11.8. Страница добавления новых студентов

Как видно из рис. 11.8, форма добавления данных содержит компонент *RadioGroup*, который позволяет активизировать одну из четырех вкладок (страниц) компонента *PageControl*, имеющий в своем составе любое количество страниц *TabSheet*. В верхней части формы находится компонент *DBGrid*, отражающий данные тех таблиц, с которыми пользователь в настоящее время работает. Поэтому необходимо программным путем организовать активизацию таблиц в определенные моменты времени.

На форме располагаются четыре набора компонентов для обработки таблиц базы данных (СТУДЕНТ, ОЦЕНКА, ПРЕДМЕТ, ПРЕПОДАВАТЕЛЬ). Необходимо заметить, что для первой функции добавления достаточно работать только с таблицей СТУДЕНТ, для второй функции – с таблицей ПРЕПОДАВАТЕЛЬ, для третьей функции – с таблицами ПРЕПОДАВАТЕЛЬ и ПРЕДМЕТ, для четвертой – с таблицами СТУДЕНТ, ПРЕДМЕТ и ОЦЕНКА.

Чтобы не усложнять обработку данных, необходимо заметить, что все пары компонентов настроены на соответствующие таблицы, но без связей между ними. Кроме того, достаточно сложно добавить фотографию в таблицу уже известным способом. Поэтому есть смысл фотографию добавить после того, как запись введена в таблицу, т. е. в режиме редактирования, а не добавления. Последний вариант наиболее прост и удобен.

I. Разработка первой функции – добавление нового студента

Особенность обработки:

~ при открытии страницы (выбор переключателя) должны быть установлены какие-либо параметры для работы данной вкладки. Так, например, код студента должен формироваться автоматически, увеличиваясь на 1 (при открытии страницы);

~ формирование даты должно осуществляться каждый раз после выбора дня, месяца и года;

~ кнопка «*Добавить в базу данных*» должна добавлять запись в таблицу СТУДЕНТ и автоматически обновлять компонент *DBGrid*.

Ход работы:

1. На новую форму добавить компонент *PageControl*.
2. Выполнить команду *New Page* из контекстного меню компонента *PageControl*.
3. Активизировать первую страницу «щелчком» мыши. При этом проверить, чтобы в «Инспекторе объектов» была указана именно первая страница, например, *TabSheet1*.
4. Через свойство *Caption* задать странице название согласно рис. 11.8. Аналогичным способом задать необходимое число страниц с заголовками.

5. Установить на области страницы необходимые компоненты *Label, Edit, ComboBox, Button, Memo, DBImage*, *EditMask* (для телефона).

6. Установить для каждого компонента страницы свойства согласно интерфейсу.

Свойство *TabOrder* позволяет установить для компонента порядок передачи фокуса. Индекс 0 должен быть у текстового поля фамилии; затем по порядку устанавливается индекс для имени, отчества, домашнего адреса и т. п.

Задать фиксированные списки для компонентов *ComboBox* через свойство *Items* (день: 1-31; месяц: 1-12; год: 1970-2005; группа; специальность).

Для компонента *EditMask* описать свойство *MaskEdit*.

7. Активизировать компонент *RadioGroup* и описать программный код события *OnClick*:

Необходимо заметить, что в программном коде описывается поиск наибольшего значения среди кодов студентов, и найденное число увеличивается на 1. Таким образом, автоматически определяется код для нового студента, что очень важно, так как это поле – *Nomer* – не допускает повторяющихся значений. По нему существует первичный ключ (индекс).

Листинг 11.6. Ввод неповторяющихся значений в ключевое поле

```
    // если выбран первый переключатель
if (RadioGroup1->ItemIndex==0)
{
    // задать источник данных для компонента DBGrid из первой
    // таблицы – СТУДЕНТ
    DBGrid1->DataSource=DataSource1;
    // инициализация максимума среди значений поля Nomer
    int max=0;
    // активизация и видимость первой страницы (индекс 0)
    PageControl1->ActivePageIndex=0;
    PageControl1->Visible=true;
    // установка табличного курсора на первую запись
    Table1->First();
    // пока не конец файла таблицы
    while (!Table1->Eof)
    {
        // если max < считанного значения текущей записи поля Nomer,
        if (max < Table1->FieldByName("Nomer")->AsInteger)
        // то max=считанному значению текущей записи поля Nomer
            max = Table1->FieldByName("Nomer")->AsInteger;
```

```
        // переход на следующую запись
      Table1->Next();
    }
    // получение нового значения, равное +1
    Edit1->Text=max+1;
    // переход на первую запись
    Table1->First();
    // передача фокуса текстовому полю фамилии
    Edit2->SetFocus();
}
```

8. Активизировать компонент *ComboBox*, содержащий дни, и войти в программный код события *OnChange*.
9. Ввести следующий код:

Листинг 11.7. Обработка комбинированного списка

```
void __fastcall TForm1::ComboBox3Change(TObject *Sender)
{
    // объявление переменных
AnsiString str01, str02, str03, str;
    // считывание значения дня
str01=ComboBox3->Text;
    // считывание значения месяца
str02=ComboBox4->Text;
    // считывание значения года
str03=ComboBox5->Text;
    // конкатенация всех трех переменных
str=str01+"."+str02+"."+str03;
    // присвоение текстовому полю результирующего значения
Edit5->Text=str;
}
```

10. Активизировать компонент *ComboBox*, содержащий месяцы, и в программном коде события *OnChange*, ввести следующую команду:

ComboBox3Change(Sender);

11. Аналогичным образом ввести эту команду и для события *OnChange* компонента *ComboBox*, описывающего года.
12. Сохранить проект и проверить работу комбинированных списков.
13. Активизировать кнопку добавления и ввести программный код:

Листинг 11.8. Добавление данных в таблицы

```
    // добавить новую запись в таблицу СТУДЕНТ, при этом на ней
    // устанавливает табличный курсор (особенности работы
    // с базами данных)
Table1->Insert();
    // добавить в текущую данные из различных полей, преобразовав
    //в целое число значение для поля Nomer, в дату значение для поля
    // Data_rog
Table1->FieldByName("Nomer")->AsInteger=StrToInt(Edit1->Text);
Table1->FieldByName("Fam")->AsString = Edit2->Text;
Table1->FieldByName("Nam")->AsString = Edit3->Text;
Table1->FieldByName("Otch")->AsString = Edit4->Text;
Table1->FieldByName("Data_rog")->AsDateTime =
StrToDate(Edit5->Text);
Table1->FieldByName("Adres")->AsString = Edit6->Text;
Table1->FieldByName("Phone")->AsString = MaskEdit1->Text;
Table1->FieldByName("Gruppa")->AsString = ComboBox1->Text;
Table1->FieldByName("Spez")->AsString = ComboBox2->Text;
    // сохранить все изменения в таблице
Table1->Post();
```

14. Сохранить проект и проверить работу первого переключателя.

II. Разработка третьей функции – добавление новой дисциплины.

Особенность обработки:

~ при открытии страницы (выбор переключателя) должны быть установлены какие-либо параметры для работы данной вкладки. Так, например, код новой специальности должен формироваться автоматически, увеличиваясь на 1 (при открытии страницы); а фамилии преподавателей должны быть загружены в комбинированный список из таблицы ПРЕПОДАВАТЕЛЬ;

~ при открытии третьей страницы видимыми являются только данные для первого семестра обучения; если пользователь выбирает второй переключатель, то видимой становится и вторая область ввода; также дело обстоит и с третьей областью ввода;

~ после выбора преподавателя необходимо обратиться к таблице ПРЕПОДАВАТЕЛЬ и считать код преподавателя (для ввода кода в таблицу ПРЕДМЕТ; код преподавателя – внешний ключ);

~ после выбора семестра в первом комбинированном списке автоматически должны появляться последующие номера семестров во втором и третьем семестрах;

~ кнопка «*Добавить в базу данных*» должна добавлять запись в таблицу ПРЕДМЕТ и автоматически обновлять компонент *DBGrid*.

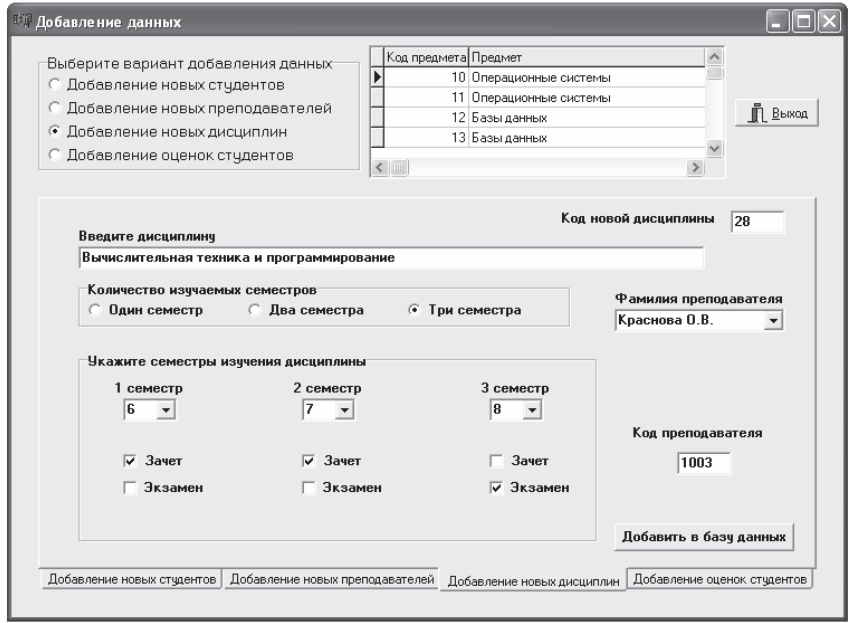


Рис. 11.9. Страница добавления новых дисциплин

Ход работы:

1. Активизировать третью страницу «щелчком» мыши. При этом проверить, чтобы в «Инспекторе объектов» указана именно первая страница, например, *TabSheet3*.

2. Установить на странице, согласно интерфейсу, необходимые компоненты: *Label*, *Edit* (код дисциплины и ее ввод, отображение кода преподавателя), *RadioGroup* (количество семестров), *ComboBox* (номера семестров и список фамилий преподавателей), *Button* и *CheckBox* (флажки для зачетов и экзаменов).

3. Установить для некоторых компонентов необходимые свойства: для комбинированных списков семестров числа 1–9, для переключателей активным первый элемент (индекс 0), заголовки меток.

4. Активизировать группу переключателей в верхней части формы (варианты добавления) и вновь войти в программный код события *OnClick*.

5. Добавить в программный код обработку третьего переключателя:

Листинг 11.9. Обработка переключателей

```
      // если активизирован третий переключатель
if (RadioGroup1->ItemIndex==2)
{
      // задать источник данных для компонента DBGrid из третьей
 таблицы – ПРЕДМЕТ
      DBGrid1->DataSource=DataSource3;
       // инициализация максимума среди значений поля Kod_predm
      int maxx=0;
       // активизация и видимость третьей страницы (индекс 2)
      PageControl1->ActivePageIndex=2;
      PageControl1->Visible=true;
      // установка табличного курсора на первую запись
      Table3->First();
      // пока не конец файла таблицы
      while (!Table3->Eof)
      {
               // если maxx < считанного значения текущей записи поля
               //  Kod_predm,
               if (maxx<Table3->FieldByName("Kod_predm")->AsInteger)
               // то maxx=считанному значению текущей записи поля
               // Kod_predm
                 maxx=Table3->FieldByName("Kod_predm")->AsInteger;
               // переход на следующую запись таблицы
               Table3->Next();
      }
      // получение нового значения, равного +1
      Edit9->Text=IntToStr(maxx+1);
      // переход на первую запись таблицы
      Table3->First();
      // передача фокуса текстовому полю ввода дисциплины
      Edit8->SetFocus();
      // данный фрагмент программы заполняет данными список
      // преподавателей
      // переход на первую запись таблицы
      Table4->First();
      // пока не конец файла таблицы
      while (!Table4->Eof)
      {
               // добавление в список значения из поля фамилии Table1Famil
               ComboBox9->Items->Add(Table4Famil->AsString);
```

```
            // переход на следующую запись
            Table4->Next();
    }
    // установка первого элемента
    ComboBox9->ItemIndex=0;
    // переход на первую запись
    Table4->First();
}
```

6. Активизировать комбинированный список преподавателей и перейти в программный код события *OnChange*:

<u>*Листинг 11.10. Поиск данных*</u>

```
   // объявление переменной целого типа
int nm;
    // поиск указанной в комбинированном списке фамилии среди
    // значений поля Famil, а затем считывание значения поля Kod_predm
nm = Table4->Lookup("Famil",ComboBox9->Text,"Kod_prep");
   // передача найденного кода после преобразования текстовому полю
Edit10->Text = IntToStr(nm);
```

7. Установить для компонентов, характеризующих второй и третий семестры, свойство *Visible* в состояние *false*.

8. Активизировать компонент *RadioGroup* и в программном коде события *OnClick* ввести следующий код:

```
// если активизирован первый переключатель (1 семестр), то первую
//  область (1 семестр) сделать видимой, а остальные – невидимыми
if (RadioGroup2->ItemIndex==0)
{
    Label15->Visible=true;
    ComboBox6->Visible=true;
    CheckBox1->Visible=true;
    CheckBox2->Visible=true;
    Label16->Visible=false;
    ComboBox7->Visible=false;
    CheckBox3->Visible=false;
    CheckBox4->Visible=false;
    Label17->Visible=false;
    ComboBox8->Visible=false;
    CheckBox5->Visible=false;
    CheckBox6->Visible=false;
}
```

// аналогичным образом обработать второй и третий переключатели
...

9. Активизировать комбинированный список 1-го семестра и в программном коде события *OnChange* ввести следующий код:

```
        // объявление переменной целого типа
int p;
        // считывание номера для первого семестра
p=StrToInt(ComboBox6->Text);
        // установка последующих семестров
ComboBox7->Text=IntToStr(p+1);
ComboBox8->Text=IntToStr(p+2);
```

Этот программный код необходим для того, чтобы после установки первого семестра автоматически установить последующие семестры, например, если первый – 5 семестр, то последующие – 6 и 7.

10. Активизировать кнопку и в программном коде события *OnClick* ввести следующий код:

Листинг 11.11. Обработка кнопок

```
        // если список первого семестра видим
if (ComboBox6->Visible==true)
{
        Table3->Insert();
        Table3->FieldByName("Kod_predm")->AsInteger=
        StrToInt(Edit9->Text);
        Table3->FieldByName("Predmet")->AsString = Edit8->Text;
        Table3->FieldByName("Semestr")->AsInteger =
        StrToInt(ComboBox6->Text);
        Table3->FieldByName("Kod_pp")->AsInteger =
        StrToInt(Edit10->Text);
        if (CheckBox1->Checked==true)
                Table3->FieldByName("Zachet")->AsString = "з";
        if (CheckBox2->Checked==true)
                Table3->FieldByName("Ekzamen")->AsString = "э";
        Table3->Post();
}
        // если список второго семестра видим
if (ComboBox7->Visible==true)
{
        Edit9->Text=IntToStr(StrToInt(Edit9->Text)+1);
        Table3->Insert();
```

```
        Table3->FieldByName("Kod_predm")->AsInteger=
        StrToInt(Edit9->Text);
        Table3->FieldByName("Predmet")->AsString = Edit8->Text;
        Table3->FieldByName("Semestr")->AsInteger =
        StrToInt(ComboBox7->Text);
        Table3->FieldByName("Kod_pp")->AsInteger =
        StrToInt(Edit10->Text);
        if (CheckBox3->Checked==true)
                Table3->FieldByName("Zachet")->AsString = "з";
        if (CheckBox4->Checked==true)
                Table3->FieldByName("Ekzamen")->AsString = "э";
        Table3->Post();
    }
        // если список третьего семестра видим
    if (ComboBox8->Visible==true)
    {
        Edit9->Text=IntToStr(StrToInt(Edit9->Text)+1);
        Table3->Insert();
        Table3->FieldByName("Kod_predm")->AsInteger=
        StrToInt(Edit9->Text);
        Table3->FieldByName("Predmet")->AsString = Edit8->Text;
        Table3->FieldByName("Semestr")->AsInteger =
        StrToInt(ComboBox8->Text);
        Table3->FieldByName("Kod_pp")->AsInteger =
        StrToInt(Edit10->Text);
        if (CheckBox5->Checked==true)
                Table3->FieldByName("Zachet")->AsString = "з";
        if (CheckBox6->Checked==true)
                Table3->FieldByName("Ekzamen")->AsString = "э";
        Table3->Post();
    }
```

Необходимо заметить, что сложность в последнем программном коде заключалась в том, что даже в невидимом состоянии комбинированные списки семестров продолжали иметь значения. Поэтому в качестве критерия (условия) проверка на отсутствие значений была бы неправильной. В связи с этим есть смысл в качестве критерия использовать видимость и невидимость компонентов.

Из программного кода видно, что он представлен тремя блоками, работающими на три переключателя. Добавление новой записи и ввод данных в конкретные поля не представляет проблем. Особенность заключается в том, что должен автоматически изменяться (увеличиваться)

код дисциплины, поэтому в начале каждого блока система обновляет код предмета в поле *Edit9*.

11.5. Удаление данных

Удаление данных также является одной из основных функций приложения. Удаление также предполагает несколько вариантов:

~ удаление преподавателя (из таблицы ПРЕПОДАВАТЕЛЬ с изменением (а не удалением) кода преподавателя в таблице ПРЕДМЕТ);

~ удаление студента в случае отчисления (из таблицы СТУДЕНТ с одновременным удалением оценок из таблицы ОЦЕНКА);

~ удаление предметов (из таблицы ПРЕДМЕТ);

~ перемещение из текущей базы группы студентов в таблицу-архив.

Рассмотрим пример удаления студента (см. прил. 3) с преждевременным поиском его в базе данных. Необходимо заметить, что данные удаляются из обеих взаимосвязанных таблиц СТУДЕНТ и ОЦЕНКА (рис. 11.10).

Особенности обработки:

~ при активизации страницы «*Удаление студента*» комбинированный список «*Выберите группу*» должен быть заполнен элементами фиксированного списка;

~ после выбора группы должен заполниться комбинированный список студентов из выбранной группы;

~ после выбора конкретного студента информация о нем должна появляться в нижней части страницы;

~ после нажатия кнопки текущая запись удаляется из таблицы СТУДЕНТ и связанной таблицы ОЦЕНКА.

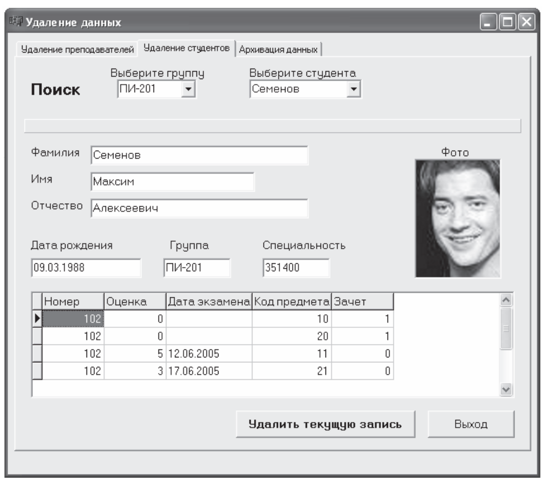


Рис. 11.10. Форма удаления данных

Ход работы:

1. Установить на форме компонент *PageControl* и через контекстное меню добавить новые страницы *TabSheet*.
2. Задать заголовки всем страницам.
3. Активизировать вторую страницу и установить на странице четыре набора компонентов для обработки таблиц (*Table, DataSource*).
4. Настроить каждый набор на соответствующую таблицу (пока без установки связей между ними).
5. Установить на странице необходимые компоненты: *Label*, *DBEdit* и *DBImage* («перетащить» из Редактора Полей таблицы СТУДЕНТ), *DBGrid* (настроить на вторую таблицу ОЦЕНКА), *Button*.
6. Активизировать форму и войти в программный код создания формы *OnCreate*.
7. Ввести команды активизации таблиц:

```
Table1->Activate = true;
Table2->Activate = true;
Table3->Activate = true;
Table4->Activate = true;
```

8. Для события *OnDestroy* формы описать обратные действия – дезактивировать таблицы.
9. Активизировать компонент *Table2* (таблицу ОЦЕНКА).
10. Установить следующие свойства:

```
    // в качестве родительского источника – таблица СТУДЕНТ
MasterSource = DataSource1
    // в качестве индекса родительской таблицы СТУДЕНТ – поле Nomer
MasterFields = Nomer
    // в качестве индекса дочерней таблицы ОЦЕНКА для связи
    // с таблицей СТУДЕНТ
IndexName = NmrO
```

11. Сохранить проект и проверить работу компонентов *DBGrid* и *DBEdit*.
12. Вернуться в рабочий режим проекта.
13. Активизировать комбинированный список групп и установить свойство *Items* как список групп.
14. Войти в программный код события *OnChange* и ввести команды по выбору формированию списка студентов выбранной группы:

```
    // объявление переменной s
String s;
    // очищение списка студентов
ComboBox2->Clear();
    // переход на первую запись иаблицы
```

```
Table1->First();
      // пока не конец файла таблицы
while (!Table1->Eof)
{
                // если значение поля Grupra текущей записи равно
                // выбранному элементу списка
    if (Table1->FieldByName("Gruppa")->AsString==ComboBox1->Text)
    {
                // присвоение переменной значения из поля Fam текущей
                //записи
      s= Table1->FieldByName("Fam")->AsString;
                // добавление в список студентов найденного значени
                // (студента)
      ComboBox2->Items->Add(s);
    }
    Table1->Next();
}
Table1->First();
```

15. Активизировать список студентов и в программном коде события *OnChange* ввести следующие команды:

```
    // задать в качестве индекса родительской таблицы поле Nomer
    // (для связи)
Table2->MasterFields="Nomer";
    // задать в качестве индекса дочерней таблицы поле Nomer (для связи)
Table2->IndexFieldNames="Nomer";
    // подключить индекс по полю Fam для дальнейшего поиска данных
    // по фамилии
Table1->IndexFieldNames="Fam";
    // нахождение по фамилии нужного значения (остановка
    //на конкретной записи)
Table1->FindNearest(&TVarRec(ComboBox2->Text),0);
    // передача фокуса компоненту DBGrid с одновременным обновлением
DBGrid1->SetFocus();
```

16. Сохранить проект и проверить работу комбинированных списков.
17. Вернуться в рабочий режим.
18. Активизировать кнопку и войти в программный код данного компонента.
19. Ввести следующие команды:

```
    // удаление текущей записи (на чем стоит табличный курсор)
    // с одновременным переходом на следующую запись
```

```
Table1->Delete();
        // при необходимости дезактивировать таблицы
Table1->Active=false;
Table2->Active=false;
```

Предложенная последовательность действия позволяет корректно и правильно осуществить операцию удаления.

11.6. Программирование базы данных

Помимо рассмотренных ранее методов модификации записей *Table* имеется ряд методов, позволяющих модифицировать таблицы и индексы.

1. Метод *CreateTable*.

Метод создает новую таблицу, исходя из установок компонента *Table*, содержащихся в свойствах *Fields* или *FieldDefs*. Если таблица с именем, указанным в свойстве *TableName* уже имеется, она будет переписана. Применение этого метода позволяет, например, взять структуру существующей таблицы, как-то изменить ее, затем изменить свойство *TableName*, на имя новой таблицы и создать эту таблицу.

2. Метод *DeleteTable*.

Метод уничтожает существующую таблицу, которая задана свойствами *DatbaseName* и *TableName*. Таблицу надо предварительно закрыть.

3. Метод *RenameTable(s)*

Метод переименовывает существующую таблицу, присваивая ей новое имя, содержащееся в *s*. Одновременно переименовываются все сопутствующие таблице файлы.

4. Метод *DeleteIndex(s)*

Удаляет вторичный индекс с именем *s* из таблицы.

Следующий фрагмент программного кода позволяет создать таблицу с учетом того, что на форме существуют необходимые компоненты, которые пока еще не настроены ни на какую таблицу.

Листинг 11.12. Добавление данных (второй способ)

```
        // дезактивация таблицы
Table1->Activate = false;
        // задание имени новой таблицы
Table1->TableName = “student.db”;
        // если такой таблицы в текущем каталоге не существует,
        значит, ее можно создать
if (!Table->Exists)
{
                // установить тип таблицы через константу системы
```

```
        Table1->TableType = ttParadox;
                // очистить список полей
        Table1->FieldDefs->Clear();
                // создать указатель на объект описания поля
                // добавить в таблицу новое поле, определив его название,
                // тип и запрет на нулевые значения в этом поле
        TFieldDef *pNewDef = Table1->FieldDefs->AddFieldDef();
        bNewdef->Name = “Nomer”;
        bNewdef->DataType = ftInteger;
        bNewdef->Required = true;
                // добавить в таблицу следующее поле
        pNewDef = Table1->FieldDefs->AddFieldDef();
        bNewdef->Name = “Fam”;
        bNewdef->DataType = ftString;
        bNewdef->Size = 20;
        bNewdef->Required = true;
                // добавить в таблицу следующее поле
        pNewDef = Table1->FieldDefs->AddFieldDef();
        bNewdef->Name = “Data_rog”;
        bNewdef->DataType = ftDate;
                // добавить в таблицу следующее поле
        pNewDef = Table1->FieldDefs->AddFieldDef();
        bNewdef->Name = “Family”;
        bNewdef->DataType = ftBoolean;
                // очистить указатель
        Table1->IndexDefs->Clear();
                // добавить для первого поля первичный индекс
        Table1->IndexDefs->Add(“”, “Nomer”,
        TIndexOptions() < ixPrimary < ixUnique);
                // создать таблицу на диске
        Table1->CreateTable();
                // открыть таблицу для работы
        Table1->Open();
                // вставить новую запись и заполнить ее записью
        Table1->Insert();
        Table1->FieldByName(“Nomer”)->AsInteger = 100;
        Table1->FieldByName(“Fam”)->AsString = “Иванов”;
        Table1->FieldByName(“Data_rog”)->AsDate = ‘02/09/1986’;
        Table1->FieldByName(“Family”)->AsBoolean = false;
                // сохранить введенные изменения
        Table1->Post();
}
```

Для работы с полями типа *Memo* необходимо воспользоваться методом Modified (например, *Form1->Memo1->Modified = false;*), чтобы разрешить пользователю изменять содержимое примечания.

Обработка поля типа *DBImage* также происходит через использование булевого компонента. Необходимо, чтобы компонент *DBImage = false*.

Вопросы для самоконтроля

1. Какие операторы используются для перехода с одной записи на другую? Какое отношение к переходу имеет проверка начала или конца файла?
2. Какие способы поиска в таблице существуют? Перечислите их и охарактеризуйте.
3. В чем особенность поиска данных метода *Lookup*?
4. Каким образом настраивается комбинированный список и компонент *Grid* для просмотра данных? Необходимо заметить, что выборка должна осуществляться после выбора элемента в ниспадающем списке.
5. Какое событие описывается выбором элемента в комбинированном списке?
6. Что такое фильтрация данных?
7. Каким образом осуществляется фильтрация данных программным путем?
8. Какое отношение фильтрация имеет к поиску данных?
9. Как осуществляется добавление данных программным путем?
10. Какое основное условие необходимо соблюдать при вводе данных в ключевые поля?
11. Как осуществляется удаление данных программным путем?
12. Каким образом возможна работа с несколькими таблицами одновременно? Каково условие их совместного существования?

12. РАЗРАБОТКА ЗАПРОСОВ К БАЗЕ ДАННЫХ

12.1. Введение в язык SQL

Язык SQL (*Structured Query Language* – язык структурированных запросов) был создан в конце 70-х годов и получил через некоторое время широкое распространение. Он позволяет формировать весьма сложные запросы к базам данных. *Запрос* — это вопрос к базе данных, возвращающий запись или множество записей, удовлетворяющих вопросу.

К сожалению, SQL в настоящее время недостаточно стандартизован. Существует стандарт SQL ANSI, но существует и множество диалектов, с которыми работают различные системы. Например, Sybase SQL Server и Microsoft SQL используют синтаксис, существенно отличающийся от стандарта ANSI. InterBase, Oracle и многие другие серверы в основном придерживаются стандарта ANSI, но каждый разработчик вносит в него и свои усовершенствования. Дальнейшее изложение будет основываться на диалекте, принятом в локальном сервере БД InterBase.

C++Builder позволяет приложению при помощи запросов SQL использовать данные:

~ таблиц Paradox и dBase – используется синтаксис локального SQL;

~ локального сервера InterBase – полностью поддерживается соответствующий синтаксис;

~ удаленных серверов SQL через драйверы SQL Links.

Общие правила синтаксиса SQL очень просты. Язык SQL не чувствителен к регистру. Если используется программа из нескольких операторов SQL, то в конце каждого оператора ставится точка с запятой «;». Впрочем, если вы используете всего один оператор, то точка с запятой в конце необязательна. Комментарий может писаться и в стиле C: /*<комментарий>*/, а в некоторых системах и в стиле Pascal: {<комментарий>}.

12.2. Оператор выбора SELECT

Оператор SELECT необходим для выборки данных из одной или нескольких таблиц.

Рассмотрим формат данной команды:

1 SELECT список_имен_полей
2 FROM список_имен_таблиц
3 WHERE связь_между_таблицами | условие_отбора
4 ORDER BY список_имен_полей
5 GROUP BY список_имен_полей;

Представленный формат команды SELECT для одной таблицы можно перевести так:

1 ВЫБРАТЬ *поле1, поле2, поле3, ...*
2 ИЗ *таблицы1*
3 ПРИ УСЛОВИИ, ЧТО *поле1=значение1* [AND | OR *поле2=значение2*] ...
4 [ОТСОРТИРОВАВ ПО *полю3* [ASC | DESC]]
5 [СГРУППИРОВАВ ПО *полю2*]

Необходимо заметить, что фрагменты 3, 4 и 5, оформленные в квадратных скобках, не являются обязательными. Если в запросе не задается условие и при выводе не указывается какое-либо выражение, то система выводит все записи. В случае, если необходимо использовать для вывода данных все поля таблицы, то достаточно использовать символ *.

В опции WHERE используются следующие операции отношения:

=	Равно
>	Больше
>=	Больше или равно
<	Меньше
<=	Меньше или равно
!=	Не равно
Like	Наличие заданной последовательности символов
Between ... and	Диапазон значений
In	Соответствие элементу множества

Оператор SELECT позволяет возвращать не только множество значений полей, но и некоторые совокупные (агрегированные) характеристики, подсчитанные по всем или по указанным записям таблицы.

К таким функциям относятся:

~ *count (условие)* – подсчет количества записей в таблице, удовлетворяющих заданным условиям;

~ *min (поле)* – возвращение минимального значения среди значений указанного поля согласно условию;

~ *max (поле)* – возвращение максимального значения среди значений указанного поля согласно условию;

~ *avg (поле)* – возвращение среднего арифметического значения среди значений указанного поля согласно условию;

~ *sum (поле)* – возвращение суммы значений среди значений указанного поля согласно условию.

При использовании суммарных характеристик надо учитывать, что в списке возвращаемых значений после ключевого слова SELECT могут

фигурировать или поля (в том числе и вычисляемые), или совокупные характеристики, но не могут фигурировать и те, и другие (без указания на группирование данных). Другими словами, оператор SELECT может возвращать или множество значений, или суммарные характеристики, но не может возвращать их совокупность, так как это несовместимо.

Смешение в одном операторе полей и совокупных характеристик возможно, если использовать группировку записей, задаваемую ключевыми словами GROUP BY.

При группировании записей с помощью GROUP BY можно вводить условия отбора записей с помощью ключевого слова HAVING.

Необходимо заметить, что не все базы данных поддерживают ключевое слово HAVING. Если с БД InterBase проблем с данной опцией нет, то БД Paradox не работает с этим словом.

Иногда возникают ситуации, когда использование запроса стандартного вида не дает нужных результатов. Для этого нужно воспользоваться так называемыми *вложенными запросами*. Это значит, что в качестве источника данных (условий) используется результат запроса.

Формат вложенного запроса может выглядеть так:

~ подзапрос как конкретное значение:

SELECT … FROM … WHERE поле знак_отношения (SELECT …)

~ подзапрос как список конкретных значений:

SELECT … FROM … WHERE поле IN (SELECT …)

При работе в условии WHERE с множеством записей можно использовать ключевые слова: *ANY* и *ALL*. *ALL* означает, что условие выполняется для всех записей, а *ANY* – хотя бы для одной записи.

До сих пор речь шла о запросах из одной таблицы.

Рассмотрим формат запроса для вывода информации *из двух и более взаимосвязанных таблиц:*

1 ВЫБРАТЬ *поле1, поле2, поле3, …*
2 ИЗ *таблицы1, таблицы2, …*
3 ПРИ УСЛОВИИ ЧТО
  (*таблица1.поле1=таблица2.поле2 [, …]*)
  AND
  (*таблица1.поле1=значение1* [AND\OR *таблица2.поле2=значение2*] …)
4 [ОТСОРТИРОВАВ ПО *полю3* [ASC | DESC]]
5 [СГРУППИРОВАВ ПО *полю2*]

Необходимо заметить, что при связи таблиц опция WHERE начинает выполнять и вторую функцию (помимо условия) – установка связи между таблицами по конкретным полям. Поля, как известно, должны

быть одного и того же типа, длины, а также иметь одинаковые идентификаторы для простоты работы.

Для работы с таблицами иногда используют так называемые псевдонимы, которые пользователь сам назначает. Однако эти псевдонимы обязательно необходимо указать вместе с таблицами. Это удобно использовать в случае, если названия таблиц слишком длинные, а также, если возникают запросы, которые без псевдонима не решишь (см. далее пример 21).

Рассмотрим несколько *примеров* запросов.

1. Выбрать из таблицы СТУДЕНТ всех студентов по всем полям.

SELECT *
FROM student

2. Выбрать из таблицы СТУДЕНТ всех студентов; при этом вывести информацию о фамилии, имени и группе.

SELECT fam, nam, gruppa
FROM student

3. Вывести всех преподавателей кафедры Информатики и ВТ; достаточно вывести фамилию, имя, отчество и кафедру.

SELECT famil, kafedra
FROM prepodav
WHERE kafedra="Информатика и ВТ"

Для вывода новых данных, например, как в случае создания нового поля, необходимо использовать опцию AS.

4. Вывести фамилии студентов с указанием возраста. Предположим, что в таблице существует поле *Year*. В запросе создается новое поле *Vozr*, которое вычисляется через текущий год и год рождения студента.

SELECT fam, (2005-Year) AS Vozr
FROM prepodav

5. Вывести всех студентов, фамилии которых начинаются с «М». Символ «%» означает любое количество любых символов.

SELECT fam
FROM prepodav
WHERE fam LIKE 'M%'

Если в качестве критерия задать ***fam LIKE '%ван%'***, то результатом запроса будут и фамилии, которые включают символы «ван» в состав фамилии. Примером будут «Иванов», «Иванников», «Иванова» и «Ванюшкина».

6. Вывести всех преподавателей, родившихся в период с 1 января 1965 года по 31 декабря 1989 года.

SELECT famil, kafedra
FROM prepodav
WHERE Dat_rog BETWEEN '01.01.1965' AND '31.12.1989'

7. Вывести всех студентов, родившихся в 1985 и 1987 годах. Предположительно, что в таблице существует поле Year.

SELECT famil, kafedra
FROM prepodav
WHERE Year IN (1985, 1987)

8. Вывести списки студентов по группам. Предполагается, что список каждой группы будет представлен в алфавитном порядке фамилий.

SELECT fam, nam, otch, gruppa
FROM student
ORDER BY gruppa, fam

Необходимо заметить, что сортировка сначала происходит по группе, а затем по фамилии. Если нарушить последовательность, то список будет представлен алфавитным порядком фамилий; и в каждой группе одинаковых фамилий отсортируется группа. Последний вариант не подходит для описанного задания.

9. Вывести списки студентов в обратном алфавитном порядке.

SELECT fam, nam, otch
FROM student
ORDER BY fam DECS

Опция сортировки имеет подопцию, которая характеризует направление сортировки: *ASC* (Ascending – возрастание), *DESC* (Descending – убывание).

В представленном примере результирующий запрос будет содержать фамилии в обратном алфавитном порядке.

10. Вывести на экран общее количество записей таблицы СТУДЕНТ.

SELECT count(*)
FROM student

11. Вывести на экран количество записей, удовлетворяющих условию: коды студентов от 100 до 130.

SELECT count(*)
FROM student
WHERE Nomer BETWEEN 100 AND 130

12. Вывести на экран дату рождения самого младшего и самого старшего студентов группы ПИ-301.

```
SELECT min(Data_rog), max(Data_rog)
FROM student
WHERE gruppa='ПИ-301'
```

13. Вывести на экран информацию о количестве студентов в каждой группе.

```
SELECT gruppa, count(*)
FROM student
GROUP BY gruppa
```

14. Вывести на экран информацию о количестве студентов в каждой группе, кроме групп ПИ-101 и ПИ-102.

```
SELECT gruppa, count(*)
FROM student
GROUP BY gruppa
HAVING gruppa<>'ПИ-101' OR gruppa!='ПИ-102'
```

15. Вывести на экран фамилию и дату рождения самого старшего в группе ПИ-201.

```
SELECT fam, data_rog
FROM student
WHERE Data_rog = ( SELECT max(Data_rog)
FROM student
WHERE gruppa='ПИ-201' )
```

Вложенный запрос возвращает лишь одно значение – максимальную дату среди всех дат поля *Data_rog* в записях, где *gruppa='ПИ-201'*. Поэтому для главного запроса вложенный запрос становится лишь конкретным *значением*.

16. Допустим, что существует таблица НАГРАДА, в которой находятся фамилии студентов, награжденных за различные достижения. В отдел кадров необходимо предоставить по данным студентам всю информацию из таблицы СТУДЕНТ.

```
SELECT fam, nam, otch, gruppa, …
FROM student
WHERE Fam IN (SELECT fam FROM nagrada)
```

17. Вывести студентов из таблицы СТУДЕНТ, которые не старше любого студента в таблице СТУДЕНТ01.

```
SELECT *
FROM student
WHERE Data_rog >= ALL (SELECT Data_rog01 FROM student01)
```

18. Вывести преподавателей таблицы ПРЕПОДАВАТЕЛЬ, которые моложе ***хотя бы одного*** сотрудника в таблице ПРЕПОДАВАТЕЛЬ01.

```
SELECT *
FROM student
WHERE Data_rog > ANY (SELECT Data_rog01 FROM prepodav01)
```

19. Вывести информацию о сессиях (экзаменационных оценках) студентов группы ПИ-303.

```
SELECT fam, nam, gruppa, ozenka, data_ekz
FROM student, ozenka
WHERE (student.nomer=ozenka.nomer) AND (gruppa='ПИ-303')
```

В некоторых системах описание полей сопровождается именем таблицы как при указании связи между объектами. Если в двух таблицах есть одинаковые поля (по имени), то обязательно необходимо указывать таблицу, в которой это поле находится.

20. Вывести данные из двух таблиц (с использованием псевдонимов) по семейным студентам.

```
SELECT S.*, O.*
FROM student S, ozenka O
WHERE (S.nomer=O.nomer) AND (S.family=true)
```

21. Вывести студентов, обучающихся в одной группе.

```
SELECT p1.fam, p1.gruppa, p2.fam, p2.gruppa
FROM student p1, student p2
WHERE (p1.gruppa=p2.gruppa) AND (p1.fam!=p2.fam) AND
(p1.fam<p2.fam)
```

На рис. 12.1 приведен интерфейс приложения с описанным выше запросом:

Использование псевдонимов таблиц

Фамилия	Группа	Фамилия одногруппника	Группа второго студента
Семенова	ПИ-101	Сорокина	ПИ-101
Семенова	ПИ-101	Чкалова	ПИ-101
Сорокина	ПИ-101	Чкалова	ПИ-101
Семенов	ПИ-201	Соломеев	ПИ-201
Катин	ПИ-201	Соломеев	ПИ-201
Катин	ПИ-201	Семенов	ПИ-201
Гордеева	ПИ-202	Дымина	ПИ-202
Гордеева	ПИ-202	Моргунов	ПИ-202
Дымина	ПИ-202	Моргунов	ПИ-202
Иванова	ПИ-301	Маринадов	ПИ-301
Иванова	ПИ-301	Петрова	ПИ-301
Васечкина	ПИ-301	Маринадов	ПИ-301
Васечкина	ПИ-301	Петрова	ПИ-301
Васечкина	ПИ-301	Иванова	ПИ-301
Маринадов	ПИ-301	Петрова	ПИ-301
Иванов	ПИ-302	Циркович	ПИ-302
Иванов	ПИ-302	Кукушкина	ПИ-302
Кукушкина	ПИ-302	Циркович	ПИ-302

Рис. 12.1. Запрос на поиск одногруппников

Как видно, в запросе используется одна таблица, которая рассматривается под двумя псевдонимами. Это необходимо для того, чтобы выполнить задание. При этом ставятся следующие условия:

~ первое – для проверки равенства группы;

~ второе – для того, что студент при поиске не стал сам себе одногруппником;

~ третье – для удаления дублирования данных, так как система будет выдавать по две записи на каждую пару одногруппников.

12.3. Операции с записями

Для работы с записями таблицы используются три оператора:

~ INSERT – вставка записи в таблицу;

~ UPDATE – модификация записи таблицы;

~ DELETE – удаление текущей записи таблицы.

Формат оператора вставки записи:

INSERT INTO таблица [(поле1, поле2, поле3, …)]

VALUES (значение1, значение2, значение3, …)

Если запись добавляется в таблицу не по всем полям, то необходимо указывать список полей, а в опции VALUES значения полей указывать в том же порядке. Если запись добавляется полностью (по всем полям), то перечисление полей после названия таблицы необязательно.

Пример. Добавить в таблицу СТУДЕНТ запись по некоторым полям.

INSERT INTO student (nomer, fam, nam, gruppa, data_rog, family)

VALUES (200, 'Петрова', 'Мария', 'ПИ-101', '12.03.1989', false)

Необходимо иметь в виду, что поля, в которые пользователь не вводит сразу значений, должны иметь разрешение на наличие в них «пустых» значений.

Если необходимо ввести несколько записей, то можно воспользоваться подзапросом, например,

INSERT INTO student

(SELECT * FROM nagrada)

Необходимо заметить, что при добавлении из одной таблицы в другую полных записей (или из запроса), нужно строго следить за тем, чтобы обе таблицы имели одинаковую структуру.

Формат оператора изменения записи:

UPDATE таблица

SET поле1=значение1, поле2=значение2 [, …]

WHERE условие

Пример 1. Изменить у всех студентов сумму стипендий согласно индексации 15%. Предполагается, что в таблице СТУДЕНТ существует поле стипендии – *stipend*.

```
UPDATE student
SET stipend=stipend*1.15
```

Пример 2. Изменить студентке Петровой из группы ПИ-301 фамилию на Морозову в связи с документом о заключении брака.

```
UPDATE student
SET fam='Морозова'
WHERE fam='Петрова' AND gruppa='ПИ-301'
```

Формат оператора удаления записи:

```
DELETE
FROM таблица
[ WHERE условие ]
```

Если необходимо удалить все записи, то условие в операторе не задается.

Пример 1. Удалить все записи в таблице СТУДЕНТ.

```
DELETE
FROM student
```

*Пример 2.*Удалить в таблице СТУДЕНТ всех выпускников (возможно, что критерий не совсем корректен).

```
DELETE
FROM student
WHERE gruppa='ПИ-501' OR gruppa='ПИ-502' OR gruppa='Пи-503'
```

12.4. Операции с таблицами и индексами

Для работы с таблицами используют следующие операторы:

~ CREATE TABLE – создание таблицы;

~ DROP TABLE – удаление таблицы;

~ ALTER TABLE – изменение таблицы.

Формат оператора создания таблицы:

```
CREATE TABLE имя_таблицы
(
    имя_поля1 тип_поля [(размер)],
    имя_поля2 тип_поля [(размер)] [, ...]
)
```

Размер указывается только для полей строковых и некоторых других типов. После объявления некоторых полей могут включаться слова PRIMARY KEY, что указывает на то, что данное поле входит в первичный ключ. Кроме того, после объявления некоторых полей можно вставлять слова NOT NULL, означающие, что значение этого поля обязательно должно быть задано в каждой записи.

Пример. Создать таблицу СТУДЕНТ с полями, в которых не разрешаются «пустые» значения.

CREATE TABLE student
(
nomer INTEGER NOT NULL PRIMARY KEY,
fam char(20) NOT NULL,
nam char(15) NOT NULL,
data_rog date NOT NULL,
gruppa NOT NULL
)

Формат оператора изменения таблицы:

ALTER TABLE имя_таблицы
[DROP поле1, поле2, ...]
[ADD поле1 тип_поля [(размер)], ...]

Опция ADD используется для добавления поля, а опция DROP – для удаления поля.

Пример. Добавить в структуру таблицы СТУДЕНТ новое поле «Домашний адрес» и удалить поле «Группа».

ALTER TABLE student
DROP gruppa,
ADD Adres char(40)

Формат оператора удаления таблицы:

DROP TABLE имя_таблицы

Пример. Удалить таблицу СТУДЕНТ.

DROP TABLE student

Обычно создание структуры таблицы сопровождается созданием и индексов таблицы.

Формат оператора создания индекса:

CREATE INDEX имя_индекса
ON имя_таблицы поле1, поле2, поле3, ...

Пример 1. Создать индекс nmrS по полю Nomer таблицы СТУДЕНТ.

CREATE INDEX nmrS
ON student Nomer

Пример 2. Создать индекс grupfam по полям группа + фамилия.

CREATE INDEX grupfam
ON student gruppa, fam

Для удаления индекса используется команда DROP INDEX:

DROP INDEX student.grupfam

Если таблица многократно изменяется и в нее вносится много новых записей, индексы могут оказаться разбалансированы и их эффективность при выполнении запросов уменьшается. В этом случае полезно проводить повторное создание и балансировку индекса последовательным применением операторов деактивации и активации:

ALTER INDEX nmrS DEACTIVATE
ALTER INDEX nmrS ACTIVATE

12.5. Компонент Query

Для выполнения запроса в приложениях используется компонент *Query* со страницы *BDE*.

Однако необходимо заметить, что в серверных приложениях целесообразнее использовать компонент *Query*, а при работе с локальными базами данных – компонент *Table*.

Рассмотрим использование компонента *Query* в приложении просмотра данных. Для этого необходимо выполнить следующие действия:

1. Установить на форме компоненты *Query1*, *DataSource1* и *DBGrid1*.
2. Активизировать компонент *Query1* и установить свойство *DatabaseName*, значение которого равно псевдониму таблицы – *aliasDB*.
3. Активизировать компонент *DatabaseName1* и установить свойство *DataSet*, равное *Query1*.
4. Активизировать компонент *DBGrid1* и установить значение *DataSource1* в свойстве *DataSource*.
5. Активизировать компонент *Query1* и войти в редактор списка запроса свойства *SQL*.
6. В открывшемся окне *String List Editor* ввести SQL-команду:

SELECT p1.fam, p1.gruppa, p2.fam, p2.gruppa
FROM student p1, student p2

```
WHERE (p1.gruppa=p2.gruppa) AND (p1.fam!=p2.fam) AND
                                        (p1.fam<p2.fam)
```

7. Выйти из редактора списка запроса и установить свойство *Active* в состояние *true*.

После этого в компоненте *DBGrid1* должен появится результат запроса. В противном случае необходимо проверить SQL-предложение, то есть SELECT-команду.

Находясь в редакторе списка запроса (свойство SQL), пользователь через контекстное меню может воспользоваться командами *Save* и *Load*. Первая команда позволяет сохранить в текстовом файле (расширение *txt*) сам запрос, а вторая – загрузить предложение запроса из существующего текстового файла.

Необходимо заметить, что предлагаемая последовательность установки свойств для компонентов *Query*, *DataSource* и *DBGrid* выбрана в данном порядке из-за того, что неправильный подход не даст системе «увидеть» компонент *Query* в качестве источника.

Рассмотрим еще один пример.

Вывести в компонент DBGrid один из трех запросов согласно требованиям.

Для этого необходимо выполнить следующие действия:

1. Создать новый проект и сохранить его.
2. Установить на форме компоненты *DataSource1, DBGrid1, Query1, Query2, Query3*, три кнопки для вызовов запросов (при необходимости оформить метки).
3. Активизировать по очереди компоненты *Query* и установить для него свойство *DatabaseName*, равное псевдониму области, в которой хранятся все таблицы базы данных.
4. Активизировать компонент *DBGrid1* и установить для него свойство *DataSource*, равное *DataSource1.*
5. Активизировать первую кнопку и ввести в программном коде события *OnClick* следующие операторы:

```
    // дезактивировать второй и третий запросы
Query2->Active=false;
Query3->Active=false;
    // установить в качестве источника данных результаты запроса 1
DataSource1->DataSet=Query1;
    // активировать запрос 1
Query1->Active=true;
    // передать управление компоненту-просмотрщику таблиц для
    // активизации последнего
DBGrid1->SetFocus();
```

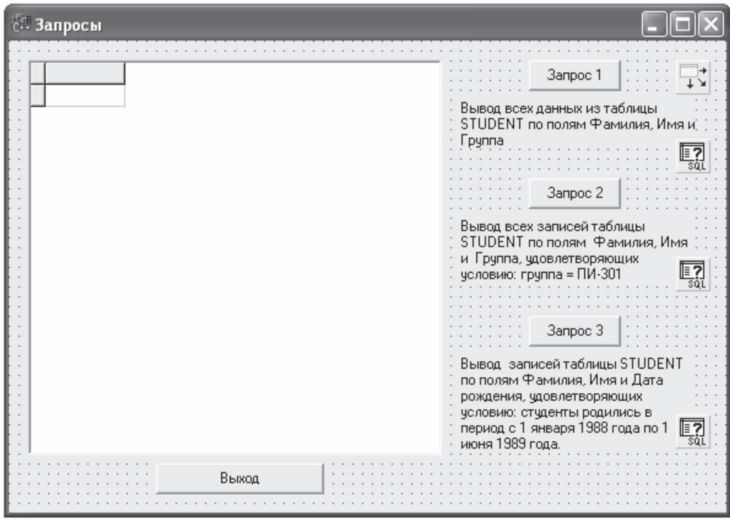


Рис. 12.2. Рабочий режим приложения

6. Аналогичным образом оформить программные коды для двух оставшихся кнопок, учитывая то, что в определенный момент активным должен быть только один компонент *Query*.
7. Сохранить проект и проверить его работу (рис. 12.3).

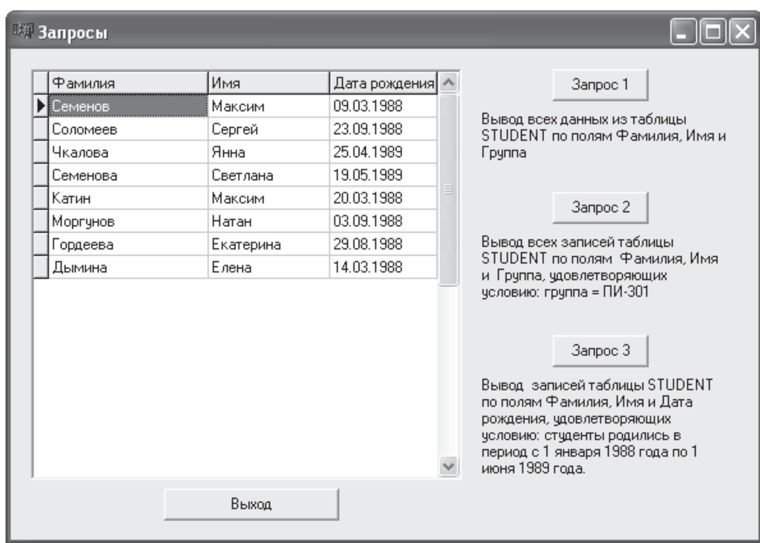


Рис. 12.3. Результат запроса

12.6. Динамические запросы

Все запросы SQL, которые до сих пор рассматривались – это так называемые *статические запросы*. В них фиксировано все: имена таблиц, поля, константы в выражениях и т. п. Но помимо таких статических запросов SQL допускает и *динамические запросы*, использующие параметры. Причем параметры можно применять вместо имен таблиц, имен полей и их значений. Значения этих параметров передаются извне и тем самым, не изменяя текст самого запроса, можно менять возвращаемый им результат.

Параметры задаются в запросе с двоеточием, предшествующим имени параметра:

: имя_параметра

Например, если в запросе SELECT элемент WHERE записан в виде:

WHERE Data_rog <=: dDate

то для сравнения используется не значение поля, а значение параметра *dDate*.

Рассмотрим компонент *Query*, точнее как он будет работать с данным параметром.

Предположим, что создан новый проект, и на форме установлены компоненты *Query, DataSource, DBGrid*. При этом настройки все произведены, а в свойстве SQL компонента *Query* введен следующий запрос:

SELECT fam, nam, otch, adres, gruppa, family
FROM student
WHERE (gruppa LIKE:sGrup) AND (family=:true)

Далее выполним следующие действия:

1. После этого произвести «щелчок» в «Инспекторе объектов» на свойстве *Params*, в результате чего появится дополнительное окно (рис. 12.4).

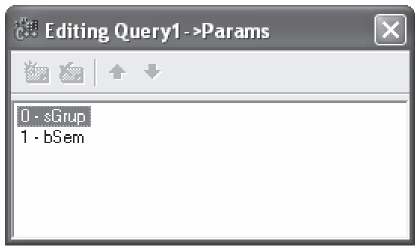
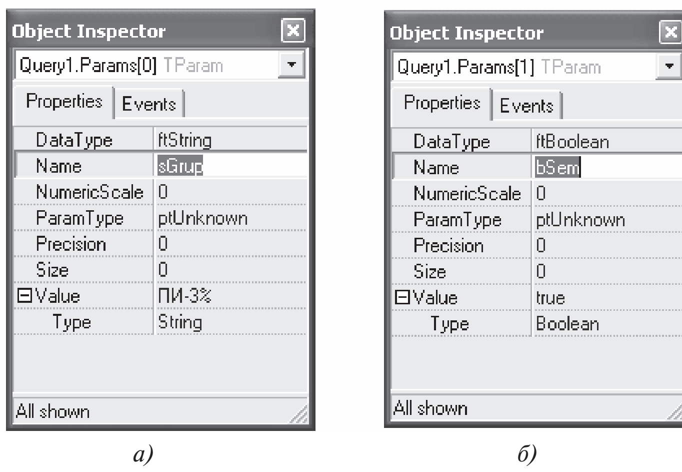


Рис. 12.4. Окно параметров

2. Активизировать первый параметр *dDate*. Обратить внимание на то, что первый параметр обозначается в «Инспекторе объектов» как *Query1.Params[0]*.

3. Установить при необходимости следующие свойства: *DataType* (тип данных), *Name* (идентификатор), *ParamType* (тип параметра – используется при обращении к процедурам, хранимым на сервере), *Precision* (количество знаков после запятой), *Size* (размер), *Value* (значение параметра по умолчанию), *Type* (тип значения *Value* по умолчанию), используя рис. 12.5*а*.



а) *б)*

Рис. 12.5. Свойства параметров sGrup и bSem

Необходимо заметить, что для типа данных используется константа, о которой было упоминание в предыдущей работе, когда создавалась таблица. Поэтому в этом свойстве (представленном списком) можно более подробно ознакомиться с константами типов данных.

4. Аналогичным способом установить свойства и для параметра *bSem* (рис. 12.5*б*).

5. Активизировать компонент *Query* через свойство *Active = true*.

6. Сохранить проект и проверить его работу (рис. 12.6).

Фамилия	Имя	Отчество	Группа	Домашний адрес	Семейность
Кукушкина	Ольга	Александровна	ПИ-302	ул.Жукова, 12-387	True
Иванов	Сергей	Александрович	ПИ-302	ул.Ворошилова, 34-190	True
Лунина	Елена	Олеговна	ПИ-303	ул.Жукова, 2-90	True
Васечкина	Наталья	Валентиновна	ПИ-301	ул.Ворошилова, 3-49	True
Горин	Алексей	Николаевич	ПИ-303	ул.Королева, 2-49	True
Маринадов	Сергей	Александрович	ПИ-301	ул.Ворошилова, 34-91	True

Рис. 12.6. Результат запроса с параметрами

Программный доступ к параметрам во время выполнения приложения осуществляется аналогично доступу к полям набора данных. Свойство *Params* является указателем на массив параметров типа *TParam*, к элементам которого можно обращаться по индексу через его свойство *Items[Word Index]*. Последовательность, в которой располагаются параметры в массиве, определяется последовательностью их упоминания в запросе SQL.

Значения параметров, как и значения полей, определяются такими свойствами объектов-параметров, как *Value*, *AsString*, *AsInteger* и т. п. Например, оператор

```
for (int I=0; I < Query1->Params->Count; I++)
    if (Query1->Params->Items[I]->IsNull &&
                        Query1->Params->Items[I]->DataType ==ftInteger)
            Query1->Params->Items[I]->AsInteger = –1;
```

задаст значение «-1» всем целым параметрам, которым до этого не было присвоено значение, В нем использовано свойство *Count* – число параметров, свойство *isNull*, равное true, если параметру не задано никакое значение, свойство *DataType*, указывающее тип параметра, и свойство *AsInteger*, дающее доступ к значению параметра как к целому числу.

Другой пример: операторы

```
Query1->Params->Items[0]->AsString = "ПИ-201";
Query1->Params->Items[1]->AsBoolean= true;
```

задают значения первому (индекс 0) и второму (индекс 1) параметрам компонента *Query1*, в свойстве SQL. которого записан приведенный ранее оператор *Select* с параметрами:*sGrup* и:*bSem*. Поскольку в этом операторе параметр:sGrup упоминается первым, то его индекс равен 0.

Кроме свойства *Items,* у *Params* есть еще одно свойство – *ParamValues*. Оно представляет собой массив значений параметров типа *Variant*. В качестве индекса в это свойство передается имя параметра или несколько имен, разделяемых точками с запятой. Например, операторы

```
Query1->Params->ParamValues["sGrup"] = "ПИ-3%";
Query1->Params-> ParamValues["bSem"]= true;
```

задают те же значения параметрам, что и приведенные выше. Преимуществом является то, что при записи этих операторов не надо помнить индексы параметров. Другим преимуществом свойства *ParamValues* является возможность задать значения сразу нескольким параметрам. Например:

```
Variant par[] = {"ПИ-3%", true};
Query1->Params->ParamValues["sGrup; bSem"] = VarArrayOf (par,1);
```

Другой способ обращения к параметрам, при котором не надо помнить их индексы – использование метода *ParamByName* компонента *Query*. Например, операторы

Query1->ParamByName("sGrup")->AsString = "ПИ-3%";
Query1-> ParamByName("bSem")->AsBoolean = true;
задают параметрам с именами *sGrup* и *bSem* те же значения, что и приведенные ранее операторы.

Следует оговориться, что задание нового значения параметру само по себе еще не обеспечивает влияния на возвращаемый из запроса результат. Надо повторно выполнить данный запрос, чтобы ощутить изменения.

12.7. Связывание таблиц

Большинство свойств *Query* аналогичны свойствам *Table*. Программный доступ к этим полям осуществляется так же, как в *Table*, с помощью свойства *Fields* (например, *Query1->Fields[0]*) или методом *FieldByName* (например, *Query1->FieldByName("Gruppa"))*. Можно также создавать объекты полей с помощью «Редактора полей», вызываемого двойным щелчком на *Query* или из меню, всплывающего при щелчке на *Query* правой кнопкой мыши. В этом случае доступ к объекту поля можно осуществлять также и по его имени (например, *Query1Gruppa*).

Для доступа к значениям полей используются те же их свойства *Value*, *AsString, AsInteger* и т. п., что и в *Table*. Точно так же, как в *Table*, можно осуществлять навигацию по набору данных, устанавливать фильтры, ограничивать вводимые значения полей, кэшировать изменения.

Из свойств, отличных от *Table*, остановимся на свойстве *DataSource*. Это свойство позволяет строить приложения, содержащие связанные друг с другом таблицы.

Рассмотрим пример использования запроса к двум взаимосвязанным таблицам.

Для этого необходимо выполнить следующие действия:

1. Создать новое приложение.
2. Установить на форме два набора компонентов *Query*, *DataSource* и *DBGrid*. (рис. 12.6).
3. Настроить первый набор компонентов: в *DBGrid1* свойство *DataSource=DataSource1;* в *DataSource1* свойство *DataSet=Query1;* в *Query1* свойства *DatabaseName = aliasDB* и *SQL*:

   SELECT * FROM student
4. Активизировав компонент *Query1*, проверить наличие записей в компоненте *DBGrid1.*
5. Сохранить проект и проверить работу.

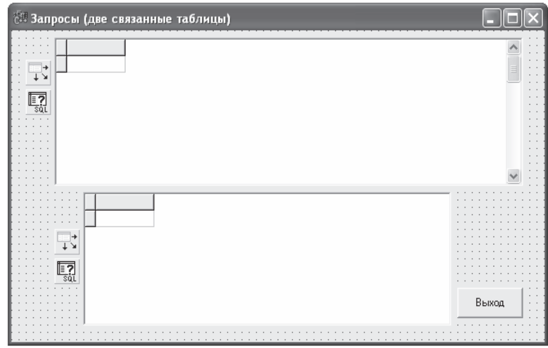


Рис. 12.6. Вид формы в режиме редактора

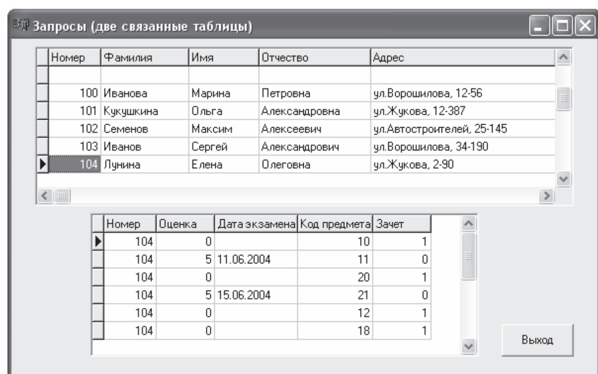
6. Аналогичным способом установить свойства второго набора компонентов и задать запрос для второй таблицы:

SELECT * FROM ozenka

7. Сохранить проект и проверить работу приложения, убедившись в том, что оба набора компонентов работают самостоятельно, но не в связи друг с другом.
8. Вернуться в рабочий режим проекта.
9. Дезактивировать компоненты *Query*.
10. Активизировать свойство SQL запроса Query2 и дополнить его связью таблиц:

SELECT * FROM ozenka WHERE (Nomer=:Nomer)

11. Изменить для компонента *Query2* свойство *DataSource=DataSource1* (то есть достаточно изменить источник данных для дочернего запроса).
12. Активировать компоненты *Query*.
13. Сохранить проект и проверить его работу (рис. 12.7).



Номер	Фамилия	Имя	Отчество	Адрес
100	Иванова	Марина	Петровна	ул.Ворошилова, 12-56
101	Кукушкина	Ольга	Александровна	ул.Жукова, 12-387
102	Семенов	Максим	Алексеевич	ул.Автостроителей, 25-145
103	Иванов	Сергей	Александрович	ул.Ворошилова, 34-190
104	Лунина	Елена	Олеговна	ул.Жукова, 2-90

Номер	Оценка	Дата экзамена	Код предмета	Зачет
104	0		10	1
104	5	11.06.2004	11	0
104	0		20	1
104	5	15.06.2004	21	0
104	0		12	1
104	0		18	1

Рис. 12.7. Результат запроса

К основным методам *Query* можно отнести методы открытия и закрытия соединения с базой данных.

Метод *Close* закрывает соединение с базой данных, а метод *Open* открывает соединение.

Если запрос заранее не задан, а должен быть сформирован в течение работы программы, то можно запрос в виде SQL-предложения задать переменной, например, ssql. Тогда фрагмент программы, открывающий и выполняющий запрос, будет выглядеть следующим образом:

```
    // очистить SQL-предложение
Query1->SQL->Clear();
    // добавить из переменной ssql SQL-предложение
Query1->SQL->Add(ssql);
    // открыть SQL-предложение
Query1->Open();
    // запустить на выполнение SQL-предложение
Query1->ExecSQL();
```

Вопросы для самоконтроля

1. Каково назначение языка SQL?
2. С какой целью используется оператор SELECT?
3. Каков формат команды SELECT при работе с одной и более таблицами? Пояснить.
4. Что такое функции агрегации?
5. Что такое псевдоним таблицы? Как он задается?
6. Какие операторы используются для работы с записями? Перечислить и охарактеризовать.
7. Какие операторы используются для работы с таблицами? Перечислить и охарактеризовать.
8. Какой из визуальных компонентов работает с запросами на форме? Какие свойства данного компонента необходимо устанавливать для его настройки?
9. Что может служить источником данных для компонента Query?
10. Каким подходом к работе похожи компоненты Query и Table?
11. Что такое динамические запросы? Дать определение.
12. В чем особенность задания динамического запроса? Привести пример.
13. Как задаются значения параметров в запросе?
14. Каким образом происходит связывание таблиц в запросе (при выводе информации)?
15. Привести примеры использования команды SELECT.

13. СОЗДАНИЕ ОТЧЁТОВ

13.1. Компонент QuickRep

Для создания отчётов в C++Builder включена система *QuickReport*. Компоненты этой системы размещены на странице *QReport* палитры компонентов.

QuickReport использует генератор отчетов, состоящих из множества полос. Полоса (*band*) – это область отчета или раздел, содержащий некоторый текст, изображения, графики, диаграммы и т. п. Полоса является контейнером для других компонентов, вносящих в отчет информацию или графику.

Если полоса и размещенные на ней компоненты связаны с базой данных, то содержание этой полосы печатается столько раз, сколько соответствующих записей имеется в источнике данных. Таким образом, достаточно расположить компоненты, связанные с данными, на полосе, а печатаемые значения и их количество будут автоматически управляться базой данных.

Как в компонентах, связанных с данными, вы можете задавать головную и вспомогательную таблицы, так и между полосами можно задавать аналогичные связи. Таким образом, все возможности, реализуемые в приложениях, реализуются с тем же успехом и в отчетах.

Основным компонентом, на котором строится весь отчет, является *QuickRep*. Он предоставляет ряд возможностей по управлению создаваемым отчетом, включая формирование заголовка, полос, шрифтов, установок принтера и др. Этот компонент является визуальным и после его соединения с базой данных может использоваться как контейнер полос, составляющих отчёт.

Компонент *QuickRep* имеет ряд свойств, определяющих характеристики печати отчёта:

PrinterSetting	– задает число копий отчета и диапазон печатаемых страниц;
Page	– задает размер страницы *PaperSize* (можно установить заказной размер – *Custom* и определить длину и ширину страницы свойствами *Length* и *Width*), ее ориентацию и поля;
Options	– определяет, надо ли печатать верхний колонтитул первой страницы (*FirstPageHeader*) и нижний колонтитул последней (*LastPageFooter*);
Units	– задает единицу измерения размеров страницы, полей и т. п.: миллиметры, дюймы, пикселы и т. д.;
Zoom	– масштаб печати в процентах;
ReportTitle	заголовок окна предварительного просмотра.

Свойство *DataSet* определяет набор данных, к которому подключается отчет. Этим набором может являться компонент типа *TTable*, *TQuery* и т. п.

Компонент *QuickRep* имеет два основных метода: *Preview* – предварительный просмотр, и *Print* – печать. Предварительный просмотр и даже печать отчета можно осуществлять и в процессе проектирования. Для этого надо щелкнуть правой кнопкой мыши на компоненте *QuickRep* и из всплывшего меню выбрать команду Preview. Перед вами откроется окно предварительного просмотра, в котором, в частности, имеется кнопка печати.

Компоненты *QRLabel, QRMemo, QRRichText, QRShape, QRImage,* размещаемые на полосах отчета, являются аналогами обычных компонентов – *Label, Memo, RichEdit, Shape, Image*. Основной особенностью соответствующих компонентов QuickReport является их способность печататься в тех полосах отчета, в которых они размещены. Компоненты имеют два свойства, отсутствующих в обычных компонентах: *Frame* и *Size*.

Свойство *Frame* имеет ряд подсвойств, определяющих рамку вокруг компонента: *Color* – цвет, *Style* – стиль, *Width* – ширина, *DrawBottom, DrawLeft, DrawRight, DrawTop* – определяют наличие рамки соответственно внизу, слева, справа и вверху компонента.

Свойство *Size* имеет подсвойства, определяющие размер и место размещения компонента при печати. Все определяется в единицах измерения, заданных свойством *Units* компонента *QuickRep*.

Некоторые компоненты имеют свойство *AlignToBand* – выравнивание в полосе. Если это свойство установить в *true*, то компонент будет выровнен по краю полосы, заданному свойством *Alignment: taLeftJustify* – влево, *taCenter* – по центру, *taRightJustify* – вправо.

При установке полос отчета используется свойство *Bands*, которое в своем составе имеет несколько полос. Чтобы та или иная полоса была доступна в отчете, необходимо установить значение *true* напротив соответствующей полосы:

HasTitle	– имеется полоса заголовка отчета, которая печатается один раз в начале отчета;
HasDetail	– имеется полоса детализации, которая печатается столько раз, сколько записей в нее передается;
HasPageHeader	– имеется верхний колонтитул (заголовок) на каждой странице отчета;
HasPageFooter	– имеется нижний колонтитул на каждой странице отчета;
HasColumnHeader	– имеется заголовок печатаемой таблицы.

Компонент *QRSysData* используется для того, чтобы устанавливать какие-либо системные параметры, например, дату, время номер страницы и т. п.

qrsDate – текущая дата;

qrsDateTime – текущие дата и время;

qrsDetailCount – число записей в базе данных;

qrsDetailNo – текущий номер записи в базе данных;

qrsPageNumber – номер текущей страницы;

qrsReportTitle – заголовок отчета;

qrsTime – текущее время.

Если необходимо установить ***вычисляемое*** поле, то используется компонент *QRExpr*.

13.2. Разработка простых отчётов

Рассмотрим разработку нескольких отчетов, начиная с простого и заканчивая более сложными (с использованием нескольких таблиц).

Пример 1. Создать отчет – список всех студентов.

Для реализации данной задачи необходимо выполнить следующие действия:

1. Создать новый проект.
2. Установить на форме со страницы *QReport* компонент *QuickRep*. При этом компонент приобретет по умолчанию размеры, характерные для листа формата А4.

В случае, если размеры листа (по умолчанию) не устраивают пользователя, то можно произвести «двойной щелчок» на области компонента *QuickRep*. В результате чего появится окно *Report Settings*. В данном окне можно установить различные параметры отчета. Список *Paper Size* позволяет установить различные размеры отчета.

3. Установить на форме компоненты *Table* и *DataSource*.
4. Настроить компоненты таблицы и источника на соответствующую таблицу СТУДЕНТ.
5. Активизировать компонент *QuickRep* и раскрыть свойство *Bands*.
6. Установить в значение *true* полосы *HasTitle, HasColumnHeader, HasDetail* (рис. 13.1).

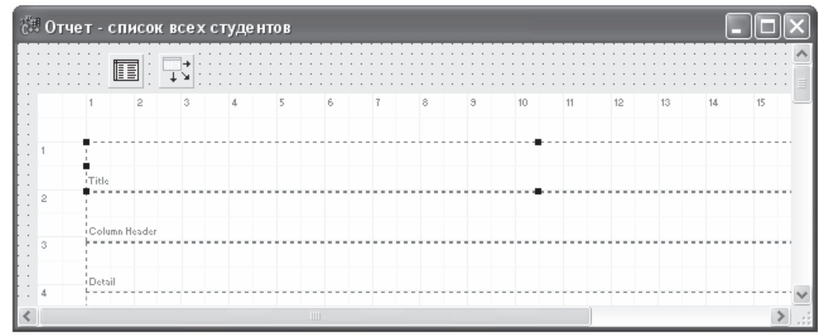


Рис. 13.1. Полосы отчета

Эти полосы будут добавлены именно в указанном порядке, хотя добавлять их можно в любой последовательности. Данный порядок объясняется просто: заголовок для всего отчета, затем идет «шапка» таблицы в области заголовков столбцов, а завершается простой отчет (без каких-либо итоговых данных) списком данных.

Кроме того, необходимо понять следующее (это характерно для всех отчетов в различных системах): область «детализации» (*Detail*) содержит только названия полей или выражений (с полями), а при просмотре (выводе) эти поля повторяются столько раз, сколько выводится записей. Единожды установив поле, пользователь получает столько ***значений*** этого поля, сколько записей соответствует условию вывода.

7. При необходимости изменить размеры областей полос.
8. Установить в полосе заголовка (первой полосе) компонент *QRLabel* и задать ему заголовок «*СПИСОК*».
9. Аналогично установить метку «*студентов специальности 351400*».
10. Используя свойство *Font*, установить необходимые параметры.
11. Вызвать контекстное меню отчета и командой *Preview* просмотреть отчет. Убедиться в том, что заголовок в отчете присутствует.
12. Используя компонент *QRShape*, во второй полоске нарисовать графы для заголовков полей.
13. Установить во второй полоске (в графах) заголовки полей компонентами *QRLabel*.
14. В области Detail установить компоненты *QRDBText* под заголовками.
15. Активизировать первый компонент *QRDBText1* и установить для него следующие свойства: *DataSet = Table1*; *DataField = Fam*. После чего компонент в области отчета приобретет имя поля.
16. Аналогичным способом настроить остальные поля (рис. 13.2).

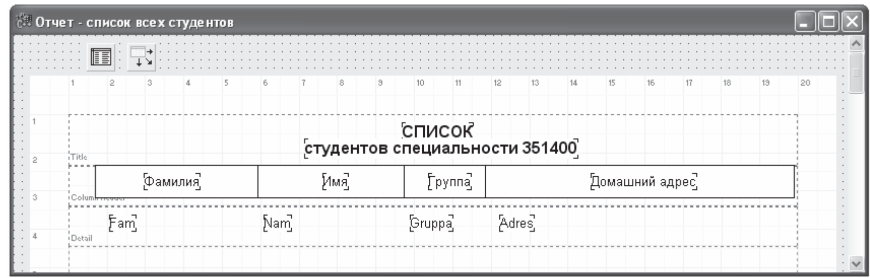


Рис. 13.2. Рабочий режим отчёта

17. Активизировать таблицу.

18. Выделить область компонента отчета *QuickRep* и установить для него свойство *DataSet = Table1.*

Необходимо заметить, что активизация таблицы является недостаточным условием демонстрации отчета. Обязательно нужно установить свойство *DataSet = источник*. Это может быть и таблица, и запрос.

Результат отчета приведен в сокращенном виде (рис. 13.3).

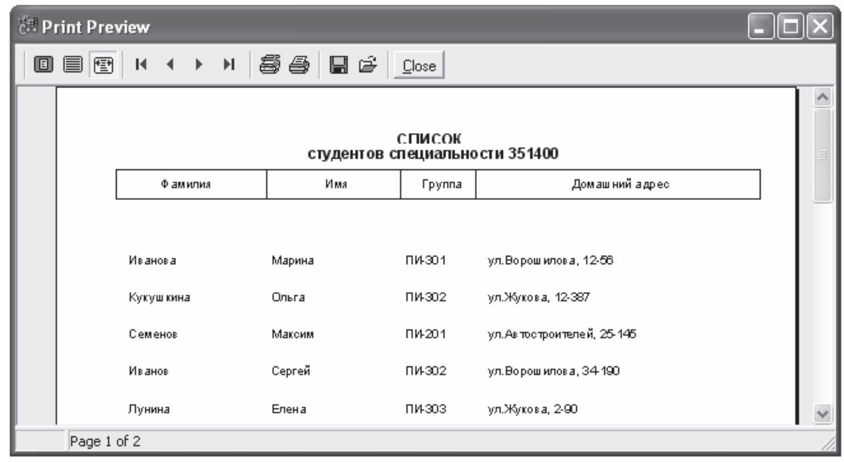
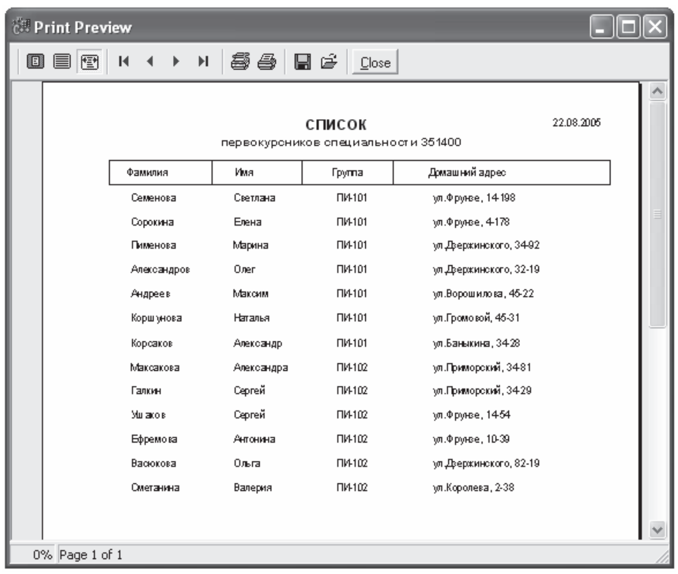


Рис. 13.3. Фрагмент демонстрации отчёта

Рассмотрим еще один отчёт.

Пример 2. Вывести список первокурсников специальности 351400 (рис. 13.4).



СПИСОК

первокурсников специальности 351400

22.08.2005

Фамилия	Имя	Группа	Домашний адрес
Семенова	Светлана	ПИ-101	ул.Фрунзе, 14-198
Сорокина	Елена	ПИ-101	ул.Фрунзе, 4-178
Пименова	Марина	ПИ-101	ул.Дзержинского, 34-92
Александров	Олег	ПИ-101	ул.Дзержинского, 32-19
Андреев	Максим	ПИ-101	ул.Ворошилова, 45-22
Коршунова	Наталья	ПИ-101	ул.Громовой, 45-31
Корсаков	Александр	ПИ-101	ул.Баныкина, 34-28
Максакова	Александра	ПИ-102	ул.Приморский, 34-81
Галкин	Сергей	ПИ-102	ул.Приморский, 34-29
Ушаков	Сергей	ПИ-102	ул.Фрунзе, 14-54
Ефремова	Антонина	ПИ-102	ул.Фрунзе, 10-39
Васюкова	Ольга	ПИ-102	ул.Дзержинского, 82-19
Сметанина	Валерия	ПИ-102	ул.Королева, 2-38

Рис. 13.4. Отчёт – список первокурсников

Разработка его отличается от первого отчёта тем, что в качестве источника используется не таблица, а *запрос*.

Необходимо для запроса *Query* задать SQL-предложение:

SELECT fam, nam, gruppa, adres
FROM student
WHERE gruppa=”ПИ-101” OR gruppa=”ПИ-102”

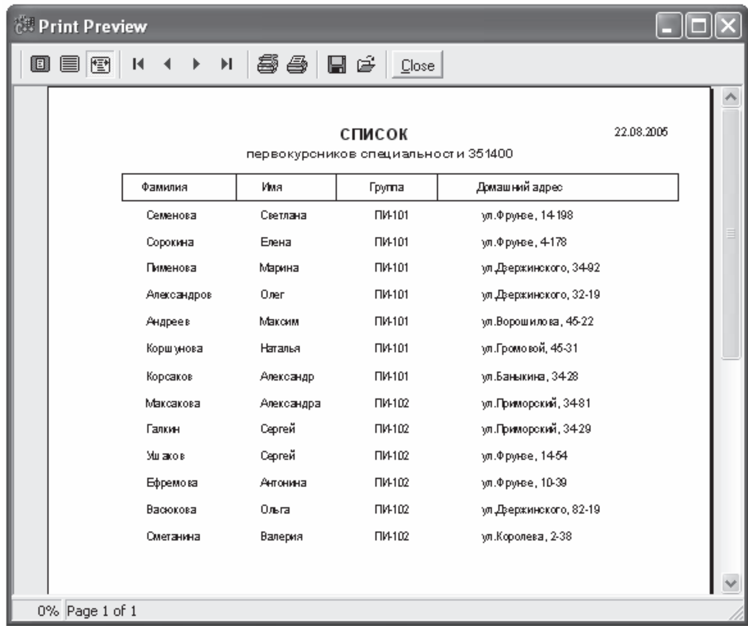
В полосе *Title* установить компонент *QRSysData*, после чего в свойстве *Data* выбрать значение *qrsDate* (даты).

Указание источника для компонента *QuickRep* обязательно: *DataSet = Query1.*

13.3. Разработка отчётов по взаимосвязанным таблицам

До сих пор рассматривались отчеты, которые выводились по одной таблице. Работа с двумя и более таблицами заключается в том, что необходимо не просто настроить таблицы на определенные компоненты, а установить связь между таблицами.

Пример. Вывести отчет по преподавателям с указанием предметов преподавания (рис. 13.5).



СПИСОК
первокурсников специальности 351400

22.08.2005

Фамилия	Имя	Группа	Домашний адрес
Семенова	Светлана	ПИ-101	ул.Фрунзе, 14-198
Сорокина	Елена	ПИ-101	ул.Фрунзе, 4-178
Пименова	Марина	ПИ-101	ул.Дзержинского, 34-92
Александров	Олег	ПИ-101	ул.Дзержинского, 32-19
Андреев	Максим	ПИ-101	ул.Ворошилова, 45-22
Коршунова	Наталья	ПИ-101	ул.Громовой, 45-31
Корсаков	Александр	ПИ-101	ул.Баныкина, 34-28
Максакова	Александра	ПИ-102	ул.Приморский, 34-81
Галкин	Сергей	ПИ-102	ул.Приморский, 34-29
Ушаков	Сергей	ПИ-102	ул.Фрунзе, 14-54
Ефремова	Антонина	ПИ-102	ул.Фрунзе, 10-39
Васюкова	Ольга	ПИ-102	ул.Дзержинского, 82-19
Сметанина	Валерия	ПИ-102	ул.Королева, 2-38

Рис. 13.5. Отчёт по двум таблицам

Для реализации данного отчёта необходимо выполнить следующие действия:

1. Установить на форму два набора компонентов для настройки на таблицу ПРЕПОДАВАТЕЛЬ и ПРЕДМЕТ.
2. Установить связь между таблицами традиционным способом, при этом таблицы не активировать.
3. Установить на форме компонент *QuickRep* (с идентификатором *QuickRep1*).
4. В свойстве *Bands* установить свойства: *HasTitle = true*.
5. В области заголовка установить компоненты *QRLabel* и задать две строки заголовка: «*СПИСОК*» и «*преподавателей*».
6. Добавить в отчет область *QRSubDetail* (с идентификатором *QRSubDetail1*).
7. Установить для данной области свойство *DataSet = Table1*.
8. Установить в области детализации компоненты *QRLabel* (название «Преподаватель») и *QRDBText*.
9. Активизировать компонент *QRDBText* и установить для него свойства: *DataSet = Table1, DataField = famil*.
10. Активизировать таблицу *Table1* и проверить работу отчета. Убедиться в том, что в отчете присутствует список преподавателей.

11. Дезактивировать таблицу ПРЕПОДАВАТЕЛЬ.

12. Добавить в отчет еще один компонент *QRSubDetail* (с идентификатором *QRSubDetail2*) и установить свойство *DataSet = Table2.*

13. Установить во второй области детализации компоненты *QRDBText* для отображения полей предмета и семестра из таблицы ПРЕДМЕТ (*Table2*).

14. Добавить в отчет комментарии по кафедре и семестру (рис. 13.6).

15. Сохранить проект и проверить работу отчета. Убедиться в том, что отчет не совсем корректно отображает данные; это связано с тем, что в отчете не указана связь между областями (хотя связь между таблицами установлена).

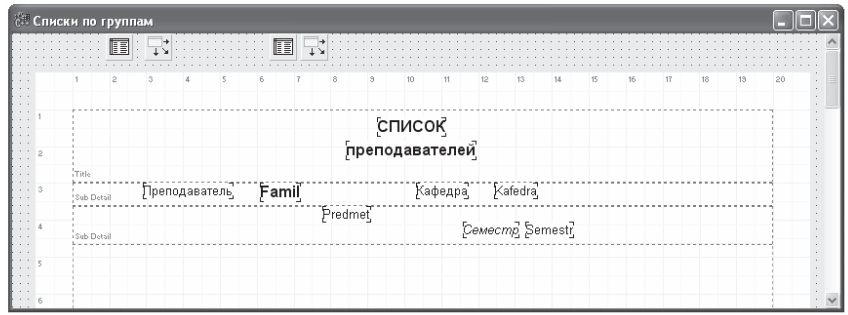


Рис. 13.6. Рабочий режим отчёта

16. Вернуться в рабочий режим проекта.

17. Активизировать вторую область детализации (*QRSubDetail2*) и войти в свойство *Master*.

18. Установить значение *QRSubDetail1* в свойстве *Master*. Это означает, что первая область будет родительской для второй (дочерней) области.

19. Активизировать все таблицы.

Необходимо заметить, что две области детализации рассматриваются как родительская и дочерняя области; при этом первая область организует внешний цикл – для фамилий преподавателей первой таблицы, а вторая область – внутренний цикл для предметов, которые представлены во второй таблице. Отношение между таблицами ПРЕПОДАВАТЕЛЬ и ПРЕДМЕТ – «один-ко-многим».

Следующий пример не связан с двумя таблицами, но он использует прием, описанный выше.

Пример 3. ***Вывод списков студентов по группам.***

На рис. 13.7 приведен отчёт, который обрабатывает только одну таблицу, но списки выводит по группам. Если бы группы и фамилии

находились в разных таблицах, то пришлось бы воспользоваться последовательностью действий, описанных в последующем пункте. В данной задаче необходимо прибегнуть к небольшой хитрости.

Print Preview

Close

СПИСОК
студентов специальности 351400

Группа **ПИ-101**

Александров	Олег	Александрович	ПИ-101
Андреев	Максим	Петрович	ПИ-101
Корсаков	Александр	Васильевич	ПИ-101
Коршунова	Наталья	Васильевна	ПИ-101
Пименова	Марина	Александровна	ПИ-101
Семенова	Светлана	Александровна	ПИ-101
Сорокина	Елена	Александровна	ПИ-101
Чкалова	Яна	Олеговна	ПИ-101

Группа **ПИ-102**

Васюкова	Ольга	Павловна	ПИ-102
Галкин	Сергей	Витальевич	ПИ-102
Ефремова	Антонина	Александровна	ПИ-102
Максакова	Александра	Валентиновна	ПИ-102
Сметанина	Валерия	Сергеевна	ПИ-102
Ушаков	Сергей	Васильевич	ПИ-102
Цыфаркина	Ольга	Александровна	ПИ-102

Группа **ПИ-201**

Катин	Максим	Петрович	ПИ-201
Семенов	Максим	Алексеевич	ПИ-201

0% Page 1 of 1

Рис. 13.7. Отчёт по студентам специальности 351400

Для реализации задачи необходимо выполнить следующие действия:

1. Установить на форме два набора компонентов: первый набор *Query1, DataSource1* и второй набор – *Table1, DataSource2.*
2. Настроить первый набор на запрос, который выводит только список групп:

SELECT gruppa
FROM student
GROUP BY gruppa

Необходимо обратить внимание на то, что, если не воспользоваться в запросе группировкой, то результат запроса будет содержать столько значений групп, сколько записей в таблице СТУДЕНТ. Использование группировки позволяет выводить каждое значение только один раз.

3. Настроить второй набор на таблицу *Table1* (СТУДЕНТ).

Учитывая то, что результат запроса представляет собой временную таблицу, можно установить связь между результатом запроса как родительской таблицей и таблицей СТУДЕНТ как дочерней, используя для связи поле *gruppa*.

4. Активизировать компонент *Table1* и установить для него свойство *MasterSource = DataSource1.*

5. Активизировать свойство *MasterFields* и войти в окно установки связей между таблицами (рис. 13.8).

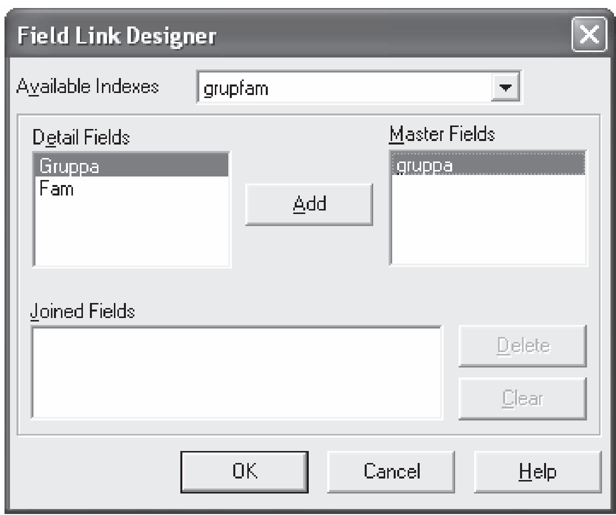


Рис. 13.8. Окно установки связи между таблицами

6. Выбрать индексы поля для связи из запроса (*Gruppa*) и из таблицы СТУДЕНТ (*gruppa*), то есть установить связь *Query1.Gruppa = Table1.gruppa*.

7. Установить на форме компонент *QuickRep* и добавить в отчёт компоненты согласно рис. 13.9: полоску *Title* и полоски *QRSubDetail1* и *QRSubDetail2.*

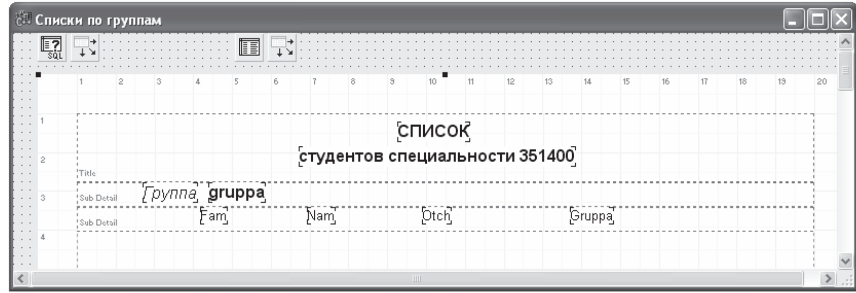


Рис. 13.9. Рабочий режим отчёта

8. Установить для каждой области детализации свойство *DataSet*. Для первой – запрос, для второй – таблицу.

9. В первой области установить комментарий и поле *gruppa* из запроса.

10. Во второй области установить поля *fam, nam, otch, gruppa* из таблицы для вывода данных.

Необходимо помнить, что связи между запросом и таблицей недостаточно для отображения нужных данных в отчёте. Нужно установить связь и между областями детализации.

11. Активизировать вторую область детализации и установить свойство *Master* со значением *QRSubDetail1*, которое характеризует первую область детализации.

12. Активизировать запрос и таблицу.

Представленная последовательность операций позволяет разработать отчёт по двум таблицам с учётом вложенности (группировки) данных.

Вопросы для самоконтроля

1. Для какой цели разрабатываются отчёты в приложениях?
2. Какой компонент системы используется для разработки отчёта?
3. Что представляет собой область отчёта в конструкторе (в режиме разработки)?
4. Какие вспомогательные компоненты системы используются для организации и/или оформления отчёта?
5. Какие свойства компонента QuickRep необходимо задать для разработки отчёта?
6. Что представляет собой «простой» отчёт? Охарактеризуйте.
7. Что представляет собой отчёт по взаимосвязанным таблицам? Охарактеризовать.
8. Каким образом осуществляется группировка в отчётах? Для какой цели эта группировка используется?

ЛАБОРАТОРНЫЕ РАБОТЫ

Лабораторная работа № 1
Создание простейшего приложения

Теоретический материал: п. 1.

Варианты

1. Разработать и реализовать программу для нахождения скорости животных, если известно, что путь в 60 км заяц-русак пробегает за 1 ч, а волк – за 1 ч 20 мин. Данные вводятся с клавиатуры.
2. Разработать и реализовать программу для нахождения времени полета ракеты, если известно, что ракета со скоростью 8 км/с отлетела от Земли на расстояние 480 км. Какое время потребуется на преодоление такого же расстояния самолету ИЛ–18, если его скорость 0,18 км/с?
3. Разработать и реализовать программу для определения объема пирамиды, если известно, что ее высота равна 12 см, основание имеет вид прямоугольника со сторонами 8,3 см и 4 см.
4. Разработать и реализовать программу для определения массы и объема куба, если известно, что ребро равно 23,1 см, а плотность вещества, из которого он сделан, 2700 кг/м^3.
5. Разработать и реализовать программу для определения объема и площади полной поверхности прямоугольного параллелепипеда, если известно, что измерения параллелепипеда равны 12,4 см, 6 см, 20 см.
6. Разработать и реализовать программу для вычисления массы шара и куба, если известно, что радиус и ребро равны 0,5 м, плотность вещества – 2700 кг/м^3?
7. Разработать и реализовать программу для нахождения времени, через которое будет услышан выстрел охотника, если известно, что расстояние до него составляет 2 км. Скорость звука 3 м/с.
8. Разработать и реализовать программу для определения объема конуса и цилиндра, если известно, что их высота равна 14,75 см, а радиус основания – 8,5 см.
9. Разработать и реализовать программу для нахождения объема цилиндра, если известно, что его масса равна 2 кг, а плотность вещества 19,3 кг/м^3. Чему равен объем при той же массе, но с плотностью вещества 10,5 кг/м^3?
10. Разработать и реализовать программу для определения плотности вещества, из которого сделан брусок. Известно, что длина точильного бруска равна 30 см, ширина – 5 см и толщина – 2 см. Масса бруска 1,2 кг.

Лабораторная работа № 2
Управляющие структуры С++

Теоретический материал: п. 2.

Задание А

1. Компонент ListBox содержит фамилии писателей. Составить программу, позволяющую узнать, в каком жанре работает выбранный писатель.
2. Составить программу, которая при выборе в компоненте ListBox названия месяца, выводит количество дней в выбранном месяце.
3. Компонент ListBox содержит наименование валют. Составить программу, пересчитывающую введенную сумму (в рублях) в выбранную валюту.
4. Компонент ListBox содержит наименование имеющихся в продаже моделей принтеров. Составить программу, выводящую на экран тип выбранного принтера.
5. Компонент ListBox содержит названия правильных многоугольников. Составить программу, вычисляющую площадь выбранного многоугольника по введенной длине стороны.
6. Компонент ListBox содержит названия математических функций. Составить программу, вычисляющую значение функции для введенного аргумента.
7. Компонент ListBox содержит названия стран. Составить программу, выводящую на экран наименование национальной валюты для выбранной страны.
8. Компонент ListBox содержит фамилии художников. Составить программу, позволяющую узнать, в каком жанре работает выбранный художник.
9. Компонент ListBox содержит названия стран. Составить программу, позволяющую узнать государственный язык выбранной страны.
10. Составить программу, которая при выборе в компоненте ListBox названия месяца, выводит соответствующее месяцу время года: «зима», «весна», «лето», «осень».

Задание В

Составить программу вычисления суммы ряда. Значение аргумента и точность вводить с клавиатуры. Вывод результата должен осуществляться с помощью кнопки в компонент Label. Вывести на экран количество просуммированных членов ряда.

1. $e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots$

2. $\ln(x) = (x-1) - \frac{(x-1)^2}{2} + \frac{(x-1)^3}{3} - \frac{(x-1)^4}{4} + \dots$

3. $\cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \dots$

4. $\ln(1-x) = -x - \frac{x^2}{2} - \frac{x^3}{3} - \frac{x^4}{4} - \dots$

5. $\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots$

6. $\mathrm{arctg}(x) = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \dots$

7. $\frac{e^x + e^{-x}}{2} = 1 + \frac{x^2}{2!} + \frac{x^4}{4!} + \frac{x^6}{6!} + \dots$

8. $\ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \dots$

9. $\frac{e^x - e^{-x}}{2} = 1 + \frac{x^3}{3!} + \frac{x^5}{5!} + \frac{x^7}{7!} + \dots$

10. $e^{x^2} = 1 + x^2 + \frac{x^4}{2!} + \frac{x^6}{3!} + \frac{x^8}{4!} + \dots$

Задание С

Разработать программу решения задачи с выводом результата в компонент Label.

1. В интервале от *a* до *b* найти все простые числа. Значения *a* и *b* вводить с клавиатуры.
2. Найдите все трехзначные числа, сумма цифр которых равна заданному числу. Число вводить с клавиатуры.
3. Число Армстронга – это такое число из *k* цифр, для которого сумма *k*-х степеней его цифр равна самому числу. Например, $153 = 1^3 + 5^3 + 3^3$. Найти все числа Армстронга из двух и трех цифр.
4. Найти все трехзначные натуральные числа, в записи которых есть одинаковые цифры.
5. Счастливым будем считать такое число из шести цифр, в котором сумма левых трех цифр равна сумме правых трех цифр. Например,

457961: 4 + 5 + 7 = 9 + 6 + 1. Найти все счастливые билеты и подсчитать их количество (номера билетов от 0 до 999999). Если в числе меньше шести цифр, то недостающие начальные цифры считаются нулями.

6. Дано натуральное число *n*. Среди чисел 1, ..., *n* (*n* £ 99) найти все такие, запись которых совпадает с последними цифрами записи их квадрата (например, 62 = 36, 252 = 625). Значение *n* вводить с клавиатуры.
7. Найти наименьшее общее кратное двух заданных чисел. Числа вводить с клавиатуры.
8. Найти все совершенные числа в интервале от *a* до *b*. Совершенным называется такое натуральное число, которое равно сумме всех своих делителей, за исключением самого числа, например, 28 = 1 + 2 + 4 + 7 + 14. Значения *a* и *b* вводить с клавиатуры.
9. Разбить заданное число на 2 слагаемых всеми различными способами. Разбиения, отличающиеся лишь порядком слагаемых, разными не считать. Число вводить с клавиатуры.
10. Получить все трехзначные натуральные числа, в записи которых нет двух одинаковых цифр.

Лабораторная работа № 3
Разработка MDI-приложений

Теоретический материал: п. 3.

Создать тестовую программу по заданной тематике, которая будет включать в себя меню команд и диалоговые окна: с информацией об авторе теста, инструкцией по использованию программы, вопросами и результатом тестирования:

– по литературе;
– биологии;
– химии;
– русскому языку;
– геометрии;
– географии;
– истории;
– психологии;
– физике;
– художественной культуре.

Примечания.

1. Предполагается, что проект должен содержать заставку, выполненную в отдельной форме.
2. Добавить в проект диалоговое окно, запрашивающее фамилию и имя тестируемого, чтобы эти данные учитывались при выводе результата.
3. Некоторые вопросы теста должны содержать рисунки.
4. Предусмотреть переключение страниц с помощью командной кнопки.

Лабораторная работа № 4
Символы и строки

Теоретический материал: п. 4.

Составить программу обработки строковых величин. Исходные данные для обработки должны вводиться с клавиатуры в объект *Edit*. Обработка должна осуществляться по «щелчку» командной кнопки. Для вывода результатов использовать объект *Label* или окно сообщений.

Задание A

1. Выяснить, является ли данное слово перевертышем.
2. Выяснить, все ли буквы слова X, стоящие на четных местах, различны.
3. Разработать программу обращения слова.
4. Задан список группы студентов с тремя оценками. Фамилии от оценок и оценки друг от друга отделены символом «*». Напечатать список фамилий и средний балл каждого студента на отдельных строках.
5. Определить, какие символы и сколько раз встречаются в тексте.
6. Определить символ, чаще всего встречающийся в слове X.
7. Убрать из слова X повторяющиеся буквы.
8. На какую букву начинается больше слов в тексте Y (между словами может быть несколько пробелов).
9. Составить программу, вычеркивающую из слова X те гласные, что встречаются дважды.
10. Напечатать самое длинное слово из текста (между словами может быть несколько пробелов).

Задание B

1. Составить программу подсчета гласных букв слова X, что используются при написании слова Z.
2. Составить программу, вычеркивающую из слова X те буквы слова Y, что используются при написании слова Z.
3. Составить программу, отыскивающую гласную, чаще всего встречающуюся в слове X.
4. Составить программу, выясняющую, все ли буквы слова X, стоящие на нечетных местах, различны.
5. Составить программу, подсчитывающую, сколько различных согласных слова X употребляется в написании X более одного раза.
6. Составить программу, вычеркивающую из слова X и Y те буквы, что одновременно используются при написании каждого из этих слов.
7. Составить программу, вычеркивающую из слова Y те согласные, которые используются в слове Z.

8. Составить программу, проверяющую, можно ли из букв, входящих в слово X, составить слово Y (буквы можно переставлять и каждую букву можно использовать ровно два раза.)
9. Составить программу, отыскивающую согласную, встречающуюся в слове X реже всего.
10. Составить программу, подсчитывающую, сколько различных гласных слова X употребляется в написании Y более одного раза.

Лабораторная работа № 5
Массивы

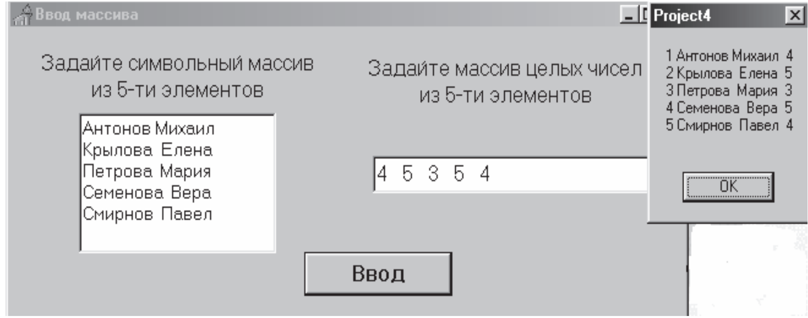
Теоретический материал: пп. 5, 6.

Задание А

I) Создать диалоговое окно, содержащее компоненты для ввода числового и символьного одномерных массивов, состоящих из 12 элементов каждый (по одному из вариантов):

1. Список учеников класса и их оценки по математике.
2. Список товаров на складе и цены на каждый товар.
3. Расписание авиарейсов.
4. Меню ресторана.
5. Учебный план (наименование изучаемых дисциплин и количество часов).
6. Годовой план по выпуску продукции (процентное выполнение по каждому месяцу).
7. Список студентов группы и их возраст.
8. Реклама туристического агентства (маршруты и цены).
9. Результаты спортивных соревнований.
10. Учет пропущенных занятий (список фамилий студентов и количество пропусков).

II) Написать функцию, позволяющую вывести эти массивы в окно сообщения (окно может выглядеть в соответствии с рисунком, расположенным ниже).



III) Добавить на форму командную кнопку <Сортировка> и разработать процедуру обработки события OnClick этой кнопки, выполняющую сортировку по возрастанию или убыванию элементов числового массива и соответственно переставляющую элементы символьного массива.

Задание В

Разработать программу обработки одномерного массива, использующую модуль функций программиста для ввода элементов (по варианту задания).

1. Создать программу, которая считывает положительное вещественное число Ans, вводит 20 вещественных чисел в одномерный массив, а затем выбирает из них то, которое по абсолютной величине ближе всего к значению Ans.
2. Создать программу, которая вводит 20 вещественных чисел в одномерный массив, подсчитывает среднее арифметическое этих элементов и выбирает элемент, абсолютная величина которого ближе всего к значению среднего арифметического.
3. Создать программу, которая считывает 30 вещественных чисел в два одномерных массива (по 15 в каждый), а затем формирует из них два новых массива с положительными и отрицательными элементами.
4. Создать программу, которая вводит 20 вещественных чисел в одномерный массив, а затем вычисляет так называемую моду этого массива, т. е. число, встречающееся наиболее часто.
5. Создать программу, которая вводит 30 целых чисел в два одномерных массива (по 15 в каждый), а затем подсчитывает, сколько в этих массивах одинаковых элементов, и выводит на экран их значения.
6. Создать программу, которая считывает целое число Ans, вводит 19 целых чисел в одномерный массив, а затем добавляет это число в массив, если оно там отсутствует.
7. Создать программу, которая вводит 20 вещественных чисел в одномерный массив, а затем выполняет циклический сдвиг элементов массива на один элемент (т. е. последний элемент становится первым, первый – вторым, второй – третьим и т. д.).
8. Создать программу, которая считывает целое число Ans, вводит 20 целых чисел в одномерный массив, а затем удаляет из массива элемент, равный по абсолютному значению Ans (если такой элемент в массиве есть).
9. Создать программу, которая вводит 20 целых чисел в одномерный массив и вычисляет сумму элементов, имеющих нечетные значения и встречающиеся в этом массиве более 2-х раз.
10. Создать программу, которая вводит 20 вещественных чисел в одномерный массив и вычисляет произведение неповторяющихся элементов.

Задание С

Составить программу обработки двумерного массива. Для вывода двумерного массива использовать компонент *StringGrid*. Ввод исходных данных в таблицу осуществить с помощью процедуры обработки события *OnActivate*. Обработка данных массива и выдача результатов должны осуществляться по щелчку командной кнопки.

1. На олимпиаде по информатике участникам были предложены для решения 5 задач. Каждая из них оценивалась по пятибалльной системе. Составить программу, использующую двумерный массив и выводящую список участников олимпиады в порядке убывания баллов.
2. В соревнованиях по прыжкам в длину принимает участие 12 спортсменов. Каждый из них имеет 3 попытки. Составить программу, использующую двумерный массив и определяющую лучшую попытку каждого спортсмена. Программа также должна выводить список участников соревнований в порядке занятых мест.
3. Результаты соревнований фигуристов по одному из видов многоборья представлены оценками судей в баллах (от 0 до 6). Места участников распределяются по сумме баллов, которые каждый участник получил у всех судей. Составить программу, использующую двумерный массив и выводящую список участников соревнований в порядке занятых мест.
4. Информация об урожае зерновых по районам для 8 областей (в каждой области 5 районов) представлена в таблице. Составить программу, использующую двумерный массив и определяющую среднюю урожайность по каждой области. Результаты должны быть представлены в порядке возрастания урожайности.
5. Для каждой участницы кросса в таблицу были введены следующие данные: фамилия, город, время старта и время финиша. Составить программу, использующую двумерный массив и выводящую список участниц в порядке возрастания времени забега.
6. Каждый абитуриент при поступлении в институт сдал 4 экзамена, оценивавшихся по пятибалльной системе. Для поступления необходимо было набрать 16 баллов. Составить программу, использующую двумерный массив и выводящую список абитуриентов в порядке убывания проходного балла, причем около каждой фамилии должна стоять отметка «зачислен», «не зачислен».
7. В некоторой компании ведется учет данных, регистрирующих количество часов, отработанных служащими (12 человек) в течение недели. Запись о каждом служащем содержит фамилию, почасовую ставку и семь чисел, представляющих продолжительность работы в каждый из семи дней недели. Составить программу, использующую двумерный массив и находящую суммарное число часов, отработанных каждым служащим за неделю, а также размер зарплаты (произведение отработанного времени на почасовую ставку). Результаты вывести в порядке возрастания заработной платы.
8. Для формирования сборной команды по хоккею предварительно выбрано 16 игроков. На основании протоколов игр (всего 6 игр) составлена таблица, в которой содержится штрафное время каждого

игрока по каждой игре (штрафное время может составлять 2, 5 или 10 мин). Составить программу, использующую двумерный массив и выводящую список кандидатов в сборную в порядке возрастания суммарного штрафного времени.

9. В некоторой торговой компании ведется учет данных, регистрирующих продажи 8 видов товаров за каждый месяц года. Запись о каждом товаре содержит название, цену за единицу товара и 12 чисел, представляющих количество товаров, проданных в каждом месяце. Составить программу, использующую двумерный массив и находящую суммарное количество товаров по каждому наименованию, проданных за год, а также их общую стоимость (произведение количества проданных за год товаров на цену за единицу товара). Результаты вывести в алфавитном порядке наименований товара.
10. Фамилии абитуриентов и их оценки после сдачи двух экзаменов при поступлении в институт представлены в виде таблицы. Исключив из этого списка фамилии абитуриентов, получивших неудовлетворительную оценку на втором экзамене, напечатать список абитуриентов в порядке убывания их суммы баллов.

Лабораторная работа № 6
Создание приложения обработки однотабличной базы данных

Теоретический материал: пп. 7, 8.

Задание A

1. Открыв программу «*BDE Administrator*», создать собственный псевдоним (область) для собственных баз данных.
2. Согласно варианту задания разработать базу данных (не менее 4-х таблиц).
3. Выбрать одну из основных таблиц и реализовать ее средствами *DataBase Desktop* (создать структуру таблицы, описать каждое поле и заполнить таблицу записями). Ввести в таблицу не менее 5 записей.

Примечания.

1. При разработке таблиц обратить внимание на поля, для которых необходимо создать маски ввода.

2. Необходимо, чтобы в базе данных (хотя бы в одном экземпляре) были поля типа OLE, Memo и Logical.

Варианты заданий

1. Железнодорожная касса
2. Аптечный склад
3. Фирма недвижимости
4. Школьный администратор
5. Касса Аэрофлота
6. Библиотека
7. Касса магазина
8. Деканат
9. Кадры (сотрудники)
10. Архив
11. Станция техобслуживания
12. Паспортный стол
13. Видеотека
14. Музыкальная коллекция
15. Спортивная секция

Задание B

Разработать форму просмотра информации из однотабличной базы данных.

Лабораторная работа № 7
Разработка многотабличной базы данных

Теоретический материал: пп. 9, 10.

Задание А

Разработать многотабличную базу данных согласно варианту задания. Таблиц должно быть не менее 3-х.

Задание В

1. Изучить теоретический материал и проанализировать области приложения, в которых можно использовать возможности, описанные в данном пункте.
2. Разработать формы просмотра данных согласно заданию.
3. Дополнить уже существующую структуру приложения дополнительными элементами:
   a) добавить графические изображения в таблицы;
   b) выполнить просмотр данных с использованием дополнительного окна («вывести» редко используемую информацию на дополнительную форму просмотра);
   c) добавить в формы просмотра ниспадающие списки для удобства и легкости ввода данных;
   d) добавить в просмотр вычисляемое поле;
   e) добавить на формы необходимые маски ввода;
   f) реализовать форму просмотра трех таблиц через компонент *DBCtrlGrid*;
   g) добавить при необходимости различные виды диалоговых окон.
4. Этап реализации – согласно разработанным и измененным блокам информационной системы реализовать формы просмотра, выполнив все условия задания (условия a-g).

Лабораторная работа № 8
Реализация основных функций приложения

Теоретический материал: п. 11.

Задания

1. Изучить теоретический материал и проанализировать области приложения, в которых можно использовать возможности, описанные в данном пункте.
2. Разработать формы добавления, удаления и модификации согласно заданию.
3. Разработать функцию, которая является основной линией приложения (например, продажа, покупка, сдача, выдача и т. п.).
4. Дополнить уже существующую структуру приложения дополнительными элементами – формами, выполняющими основные функции: добавление, удаление, модификация и поиск (который может быть включен в описанные процедуры).
5. Согласно разработанным и измененным блокам информационной системы реализовать формы добавления, удаления и модификации, выполнив все условия задания.

Лабораторная работа № 9
Разработка запросов приложения

Теоретический материал: п. 12.

Разработать и реализовать несколько запросов к вариативной базе данных (с учетом специфики создаваемого приложения).

1. Определить, какая функция будущего приложения может быть разработана через многофункциональной запрос, а затем реализовать ее.

   Например, разработана форма поиска данных.

   На данной форме присутствуют поля для ввода большого количества критериев. Пользователь вводит критерий (или критерии) только в определенные (а не во все!) поля. В данной ситуации целесообразно разработать запросы, которые будут выполняться, например, от нажатия на кнопку или переключатель (или другой компонент формы). Запрос должен обрабатывать как данные из одной таблицы, так и данные из двух таблиц.
2. Разработать и реализовать два запроса с использованием агрегированных функций.
3. Разработать и реализовать запросы по использованию даты, сравнения строковых значений, а также использования списка значений (отношения BETWEEN, IN).
4. Разработать и реализовать два динамических запроса согласно вариативному заданию.

Замечание. *Все разрабатываемые запросы обязательно должны соответствовать основной функции будущего приложения. Система не должна содержать операции, которые не дают существенных результатов.*

Лабораторная работа № 10
Разработка отчётов приложения

Теоретический материал: п. 13.

Разработать несколько отчётов по вариативной базе данных, предварительно определив, какие виды отчётов необходимы для приложения.

1. Разработать и реализовать простой отчёт (из одной таблицы).
2. Разработать и реализовать отчёт из двух взаимосвязанных таблиц. Разработать и реализовать отчёт с использованием группировки.

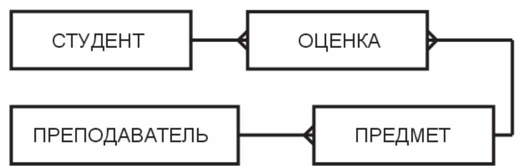
БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Borland C++Builder 6. – [Электронный ресурс]. – http://www.interface.ru/borland/cb5main.htm
2. Архангельский, А.Я. C++Builder 6 : справ. пособие. Кн. 1. Язык C++ / А.Я. Архангельский. – М. : Бином-Пресс, 2004. – 544 с.
3. Архангельский, А.Я. C++Builder 6 : справ. пособие. Кн. 2. Классы и компоненты / А.Я. Архангельский. – М. : Бином-Пресс, 2004. – 548 с.
4. Архангельский, А.Я. Приемы программирования в Delphi / А.Я. Архангельский. – 2-е изд., перераб. и доп. – М. : Бином-Пресс, 2004. – 848 с.
5. Архангельский, А.Я. Программирование в C++Builder 6 / А.Я. Архангельский. – М. : Изд-во БИНОМ, 2004. – 1152 с.
6. Боровский, А.Н. C++ и C++Builder : самоучитель / А.Н. Боровский. – СПб. : Питер, 2005. – 256 с.
7. Дворжецкий, А.В. SQL: Structured Query Language : руководство пользователя / А.В. Дворжецкий. – М. : Познавательная книга плюс, 2001. – 416 с.
8. Ермолаев, В. C++Builder: Книга рецептов / В. Ермолаев, Т. Сорока. – М. : КУДИЦ-ОБРАЗ, 2006. – 208 с.
9. Павловская, Т.А. C/C++. Программирование на языке высокого уровня / Т.А. Павловская. – СПб. : Питер, 2003. – 461 с.
10. Пахомов, Б.И. Interbase и C++Builder на примерах / Б.И. Пахомов. – СПб. : БХВ-Петербург, 2006. – 288 с.
11. Холингворт, Д. Borland C++Builder 6 : руководство разработчика / Д. Холингворт [и др.]; – пер. с англ. – М. : Изд. дом «Вильямс», 2003. – 976 с.
12. Элджер, Д. C++: библиотека программиста / Д. Элджер. – СПб. : Питер, 2001. – 320 с.

Приложение 1

Описание базы данных «Деканат»

Структура базы данных «Деканат»



Спецификация таблицы **student.db**

№ п/п	Имя поля	Тип поля (длина)	Комментарий
1.	Nomer	(S) Short	Уникальный номер студента
2.	Fam	(A) Alpha (20)	Фамилия студента
3.	Nam	(A) Alpha (15)	Имя студента
4.	Otch	(A) Alpha (30)	Отчество студента
5.	Data_rog	(D) Date	Дата рождения студента
6.	Adres	(A) Alpha (50)	Домашний адрес студента
7.	Phone	(A) Alpha (8)	Домашний телефон (формат 00-00-00)
8.	Photo	(O) OLE	Фотография студента (ввод данных в это поле возможно только при работе с приложением в C++Builder)
9.	Gruppa	(A) Alpha (6)	Номер группы (формат АА-000)
10.	Spez	(A) Alpha (6)	Номер специальности (формат 000000)
11.	Prim	(M) Memo (240)	Примечание – дополнительная информация (ввод данных в это поле возможно только при работе с приложением в C++Builder)
12.	Haract	(M) Memo (240)	Примечание – характеристика (ввод данных в это поле возможно только при работе с приложением в C++Builder)
13.	Family	(L) Logical	Наличие семьи у студента («да» – наличие семьи, «нет» – семья отсутствует)

Спецификация таблицы **ozenka.db**

№ п/п	Имя поля	Тип поля (длина)	Комментарий
1.	Nomer	(S) Short	Уникальный номер студента – необходим для связи с родительской таблицей СТУДЕНТ
2.	Kod_predm	(S) Short	Код предмета, необходимый для связи с таблицей ПРЕДМЕТ
3.	Dat_ekz	(D) Date	Дата экзамена
4.	Ozenka	(S) Short	Оценка за экзамен; необходимо установить минимальное значение – 2, максимальное значение – 5, значение по умолчанию – 2
5.	Zachet	(S) Short	Зачет обозначается цифрой 0 или 1. Необходимо также установить минимальное значение (0), максимальное значение (1) и значение по умолчанию (0)

Спецификация таблицы **predmet.db**

№ п/п	Имя поля	Тип поля (длина)	Комментарий
1.	Kod_predm	(S) Short	Уникальный номер, характеризующий предмет – необходим для связи с таблицей ОЦЕНКА
2.	Predmet	(A) Alpha (50)	Название дисциплины
3.	Semestr	(S) Short	Номер семестра. Ограничение – от 1 до 9
4.	Zachet	(A) Alpha (1)	Наличие зачета: 1 – есть, 0 – нет
5.	Ekzamen	(A) Alpha (1)	Наличие экзамена: 1 – есть, 0 – нет
6.	Kod_pp	(S) Short	Код преподавателя – необходим для связи с родительской таблицей ПРЕПОДАВАТЕЛЬ

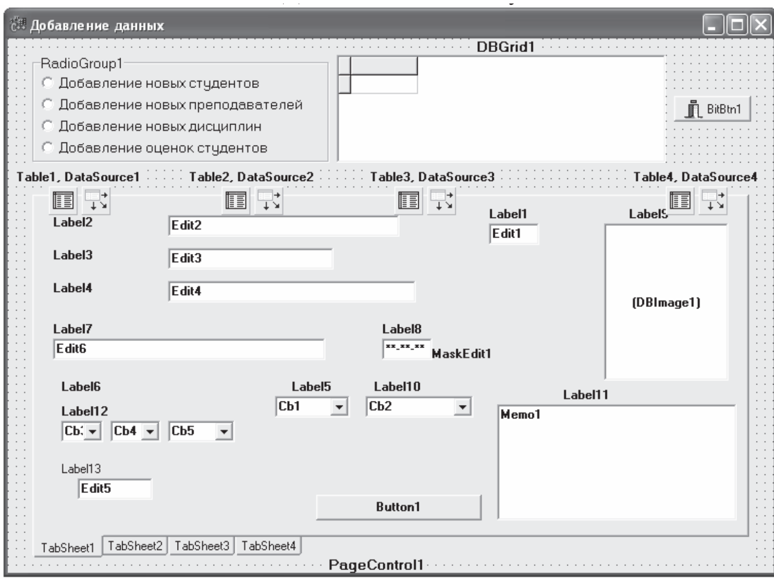
Спецификация таблицы **prepodav.db**

№ п/п	Имя поля	Тип поля (длина)	Комментарий
1.	Kod_prep	(S) Short	Уникальный номер, характеризующий предмет – необходим для связи с таблицей ПРЕДМЕТ
2.	Famil	(A) Alpha (20)	Фамилия преподавателя с инициалами
3.	Kafedra	(A) Alpha (30)	Название кафедры

Приложение 2

Форма «Добавление данных» в конструкторе форм

Вкладка «Добавление новых студентов»



Вкладка «Добавление новых дисциплин»

Приложение 3

Форма «Удаление данных» в конструктуре форм

Приложение 4

Описание некоторых пиктограмм палитры компонентов

Пиктограммы	Вкладка	Обозначение	Описание
	Standard	***Label***	Текстовая информация
	Standard	***Edit***	Однострочное поле для ввода данных
	Standard	***Button***	Кнопка
	Standard	***ListBox***	Список с возможностью выбора нескольких элементов
	Standard	***MainMenu***	Компонент меню, позволяющий оформить подпункт меню с закреплением за ним конкретного действия
	Standard	***Memo***	Многостраничное поле ввода
	Standard	***Panel***	Панель-контейнер
	Standard	***RadioGroup***	Группа радиокнопок для выбора
	Additional	***Image***	Графический компонент
	Additional	***BitBtn***	Кнопка, за которой автоматически закреплена та или иная функция

СОДЕРЖАНИЕ

ВВЕДЕНИЕ 3

1. ОСНОВНЫЕ ПРИНЦИПЫ И ПРИЁМЫ РАБОТЫ В СРЕДЕ РАЗРАБОТКИ ПРОГРАММ BORLAND C++BUILDER 6.0 5

1.1. Начальные сведения о Borland C++Builder 6.0 5
1.2. Элементы интерфейса Borland C++Builder 5
1.3. Пример создания проекта «Вычисление силы тока» 10

2. УПРАВЛЯЮЩИЕ СТРУКТУРЫ C++ 15

2.1. Реализация выбора 15
2.2. Реализация циклов 23

3. ПРИНЦИПЫ И ПРИЁМЫ СОЗДАНИЯ ПРОГРАММЫ, УПРАВЛЯЕМОЙ СОБЫТИЯМИ 31

3.1. Разработка главной формы «Тест по информатике» 31
3.2. Создание окон «О программе», «Инструкция» 32
3.3. Разработка формы «Тестирование» 35
3.4. Разработка формы «Результат» и завершение проекта 37

4. СИМВОЛЫ И СТРОКИ 40

4.1. Символы 40
4.2. Массивы символов 40
4.3. Тип срок AnsiString 43
4.4. Примеры использования строковых значений 48

5. ТИПОВЫЕ ДЕЙСТВИЯ С МАССИВАМИ ДАННЫХ 52

5.1. Объявление массива 52
5.2. Вывод элементов массива на экран 53
5.3. Ввод элементов массива 54

6. МНОГОМЕРНЫЕ МАССИВЫ 59

7. СОЗДАНИЕ ОДНОТАБЛИЧНОЙ БАЗЫ ДАННЫХ 64

7.1. Организация связи с базами данных в C++Builder 64
7.2. Типы данных в среде C++Builder 6.0 65
7.3. Создание новой таблицы с помощью Database Desktop 66
7.4. Использование маски для строковых полей 69
7.5. Создание и редактирование псевдонимов баз данных 70

8. ВИЗУАЛЬНЫЕ КОМПОНЕНТЫ ДЛЯ РАБОТЫ С БАЗОЙ ДАННЫХ 74

8.1. Компоненты формы, работающие с базой данных 74
8.2. Разработка формы просмотра данных таблицы 74
8.3. Использование навигатора для работы с данными 76
8.4. Установка подписей полей в компоненте DBGrid 79

9. СОЗДАНИЕ МНОГОТАБЛИЧНОЙ БАЗЫ ДАННЫХ 81
9.1. Установка индексов для связывания таблиц 81
9.2. Разработка формы просмотра данных двух таблиц 84
10. ДОПОЛНИТЕЛЬНЫЕ ВОЗМОЖНОСТИ ПО РАБОТЕ С БАЗАМИ ДАННЫХ 88
10.1. Ввод данных в поля типа OLE и Memo 88
10.2. Вычисляемые поля. Ниспадающий список 90
10.3. Использование маски ввода 94
10.4. Дополнительные возможности просмотра данных 97
10.5. Диалоговые окна 104
11. РЕАЛИЗАЦИЯ ОСНОВНЫХ ФУНКЦИЙ ПРИЛОЖЕНИЯ 112
11.1. Разработка пользовательского навигатора 112
11.2. Поиск данных в проекте 114
11.3. Фильтрация данных 120
11.4. Добавление данных 125
11.5. Удаление данных 135
11.6. Программирование базы данных 138
12. РАЗРАБОТКА ЗАПРОСОВ К БАЗЕ ДАННЫХ 141
12.1. Введение в язык SQL 141
12.2. Оператор выбора SELECT 141
12.3. Операции с записями 148
12.4. Операции с таблицами и индексами 149
12.5. Компонент Query 151
12.6. Динамические запросы 154
12.7. Связывание таблиц 157
13. СОЗДАНИЕ ОТЧЁТОВ 160
13.1. Компонент QuickRep 160
13.2. Разработка простых отчётов 162
13.3. Разработка отчётов по взаимосвязанным таблицам 165
ЛАБОРАТОРНЫЕ РАБОТЫ 171
БИБЛИОГРАФИЧЕСКИЙ СПИСОК 187
ПРИЛОЖЕНИЯ 188

Учебное издание

Светлана Васильевна Лаптева

ВИЗУАЛЬНОЕ ПРОГРАММИРОВАНИЕ В СРЕДЕ BORLAND C++BUILDER 6.0

Учебное пособие

Редактор *В.С. Павлова*
Технический редактор *З.М. Малявина*
Компьютерная вёрстка *И.И. Шишкиной*
Дизайн обложки *И.И. Шишкиной*

Подписано в печать 17.03.2008. Формат 60x84/16.
Печать оперативная. Усл. п. л. 12,25. Уч.-изд. л. 11,4.
Тираж 100 экз. Заказ № 1-10-08.

Тольяттинский государственный университет
445667, г. Тольятти, ул. Белорусская, 14